12th Symposium on Languages, Applications and Technologies

SLATE 2023, June 26–28, 2023, Vila do Conde, Portugal

Edited by

Alberto Simões
Mario Marcelo Berón
Filipe Portela

OASICS

Editors

Alberto Simões
2Ai, School of Technology, Polytechnic Institute of Cávado and Ave (IPCA), Barcelos, Portugal
asimoes@ipca.pt

Mario Marcelo Berón
Departamento de Informática, Facultad de Ciencias Física Matemáticas y Naturales (FCFMyN),
Universidad Nacional de San Luis, Argentina
mberon@unsl.edu.ar

Filipe Portela
Centro Algoritmi, Escola de Engenharia, Universidade do Minho, Guimarães, Portugal
cfp@dsi.uminho.pt

ACM Classification 2012
Theory of computation → Formal languages and automata theory; Computing methodologies → Natural language processing; Information systems → World Wide Web

ISBN 978-3-95977-291-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-291-4.

Publication date
August, 2023

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

License
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0):
https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASIcs.SLATE.2023.0

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Daniel Cremers (TU München, Germany)
- Barbara Hammer (Universität Bielefeld, Germany)
- Marc Langheinrich (Università della Svizzera Italiana – Lugano, Switzerland)
- Dorothea Wagner (*Editor-in-Chief*, Karlsruher Institut für Technologie, Germany)

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Preface
Alberto Simões, Mario Marcelo Berón, and Filipe Portela 0:vii

List of Authors
.... 0:ix

Committees
.... 0:xi

Papers

Question Answering over Linked Data with GPT-3
Bruno Faria, Dylan Perdigão, and Hugo Gonçalo Oliveira 1:1–1:15

A Framework for Fostering Easier Access to Enriched Textual Information
Gabriel Silva, Mário Rodrigues, António Teixeira, and Marlene Amorim 2:1–2:14

A Pseudonymization Prototype for Hungarian
Attila Novák and Borbála Novák 3:1–3:10

Generating and Ranking Distractors for Multiple-Choice Questions in Portuguese
Hugo Gonçalo Oliveira, Igor Caetano, Renato Matos, and Hugo Amaro 4:1–4:9

Web of Science Citation Gaps: An Automatic Approach to Detect Indexed but Missing Citations
David Rodrigues, António L. Lopes, and Fernando Batista 5:1–5:11

Querying Relational Databases with Speech-Recognition Driven by Contextual Knowledge
Dietmar Seipel, Benjamin Förster, Magnus Liebl, Marcel Waleska, and Salvador Abreu 6:1–6:15

Automatic Speech Recognition of Non-Native Child Speech for Language Learning Applications
Simone Wills, Yu Bai, Cristian Tejedor-García, Catia Cucchiarini, and Helmer Strik 7:1–7:8

OCRticle - a Structure-Aware OCR Application
Sofia G. Rodrigues dos Santos and J. João Dias de Almeida 8:1–8:14

Narrative Extraction from Semantic Graphs
Daniil Lystopadskyi, André Santos, and José Paulo Leal 9:1–9:8

Large Language Models: Compilers for the 4th Generation of Programming Languages?
Francisco S. Marcondes, José João Almeida, and Paulo Novais 10:1–10:8

Hierarchical Data-Flow Graphs
José Pereira, Vitor Vieira, and Alberto Simões 11:1–11:9

Type Annotation for SAST
Marco Pereira, Alberto Simões, and Pedro Rangel Henriques 12:1–12:13

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Characterization and Identification of Programming Languages
Júlio Alves, Alvaro Costa Neto, Maria João Varanda Pereira, and Pedro Rangel Henriques 13:1–13:13

Towards a Universal and Interoperable Scientific Data Model
João Oliveira, Diogo Gomes, Francisca Santana, Jorge Oliveira e Sá, and Filipe Portela 14:1–14:16

Integrating Gamified Educational Escape Rooms in Learning Management Systems
Ricardo Queirós, Carla Pinto, Mário Cruz, and Daniela Mascarenhas 15:1–15:8

Romaria De N$^{\underline{a}}$ Sr$^{\underline{a}}$ D'Agonia: Building a Digital Repository and a Virtual Museum
Sara Cristina Freitas Queirós, Cristiana Araújo, and Pedro Rangel Henriques 16:1–16:16

Preface

In the realm of communication, language serves as the foundation pillar. Within the domain of computer science, three distinct modes of communication have emerged: Human to Human Languages (HHL), Human to Computer Languages (HCL), and Computer-to-Computer Languages (CCL). Each of these modes possesses its unique attributes and significance. HHL encompasses the languages humans employ to interact and converse, constituting the realm of natural language processing. For the HCL, programming languages enable humans to articulate their knowledge and issue commands that instruct computers' behaviour. Finally, CCL assumes its role as an inter-computer communication language. Particularly noteworthy is the revolutionary transformation in the communication process, especially within HCL, driven by the emergence of generative languages that facilitate using natural language to communicate with computers.

The pursuit of understanding and advancing these three types of communication means has been guiding the SLATE conference since its inception. The twelfth edition culminated at the School of Media Arts and Design (P. Porto, ESMAD) in Vila do Conde, Portugal, from the 26th to the 28th of July. SLATE serves as a nexus where researchers, developers, and educators converge, facilitating the exchange of ideas and knowledge across three tracks: the Human-to-Human Language (HHL) track, the Human-to-Computer Language (HCL) track, and the Computer-to-Computer Language (CCL) track.

Within the context of this edition, we received twenty-one papers. The rigorous review process realized by the Scientific Committee led to the selection of sixteen articles for publication and oral presentation during the symposium.

We thank the individuals and groups instrumental in the resounding success of the 2023 edition of SLATE. The dedication of the Members of the Scientific Program Committee is deeply appreciated; their meticulous efforts in reviewing submissions, providing insightful corrections, and contributing novel ideas were pivotal in shaping the final roster of accepted papers. The Organizing Committee equally deserves acknowledgement for their invaluable contributions in ensuring the seamless execution of the conference. Our sincere appreciation extends to the invited Speaker, Gerardo Sierra, the Head of the Language Engineering Group at Universidad Nacional Autónoma de México (UNAM), for graciously accepting our invitation and generously sharing their expertise.

Finally, we highly esteem the Authors, whose unwavering commitment and scholarly contributions enriched SLATE with their ongoing projects and research pursuits. Among these outstanding contributions, special recognition for the work by Bruno Faria, Dylan Perdigão and Hugo Gonçalo Oliveira, entitled "Question Answering over Linked Data with GPT-3," which awarded the distinction of the Best Paper Award.

After SLATE 2023, it becomes manifest that the convergence of these myriad linguistic dimensions fosters a harmonious symphony of knowledge and innovation, enabling profound dialogue between humans and computers. Embrace the wealth of insights within this compendium of proceedings and allow your curiosity to flourish.

Alberto Simões, Mario Marcelo Berón, and Filipe Portela

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Salvador Abreu (6)
Nova-Lincs, University of Évora, Portugal

José João Almeida (10)
ALGORITMI Research Centre/LASI,
University of Minho, Braga, Portugal

Júlio Alves (13)
ALGORITMI Research Centre/LASI,
University of Minho, Braga, Portugal

Hugo Amaro (4)
Instituto Pedro Nunes, LIS, Coimbra, Portugal

Marlene Amorim (2)
GOVCOPP, DEGEIT, University of Aveiro,
Portugal

Cristiana Araújo (16)
ALGORITMI Research Centre / LASI,
DI-University of Minho, Braga, Portugal

Yu Bai (7)
Radboud University, Nijmegen, The Netherlands;
NovoLearning, Nijmegen, The Netherlands

Fernando Batista (5)
Iscte - University Institute of Lisbon, Portugal;
INESC-ID Lisbon, Portugal

Igor Caetano (4)
Instituto Pedro Nunes, Coimbra, Portugal;
Department of Informatics Engineering,
University of Coimbra, Portugal

Alvaro Costa Neto (13)
Federal Institute of Education, Science and
Technology of São Paulo, Barretos, Brazil

Mário Cruz (15)
School of Education & inED,
Polytechnic Institute of Porto, Portugal

Catia Cucchiarini (7)
Radboud University, Nijmegen, The Netherlands

J. João Dias de Almeida (8)
ALGORITMI/LASI, University of Minho,
Braga, Portugal

Bruno Faria (1)
Department of Informatics Engineering,
University of Coimbra, Portugal;
Centre for Informatics and Systems of the
University of Coimbra, Portugal

Benjamin Förster (6)
Department of Computer Science,
Universität Würzburg, Germany

Diogo Gomes (14)
University of Minho, Guimarães, Portugal

Hugo Gonçalo Oliveira (1, 4)
Department of Informatics Engineering,
University of Coimbra, Portugal;
Centre for Informatics and Systems of the
University of Coimbra, Portugal

Pedro Rangel Henriques (12, 13, 16)
ALGORITMI Research Centre/ LASI,
DI-University of Minho, Braga, Portugal

José Paulo Leal (9)
CRACS & INESC TEC, Porto, Portugal;
Faculty of Sciences, University of Porto,
Portugal

Magnus Liebl (6)
Department of Computer Science,
Universität Würzburg, Germany

António L. Lopes (5)
Instituto de Telecomunicações, Iscte – University
Institute of Lisbon, Portugal

Daniil Lystopadskyi (9)
Faculty of Sciences, University of Porto,
Portugal

Daniela Mascarenhas (15)
School of Education & inED,
Polytechnic Institute of Porto, Portugal

Renato Matos (4)
Center of Informatics and Systems, University of
Coimbra, Portugal;
Department of Informatics Engineering,
University of Coimbra, Portugal

Paulo Novais (10)
ALGORITMI Research Centre/LASI,
University of Minho, Braga, Portugal

Attila Novák (3)
Faculty of Information Technology and Bionics,
Pázmány Péter Catholic University, Budapest,
Hungary

Borbála Novák (3)
Faculty of Information Technology and Bionics,
Pázmány Péter Catholic University, Budapest,
Hungary

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

João Oliveira (14)
University of Minho, Guimarães, Portugal

Dylan Perdigão (1)
Department of Informatics Engineering, University of Coimbra, Portugal;
Centre for Informatics and Systems of the University of Coimbra, Portugal

José Pereira (11)
Checkmarx, Braga, Portugal

Marco Pereira (12)
Checkmarx, Braga, Portugal;
University of Minho, Braga, Portugal

Maria João Varanda Pereira (13)
Research Centre in Digitalization and Intelligent Robotics, Polythechnic Insitute of Bragança, Portugal

Carla Pinto (15)
School of Engineering & CMUP, Polytechnic Institute of Porto, Portugal

Filipe Portela (14)
Algoritmi Centre, University of Minho, Guimarães, Portugal;
IOTECH – Innovation on Technology, Trofa, Portugal

Ricardo Queirós (15)
School of Media Arts and Design & CRACS – INESC TEC, Polytechnic Institute of Porto, Portugal

Sara Cristina Freitas Queirós (16)
ALGORITMI Research Centre / LASI, DI-University of Minho, Braga, Portugal

Sofia G. Rodrigues dos Santos (8)
Informatics Department, University of Minho, Braga, Portugal

David Rodrigues (5)
Iscte – University Institute of Lisbon, Portugal

Mário Rodrigues (2)
IEETA, ESTGA, University of Aveiro, Portugal;
LASI – Intelligent System Associate Laboratory, Coimbra, Portugal

Francisco S. Marcondes (10)
ALGORITMI Research Centre/LASI, University of Minho, Braga, Portugal

Francisca Santana (14)
University of Minho, Guimarães, Portugal

André Santos (9)
CRACS & INESC TEC, Porto, Portugal;
Faculty of Sciences, University of Porto, Portugal

Dietmar Seipel (6)
Department of Computer Science, Universität Würzburg, Germany

Gabriel Silva (2)
IEETA, DETI, University of Aveiro, Portugal;
LASI – Intelligent System Associate Laboratory, Coimbra, Portugal

Alberto Simões (11, 12)
Checkmarx, Braga, Portugal; 2Ai, School of Technology, IPCA, Barcelos, Portugal

Helmer Strik (7)
Radboud University, Nijmegen, The Netherlands

Jorge Oliveira e Sá (14)
Algoritmi Centre, University of Minho, Guimarães, Portugal

António Teixeira (2)
IEETA, DETI, University of Aveiro, Portugal;
LASI – Intelligent System Associate Laboratory, Coimbra, Portugal

Cristian Tejedor-García (7)
Radboud University, Nijmegen, The Netherlands

Vitor Vieira (11)
Checkmarx, Braga, Portugal

Marcel Waleska (6)
Department of Computer Science, Universität Würzburg, Germany

Simone Wills (7)
Radboud University, Nijmegen, The Netherlands

Committees

Organizing Committee

Alberto Simões
2Ai, School of Technology, IPCA
Barcelos, Portugal
asimoes@ipca.pt

Filipe Portela
Algoritmi, University of Minho
Guimarães, Portugal
cfp@dsi.uminho.pt

Ricardo Queirós
ESMAD, Polytechnic of Porto
Vila do Conde, Portugal
ricardoqueiros@esmad.ipp.pt

Mário Pinto
ESMAD, Polytechnic of Porto
Vila do Conde, Portugal
mariopinto@esmad.ipp.pt

Scientific Committee

Alberto Simões
Portugal

Alvaro Costa Neto
Brasil

Antoni Oliver
Spain

António Leitão
Portugal

António Miguel Rosado da Cruz
Portugal

António Teixeira
Portugal

Bostjan Slivnik
Slovenia

Brett Drury
United Kingdom

Cristina Ribeiro
Portugal

David Martins de Matos
Portugal

Diana Santos
Norway

Dietmar Seipel
Germany

Dušan Kolář
Czech Republic

Fernando Batista
Portugal

Filipe Portela
Portugal

Gergő Balogh
Hungary

Hugo Gonçalo Oliveira
Portugal

Irene Rodrigues
Portugal

Ivan Luković
Serbia

Jakub Swacha
Poland

Jan Janousek
Czech Republic

João Cordeiro
Portugal

João Saraiva
Portugal

José Carlos Paiva
Portugal

José Carlos Ramalho
Portugal

José João Almeida
Portugal

Jose Luis Sierra
Spain

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

José Paulo Leal
Portugal

Josep Silva
Spain

Luís Ferreira
Portugal

Luis Filipe Costa Cunha
Portugal

Luís Matos
Portugal

Luis Morgado Da Costa
Czech Republic

Marcel Waleska
Germany

Maria João Varanda Pereira
Portugal

Mário Beron
Argentina

Mário Pinto
Portugal

Mário Rodrigues
Portugal

Nuno Almeida
Portugal

Pablo Gamallo
Spain

Paulo Matos
Portugal

Pedro Rangel Henriques
Portugal

Renato Preigschadt de Azevedo
Brasil

Ricardo Queirós
Portugal

Ricardo Rocha
Portugal

Ricardo Rodrigues
Portugal

Salvador Abreu
Portugal

Simão Melo de Sousa
Portugal

Teresa Guarda
Portugal

Tomaz Kosar
Slovenia

Vladimir Ivančević
Serbia

Question Answering over Linked Data with GPT-3

Bruno Faria
Department of Informatics Engineering, University of Coimbra, Portugal
Centre for Informatics and Systems of the University of Coimbra, Portugal

Dylan Perdigão
Department of Informatics Engineering, University of Coimbra, Portugal
Centre for Informatics and Systems of the University of Coimbra, Portugal

Hugo Gonçalo Oliveira
Department of Informatics Engineering, University of Coimbra, Portugal
Centre for Informatics and Systems of the University of Coimbra, Portugal

Abstract

This paper explores *GPT-3* for answering natural language questions over Linked Data. Different engines of the model and different approaches are adopted for answering questions in the *QALD-9* dataset, namely: zero and few-shot *SPARQL* generation, as well as fine-tuning in the training portion of the dataset. Answers retrieved by the generated queries and answers generated directly by the model are also compared. Overall results are generally poor, but several insights are provided on using *GPT-3* for the proposed task.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases SPARQL Generation, Prompt Engineering, Few-Shot Learning, Question Answering, GPT-3

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.1

Supplementary Material
Software (Repository): `https://github.com/brunofaria1322/GPT3-over-QALD9`, archived at `swh:1:dir:5c52a1c2df0a799ceee6ac97ea1fe3ff6e056694`

Funding This work was partially supported by the Portuguese Recovery and Resilience Plan (PRR) through project C645008882-00000055, Center for Responsible AI; and by FCT – Foundation for Science and Technology, I.P., within the scope of the project CISUC – UID/CEC/00326/2020 and by the European Social Fund, through the Regional Operational Program Centro 2020. It is also based upon work in COST Action CA18209 Nexus Linguarum, supported by COST (European Cooperation in Science and Technology). `http://www.cost.eu/`.

1 Introduction

The Generative Pre-trained Transformer 3 (*GPT-3*) [7] Language Model (*LM*), developed by *OpenAI*, is known to perform a broad range of Natural Language Processing (*NLP*) and generation tasks, like summarisation, classification, or translation, in a zero or few-shot scenario. However, there is not much work concerning its use for generating Simple Protocol And RDF Query Language (*SPARQL*). This gap, to which access limitations contribute, is the primary motivation for exploring *GPT-3* in this task. We explore this model in the generation of *SPARQL* queries for generic questions in Natural Language (*NL*). Such queries should be able to retrieve answers from Linked Data (*LD*). The advantage of using a Large Language Model (*LLM*) like *GPT-3* is that we are not limited to a Knowledge Base (*KB*) with static finite information. Not that the *LLM*s has infinite information, but it is much more flexible: it can learn, even from only a few examples (i.e., in few-shot learning), and, independently of the quality, will generate outputs for any prompt.

© Bruno Faria, Dylan Perdigão, and Hugo Gonçalo Oliveira;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 1; pp. 1:1–1:15
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On the other hand, *KB* and *LD* are aligned with the FAIR data principles (Findable, Accessible, Interpretable, Reusable) [23], in opposition to black-box *LLM*. Therefore, instead of using *GPT-3* directly for answering questions, a middle-ground would be using this model for generating human-interpretable *SPARQL*, which may then be used for querying *LD*, represented in Resource Description Framework (*RDF*).

For exploring *GPT-3* in this task, we rely on the Question Answering over Linked Data 9 (*QALD-9*) [22] dataset, which has: *NL* questions; *SPARQL* for retrieving their answers from *DBpedia* [2]; and the actual answers retrieved by these queries. Question Answering over Linked Data (*QALD*) is a series of challenges that started in 2011, and are currently in the 10th edition[1]. Questions are available in several languages, but most translations lack the necessary high quality, so we focus on English.

Using *QALD-9*, experiments are conducted for generating *SPARQL* queries for *DBpedia* with *GPT-3*, using different engines (i.e., *text-davinci-002* and *text-davinci-003*) and approaches (i.e., zero-shot, few-shot, fine-tuning). Generated queries are evaluated with BLEU [15] scores. Evaluation is complemented with the F1-score, computed on the results of running the generated queries, and on the answers directly generated by *GPT-3* when the *NL* questions are asked.

Amongst our findings, we highlight that the zero-shot approach generates many invalid *SPARQL* queries and that the queries by the fine-tuned model are the closest to the reference, followed by the few-shot approach. On the other hand, answers retrieved from *DBpedia* with queries by the few-shot approach are comparable to those of the fine-tuned model, which learned from many more examples. Still, the best answers are obtained by asking the *NL* question directly to *GPT-3*, for which the query is not necessary. Despite the insights provided by this exploration of *GPT-3*, overall, all results end up being poor according to the adopted metrics.

The remainder of this paper is organised as follows. Section 2 overviews existing *LLM*s and their use cases in the scope of Question Answering (*QA*). Section 3 highlights essential tools and frameworks for our experimentation. Section 4 describes the adopted methodologies. Section 5 presents the obtained results, further discussed in Section 6. Finally, Section 7 concludes the paper and points to possible future directions.

2 Related Work

Bidirectional Encoder Representations from Transformers (*BERT*) [9] and Generative Pre-trained Transformer (*GPT*) are two of the most popular *LM* based on the Transformer architecture. Among many other tasks, they have both been used for *QA*.

BERT, developed by *Google*, uses only encoder blocks, and can be used for providing contextual word embeddings or fine-tuned for many *NLP* tasks, including Extractive *QA*, as long as data is available. *GPT*, an auto-regressive *LM* developed by *OpenAI*, has only decoder blocks and is mostly used for text generation. However, this is enough for current versions of this model, namely *GPT-3* [7] and the recent *GPT-4* [14], performing a broad range of *NLP* tasks based on text prompts, not requiring fine-tuning (zero and few-shot), which can still be performed for specific applications.

There is much work on automatic *QA*, mainly from unstructured text, often referred to as Information Retrieval (*IR*)-based *QA*. Recent approaches rely on fine-tuning transformers for extractive *QA* [9] or *QA* on the domain of the training data [16].

---

[1] `https://www.nliwod.org/challenge` (accessed on 20/03/23)

Alternatively, knowledge-based *QA* gets answers from a structured *KB*. For this, *NL* questions must be converted to logical constraints or structured queries, e.g., through semantic parsing [6], or, more recently, deep neural networks [8].

When it comes to generating *SPARQL* queries, for *KB* in *RDF*, there are datasets of *NL* questions and their translation to *SPARQL*. These include *LC-QuAD* [20] and the *QALD* [22]. The latter results from a series of challenges, currently in their tenth edition[2].

SparseQA [3] is a framework used for answering complex questions tested in several datasets, including those previously mentioned. It adopts a word-reordering approach for creating and refining a graph based on each question. This encompasses:
(i) the classification of the question type;
(ii) the identification of entities and variables;
(iii) the construction of a graph from the sequential analysis of the question words.
The search space is then reduced by creating a knowledge sub-graph, and an approximate match is performed with the relation pattern-based graph similarity. *SParseQA* was shown to perform better than other systems that generate *SPARQL* with a broad range of approaches, such as: graph traversal [21] and other graph-based [11, 12]; traditional supervised machine learning [4]; parsing [24, 5] and rules on the underlying *KB* semantics [10]; query template learning [22] and pattern recognition [28].

The performance of *SPARQL* generation with *BERT* and *GPT-3* was compared in a *KB* of aviation accident reports [1]. Four models, namely *BM25-BERT* (baseline), *KGQA*, *BERT-QA*, *GPT-3-QA*, and two combinations, *KGQA+BERT-QA* and *KGQA+GPT-3-QA*, were tested. Results were assessed with Exact Match (*EM*), Exact Recall (*ER*), accuracy, and recall. *KGQA+GPT-3-QA* was the best approach in most metrics, which shows the benefits of combining models. Even though *GPT-3-QA* was based on *GPT-3*, it used older engines (*ada* and *curie*) and is focused on aviation reports. There are very recent reports [18] on using *GPT-3* and related models for *QA*, in *QALD* and other datasets. When noting that some of the models have difficulties for generating SPARQL, they focus only on the answer, and report a F1 of 46% (text-davinci-003). In a related task, knowledge-based visual *QA*, the steps of knowledge retrieval and reasoning were unified by prompting *GPT-3*, used implicitly as a *KB* [25].

SPBERT [19] was the first transformer-based *LM* pre-trained on a large quantity of *SPARQL* queries. After fine-tuning, it was tested in four datasets: *QALD-9*, *LC-QuAD*, *Mon* [17] and Verbalization QUestion ANswering DAtaset (*VQuAnDa*) [13] datasets, where it outperformed other approaches that model *SPARQL* generation from *NL* as Neural Machine Translation (*NMT*) [26], with Recurrent Neural Networks (*RNN*s), Convolutional Neural Networks (*CNN*s), or an encoder-decoder Transformer model. Evaluation relied on BiLingual Evaluation Understudy (*BLEU*) [15] and *EM*.

Our work complements existing research with the use of *GPT-3* for *SPARQL* generation. As in other works, *SPARQL* is evaluated with *BLEU* and the retrieved answers with F1-score.

3 Experimentation Setup

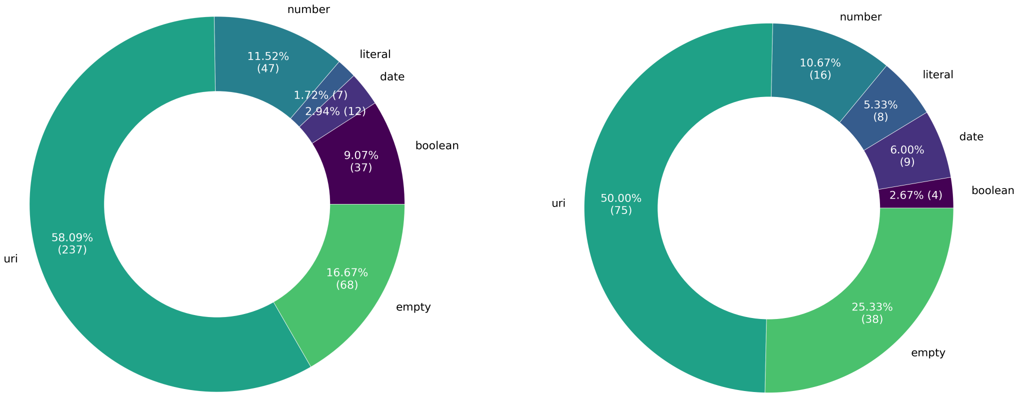
The main tools used in our experiments were:
(i) *QALD-9*, a dataset of *NL* questions and their respective *SPARQL* queries;
(ii) *OpenAI*'s Application Programming Interface (*API*), for text completion with different engines of *GPT-3*;
(iii) *SPARQLWrapper*, for executing *SPARQL* queries and getting their respective results.

---

[2] `https://www.nliwod.org/challenge` (accessed on 20/03/23)

```
-- Boolean
Q: Was Marc Chagall a jew?
A: False

-- Date
Q: When was Olof Palme shot?
A: 1986-02-28

-- Literal
Q: What is the birth name of Angela Merkel?
A: Angela Dorothea Kasner

-- Number
Q: How much is the elevation of Düsseldorf Airport?
A: 44.8

-- URI
Q: What are the specialities of the UNC Health Care?
A: http://dbpedia.org/resource/Cancer; http://dbpedia.org/resource/Trauma_center
```

Figure 1 Five categories of questions with their respective answers.



(a) Train (408 questions) **(b)** Test (150 questions)

Figure 2 Composition of *QALD-9* dataset.

Each entry of the *QALD-9* [22] dataset has:

(i) a *NL* question, in a number of languages;

(ii) the gold *SPARQL* query for getting the answer of the question from *DBpedia* (*2016-10* dump)[3];

(iii) the gold answers to the previous queries.

Answers may belong to one of the following five categories: boolean, date, literal, number, Uniform Resource Identifiers (*URIs*). Figure 1 shows an example question/answer pair for each category. We have only considered their English version. *QALD-9* is split into training and testing portions, each with 408 and 150 questions, respectively. However, we noted that some queries return an empty result due to wrong formatting or to changes in the current version of *DBpedia*. Since these queries did not work, they were discarded for our experimentation. Afterwards, we are left with 340 training and 112 testing questions. Figures 2a and 2b have the distribution of the *QALD-9* dataset, regarding the type of answers.

---

[3] Instead of *DBpedia*, version 9-plus of the dataset includes queries to *Wikidata*

OpenAI offers an *API*[4] for generating any kind of text (e.g., *NL* or code), i.e., the user prompts the model with some text and the model will generate following text. For instance, if the prompt is a question, the model is expected to generate an answer. In our case, the prompt is an instruction for generating a *SPARQL* query, and this is what we expect to be generated. A spectrum of models and engines is available for performing different tasks, with more or fewer capabilities and different prices. These models include: *davinci*, *curie*, *babbage*, or *ada*. We tested two variants of *davinci*, the most powerful for text completion: *text-davinci-002* and *text-davinci-003*. *OpenAI* also allows fine-tuning one of the available engines, which we did with *QALD*'s training set.

For executing *SPARQL* queries on the *DBpedia* endpoint[5], we use *SPARQLWrapper*[6] a Python wrapper for executing *SPARQL* queries, part of *RDFLib*. *SPARQLWrapper* also validates *GPT-3* generated queries. If the query is well-formatted, results are retrieved in a suitable format for further analysis.

4 Methods

This section describes the approaches adopted for testing *GPT-3* in the *QALD* dataset, namely: zero-shot *SPARQL* generation, few-shot *SPARQL* generation, generation with a fine-tuned model, and direct answer generation. All of them are tested in *QALD-9*'s testing data. Evaluation approaches and adopted metrics are also described.

4.1 Zero and Few-Shot

Zero and few-shot were tested in both pre-trained *GPT-3* engines, *davinci-002* and *davinci-003*. These were used with the ten prompts in Table 1, where the ⟨*question*⟩ placeholder is replaced by the questions from the *QALD-9* dataset. The result can be, for example:

- *Turn this into a DBpedia SPARQL query: "What is the time zone of Salt Lake City?"*

Since the *SPARQL* queries in *QALD-9* are meant for *DBpedia*, five prompts refer it specifically and the others do not, for later analysis of the impact of this inclusion. The response of *GPT-3* to these prompts should be a *SPARQL* query. For example, an expected query for the previous question is shown in Figure 6.

Table 1 Prompts tested for getting *SPARQL* queries.

ID	Prompt
Q1	`The SPARQL query for the question "`⟨*question*⟩`" is`
Q2	`What is the SPARQL query for the question "`⟨*question*⟩`"?`
Q3	`SPARQL for "`⟨*question*⟩`" is`
Q4	`Write the complete SPARQL query to answer the question: `⟨*question*⟩
Q5	`Turn this into a SPARQL query: "`⟨*question*⟩`"`
Q6	`The DBpedia SPARQL query for the question "`⟨*question*⟩`" is`
Q7	`What is the DBpedia SPARQL query for the question "`⟨*question*⟩`"?`
Q8	`The DBpedia SPARQL for "`⟨*question*⟩`" is`
Q9	`Write the complete DBpedia SPARQL query to answer the question: "`⟨*question*⟩`"`
Q10	`Turn this into a DBpedia SPARQL query: "`⟨*question*⟩`"`

---

[4] `https://openai.com/api` (accessed on 20/03/23)
[5] `https://dbpedia.org/sparql` (accessed on 20/03/23)
[6] `https://github.com/RDFLib/sparqlwrapper` (accessed on 20/03/23)

The main difference between zero and few-shot relies in the prompts. In zero-shot, they consist of a single *NL* instruction, followed by the *NL* question from *QALD*. The expectation is that *GPT-3* generates the *SPARQL* for the question. In few-shot, the prompt includes a number of instruction-question-*SPARQL* blocks, followed by an instruction-question pair. We only tested five-shot learning, with a prompt illustrated in Figure 3 for the previous example. The five questions for few-shot are selected from the training dataset and include one example from each question category (Figure 1).

```
Turn this into a DBpedia SPARQL query: "What are the specialities of the UNC Health Care?"
SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/UNC_Health_Care>
<http://dbpedia.org/property/speciality> ?uri }

Turn this into a DBpedia SPARQL query: "When was Olof Palme shot?"
SELECT DISTINCT ?date WHERE { <http://dbpedia.org/resource/Olof_Palme>
<http://dbpedia.org/ontology/deathDate> ?date }

Turn this into a DBpedia SPARQL query: "How much is the elevation of Düsseldorf Airport ?"
SELECT ?ele WHERE { <http://dbpedia.org/resource/Düsseldorf_Airport>
<http://dbpedia.org/ontology/elevation> ?ele } LIMIT 1

Turn this into a DBpedia SPARQL query: "Was Marc Chagall a jew?"
ASK WHERE { <http://dbpedia.org/resource/Marc_Chagall>
<http://dbpedia.org/property/ethnicity> \"Jewish\"@en }

Turn this into a DBpedia SPARQL query: "What is the birth name of Angela Merkel?"
SELECT DISTINCT ?string WHERE { <http://dbpedia.org/resource/Angela_Merkel
<http://dbpedia.org/property/birthName> ?string }

Turn this into a DBpedia SPARQL query: "What is the time zone of Salt Lake City?"
```

Figure 3 Prompt for few-shot learning.

4.2 Fine-tuning

Fine-tuning is performed in the custom *davinci* engine with the 340 questions of the *QALD-9* training data. For this purpose, a *JSONL* file is produced (see Figure 4), with each question ending in a “`->`” followed by its *SPARQL* query. To avoid lengthy answers, an end-token (i.e., `\n<EOQ>\n`)) was added after each query.

```
{
    "prompt":
        "List all boardgames by GMT. ->",
    "completion":
        " PREFIX dbo: <http:\/\/dbpedia.org\/ontology\/>
          PREFIX res: <http:\/\/dbpedia.org\/resource\/>
          PREFIX rdfs: <http:\/\/www.w3.org\/2000\/01\/rdf-schema#>
          SELECT ?uri WHERE {
              ?uri dbo:publisher res:GMT_Games
          }\n<EOQ>\n"
}
```

Figure 4 First row of the *JSONL* file containing the pre-processed dataset.

4.3 Hyperparameters

The following hyperparameters were set for all experiments: `temperature`, `max_tokens`, `top_p`, `frequency_penalty`, `presence_penalty`. The `temperature` controls the randomness of the string completion and is set to 0 to avoid randomness. The maximum number of `tokens` is `max_tokens` and is set to twice the length of the expected answer L_{EA} from *QALD*.

The `top_p` controls diversity via nucleus sampling (e.g., 0.5 means that half of all likelihood-weighted options are considered). Finally, `frequency_penalty` and `presence_penalty` are both set to 0. The former penalises new tokens based on their existing frequency in the text so far, and the latter penalises new tokens based on whether they have appeared in the text so far.

4.4 Direct Answer

The final approach does not involve *SPARQL* generation. It consists of making the *NL* question directly to the model, with the gear of finally comparing the generated answer with the query answers in *QALD*. Due to cost limitations, only *davinci-002* was used for this.

To evaluate the model's performance, and since the answers in the dataset are frequently *URI*s in *DBpedia*, the first step was to convert *URI* to text. For this, *DBpedia* is queried for a textual representation of the resource through the value of its `rdfs:label` or, if not available, of its `foaf:name`. If none is available, the *URI* is parsed, and its final part (i.e., past the last `/`) is extracted, with `_` replaced by white spaces. When the answer is a list of *URI*s, the previous steps are applied to each *URI*, and the results are joined in a single string, separated by white spaces.

The last step of this process is to normalise answers from the dataset and by *GPT-3*. This involves converting numbers and dates to a textual format, removing punctuation and stopwords[7], converting special characters (e.g., accents, cedillas) to *ASCII*, and lowercasing everything. The result of this process is illustrated in Figure 5.

```
-- Original Answer
08/01/2020 was a good day to visit Monção, Portugal, with my 2 dogs.

-- Normalised Answer
01 august 2020 good day visit moncao portugal two dogs
```

Figure 5 Answer Normalisation.

```
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT DISTINCT ?uri WHERE {
    res:Salt_Lake_City <http://dbpedia.org/ontology/timeZone> ?uri
}
```

Figure 6 Expected *SPARQL* query for the question "*What is the time zone of Salt Lake City?*".

For evaluation against the answers in *QALD*, the same normalisation was performed on the results retrieved by the generated *SPARQL*.

4.5 Metrics

Two approaches were adopted for evaluating generated *SPARQL* queries:

(i) comparison with the gold *SPARQL* queries in the dataset;
(ii) comparison of their answers, i.e., results retrieved from *DBpedia* by the generated query with the actual answer in the dataset.

---

[7] We considered the list of English stopwords from NLTK, `https://www.nltk.org/` (accessed on 20/03/23)

Answers generated by *GPT-3*, when asked the question directly, were also compared with the answers in *QALD-9*.

Since *EM* would be too strict, as in related work, we rely on *BLEU*[8] for comparing how close two queries are. This has in mind that some queries might be invalid due to simple syntactic errors that a human could quickly fix. *BLEU* is typically used in Machine Translation, in our case, of English to *SPARQL*. It compares the gold answer with the generated one and measures the weighted geometric average of all modified n-grams precision (p_n). Different values of n originate different variants of *BLEU*, such as *BLEU*-1 for unigrams and *BLEU*-2 for bigrams. We report on *BLEU*-1, *BLEU*-2, and a combined measure, Sentence-*BLEU*, which averages *BLEU*-1, 2, 3 and 4.

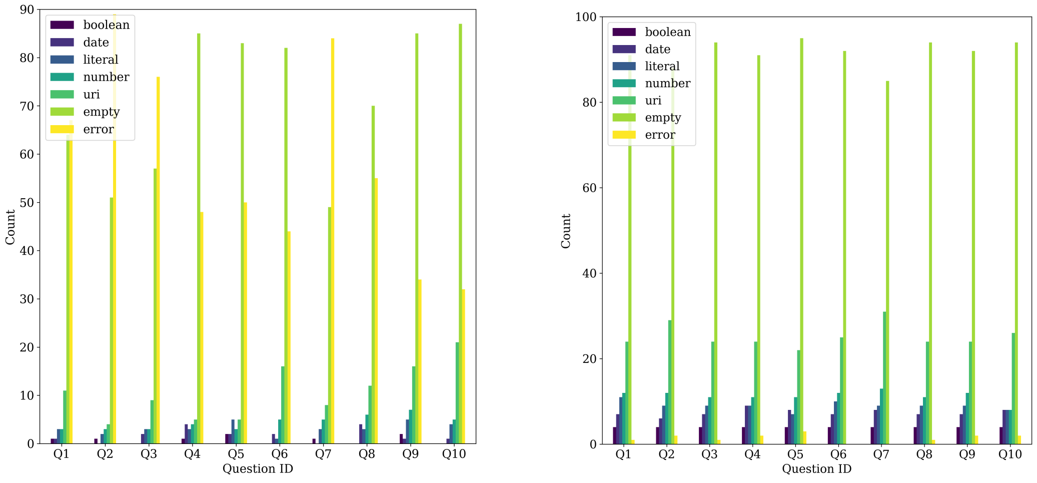
As in the *QALD* challenge, typical *IR* measures, i.e., precision, recall and F1-score, are computed for comparing generated and retrieved answers with the gold answers. When used for assessing the direct answers, their normalisation is performed (see Section 4.4).

5 Evaluation

After analysing the type of the generated queries, this section reports on the evaluation of *SPARQL* queries generated with the three methods, against the gold queries, followed by the evaluation of their results, and of the direct questions, against the gold results.

5.1 Analysis of Generated Query Types

The proportion of valid queries is an initial insight into how *GPT-3* can be used for *SPARQL* generation. Figure 7a shows a distribution of answer types, including invalid queries and empty answers, for queries generated when *QALD* test questions are concatenated to the prompts in Table 1. Results are similar for each engine, so we present them only for *davinci-002*. There are many invalid queries (yellow bar) with zero-shot, but most errors are fixed in the few-shot scenario. However, the increase in valid answer types comes at the cost of an increase in empty answers.

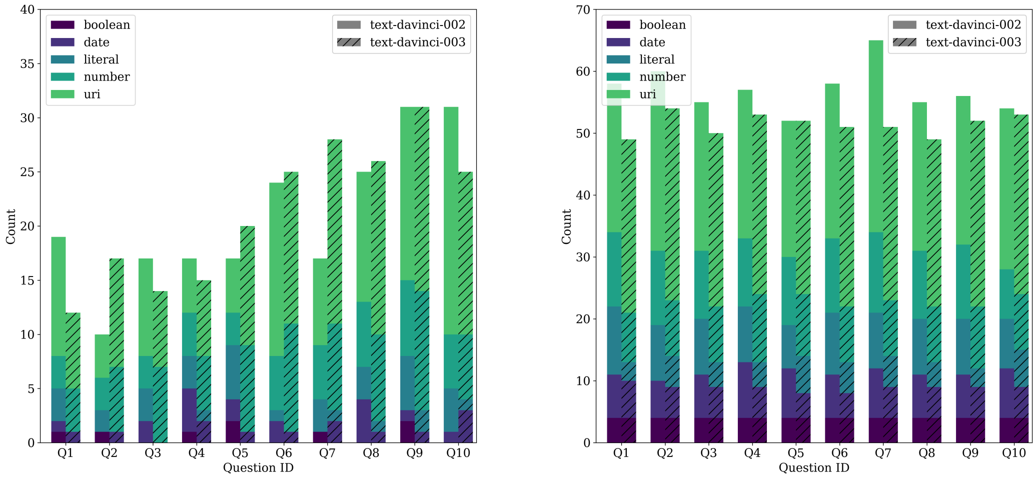


(a) zero-shot. **(b)** few-shot.

Figure 7 All *SPARQL* queries generated by *text-davinci-002*.

---

[8] We have used the *BLEU* implementation of NLTK, `https://www.nltk.org/_modules/nltk/translate/bleu_score` (accessed on 10/05/23)



(a) zero-shot.

(b) few-shot.

Figure 8 Valid *SPARQL* queries generated by each engine.

Figure 8 does the same analysis after removing empty and error queries. For each prompt, two columns are presented, one for each engine. For zero and few-shot, more valid queries can be generated with *davinci-002* than with *davinci-003*.

5.2 Evaluation of Generated SPARQL

BLEU-1, 2, 3 and 4, as well as Sentence-*BLEU* were computed for the *SPARQL* generated for the *QALD* test questions with each prompt, approach and engine, as well as with the fine-tuned model. Table 2 reports on the average *BLEU*-1, *BLEU*-2, and Sentence-*BLEU*. Besides considering the full gold query, we also report the scores when the declaration of prefixes is ignored not only in the gold query but also in the generated one. This has in mind that these declarations are not always necessary. For instance, standard prefixes like `rdf`, or `dbp` and `dbr` for *DBpedia*, are often preloaded by *SPARQL* endpoints.

When considering prefixes, differences between *davinci-002* and *davinci-003* and between different prompts are minimal. Referring *DBpedia* specifically on the prompt also seems to make no difference. When prefix declarations are ignored, performance improves. In this case, the few-shot approach performs better than the zero-shot. Still, low *BLEU*-2 and Sentence-*BLEU* scores suggest that generated queries lack consistency and that *GPT-3* is not suitable for *SPARQL* generation, neither in a zero nor in a few-shot approach.

Despite being far from perfect, the best performance for every metric is achieved by the fine-tuned model. To some extent, this was expected, because this approach was trained in more data (340 examples), and confirms the benefits of fine-tuning.

5.3 Evaluation of SPARQL Results

A different perspective is given by running the generated queries in *DBpedia* and comparing the obtained results with the gold results in *QALD-9*. This is not immune to changes in *DBpedia* because, due to hardware limitations, we queried its most recent version through its public *SPARQL* endpoint, and not the source dump of the dataset, and we know that some answers are only valid in a specific time frame (e.g., *Who is the mayor of Berlin?*).

Table 2 BLEU scores for different prompts and engines.

Engine	**Shots**	**Prompt**	**With Prefix**			**Without Prefix**		
			BLEU-1	***BLEU-2***	***Sent-BLEU***	***BLEU-1***	***BLEU-2***	***Sent-BLEU***
davinci-002	0	Q1	0.255	0.089	0.024	0.284	0.086	0.010
		Q2	0.238	0.080	0.020	0.256	0.075	0.007
		Q3	0.245	0.084	0.023	0.269	0.080	0.008
		Q4	0.254	0.087	0.024	0.276	0.082	0.007
		Q5	0.259	0.088	0.024	0.283	0.084	0.007
		Q6	0.259	0.087	0.022	0.288	0.085	0.007
		Q7	0.254	0.085	0.021	0.279	0.082	0.007
		Q8	0.254	0.084	0.020	0.283	0.083	0.007
		Q9	0.257	0.086	0.021	0.286	0.084	0.007
		Q10	0.260	0.087	0.022	0.288	0.085	0.007
	5	Q1	0.340	0.179	0.067	0.442	0.227	0.078
		Q2	0.340	0.178	0.064	0.441	0.224	0.074
		Q3	0.341	0.180	0.066	0.445	0.229	0.078
		Q4	0.340	0.178	0.064	0.443	0.226	0.075
		Q5	0.342	0.178	0.064	0.441	0.224	0.073
		Q6	0.341	0.179	0.065	0.444	0.226	0.075
		Q7	0.341	0.179	0.065	0.445	0.227	0.076
		Q8	0.341	0.178	0.065	0.445	0.227	0.076
		Q9	0.340	0.177	0.065	0.444	0.226	0.076
		Q10	0.341	0.177	0.064	0.444	0.226	0.075
davinci-003	0	Q1	0.254	0.076	0.007	0.305	0.089	0.007
		Q2	0.257	0.076	0.007	0.304	0.086	0.005
		Q3	0.252	0.074	0.005	0.303	0.088	0.005
		Q4	0.253	0.075	0.006	0.303	0.087	0.006
		Q5	0.258	0.076	0.006	0.306	0.087	0.006
		Q6	0.256	0.075	0.005	0.306	0.088	0.006
		Q7	0.257	0.074	0.005	0.307	0.087	0.006
		Q8	0.256	0.074	0.005	0.308	0.088	0.006
		Q9	0.257	0.075	0.005	0.310	0.088	0.006
		Q10	0.259	0.074	0.005	0.313	0.089	0.006
	5	Q1	0.361	0.198	0.094	0.481	0.262	0.121
		Q2	0.351	0.183	0.069	0.467	0.242	0.090
		Q3	0.349	0.181	0.067	0.465	0.240	0.088
		Q4	0.348	0.178	0.062	0.462	0.234	0.080
		Q5	0.347	0.176	0.059	0.460	0.231	0.076
		Q6	0.348	0.177	0.060	0.462	0.234	0.078
		Q7	0.347	0.176	0.058	0.461	0.232	0.075
		Q8	0.348	0.177	0.059	0.462	0.233	0.077
		Q9	0.348	0.176	0.059	0.461	0.231	0.075
		Q10	0.348	0.177	0.060	0.462	0.232	0.077
davinci-ft	-	-	**0.473**	**0.313**	**0.245**	**0.519**	**0.345**	**0.261**

Table 3 reports the evaluation of the results of the queries generated by each engine, approach, and prompt. Here, recall and precision are both low, thus leading to low F1-scores. Of course, the high number of invalid queries, considered empty, has a negative impact on the results. Towards an alternative comparison with the answers directly generated (Section 5.4), which might include unexpected results, *BLEU* metrics, this time between natural language answers, were also computed, but do not bring much more to the table.

Performance is again better for the few-shot approach than for the zero-shot. Yet, surprisingly, the few-shot compares well to the fine-tuned model. In fact, even if by an insignificant margin, the best F1-score is achieved by the few-shot approach, in *davinci-003*, using prompt Q2.

Table 3 Scores of answers retrieved by generated queries or generated directly by the model.

Engine	Shots	Prompt	Precision	Recall	F1-Score	BLEU-Score	
						BLEU-1	***BLEU-2***
davinci-002	0	Q1	0.028	0.043	0.034	0.022	0.000
		Q2	0.008	0.010	0.009	0.008	0.000
		Q3	0.033	0.035	0.034	0.026	0.000
		Q4	0.027	0.024	0.025	0.020	0.000
		Q5	0.011	0.018	0.014	0.011	0.000
		Q6	0.038	0.039	0.038	0.031	0.007
		Q7	0.023	0.028	0.025	0.016	0.000
		Q8	0.064	0.072	0.068	0.057	0.007
		Q9	0.055	0.062	0.058	0.049	0.000
		Q10	0.063	0.068	0.065	0.050	0.000
	5	Q1	0.113	**0.142**	0.126	0.110	0.000
		Q2	0.116	0.130	0.122	0.113	0.001
		Q3	0.100	0.125	0.111	0.100	0.000
		Q4	0.105	0.123	0.113	0.102	0.001
		Q5	0.094	0.119	0.105	0.093	0.001
		Q6	0.118	0.141	0.128	0.112	0.000
		Q7	0.113	0.134	0.123	0.108	0.001
		Q8	0.104	0.133	0.117	0.104	0.000
		Q9	0.098	0.124	0.109	0.095	0.001
		Q10	0.096	0.109	0.102	0.093	0.001
davinci-003	0	Q1	0.026	0.028	0.027	0.019	0.000
		Q2	0.032	0.040	0.035	0.025	0.000
		Q3	0.026	0.029	0.028	0.019	0.000
		Q4	0.027	0.029	0.028	0.021	0.000
		Q5	0.043	0.053	0.048	0.036	0.000
		Q6	0.027	0.024	0.026	0.015	0.000
		Q7	0.054	0.053	0.054	0.043	0.000
		Q8	0.026	0.024	0.025	0.015	0.000
		Q9	0.077	0.075	0.076	0.061	0.000
		Q10	0.039	0.043	0.041	0.033	0.007
	5	Q1	0.104	0.121	0.112	0.096	0.007
		Q2	0.124	0.139	**0.131**	0.112	0.007
		Q3	0.115	0.131	0.122	0.107	0.007
		Q4	0.120	0.134	0.121	0.103	0.007
		Q5	0.103	0.119	0.110	0.098	0.007
		Q6	0.112	0.119	0.115	0.104	0.007
		Q7	0.114	0.130	0.122	0.104	0.007
		Q8	0.104	0.121	0.112	0.096	0.007
		Q9	0.092	0.107	0.099	0.085	0.007
		Q10	0.114	0.130	0.122	0.104	0.007
davinci-ft	-	-	**0.126**	0.132	0.129	**0.115**	0.008
davinci-002-dir	-	-	**0.317**	**0.419**	**0.361**	**0.240**	**0.124**

Out of curiosity, considering only valid and non-empty queries, the best F1-score is 0.40, specifically with the zero-shot approach in *davinci-002*, using prompt Q8. This is, however, not comparable among approaches, because such queries and their number vary.

5.4 Evaluation of Direct Answers

In addition to *SPARQL* generation, *GPT-3* was used for answering the *QALD*-9 test questions directly, in *NL*. The generated answers were then compared with the gold answers, and performance is included the last line of Table 3. Though not especially high, all the scores are greater than for any other approach. This suggests that, if the query is not important, it is preferable to avoid the extra step of query generation.

6 Discussion

Objectively, *GPT-3* performed poorly for both *SPARQL* generation and *QA*. Yet, if we look at the official results in the *QALD*-9 challenge [22], the 0.131 F1 would rank the best few-shot approach third, which also shows that this is a challenging task. On top of that, asking the questions directly to *GPT-3* would rank it first (0.361 vs 0.298 F1).

However, we note that, since a portion of the entries were discarded from the dataset (see Section 3), these scores are not directly compared to the official. Moreover, official scores date from 2018 and, since then, there has been much progress in text to text generation. In fact, the very recent work [18] that uses *GPT-3* reports on a F1 of 46%. Besides using the full dataset, they consider its 13 languages, not just English, and we do not about some details of the experiment, e.g., whether they applied any kind of pre and post-processing, or how they handled answers that have changed.

In any case, our results suggest that it is preferable to use *GPT-3* directly. And asking a question in *NL* is indeed straightforward, while queries must comply with a formal language, to be made to a *KB* as *DBpedia*. If they are invalid, they will simply not be accepted. Moreover, when it comes to comparing queries, it is usual that the generated query will be different than the one in the dataset, even if slightly (e.g., name of a variable) because there are many different ways to query *DBpedia* and obtain the same results. On the other hand, queries are fixable and human-readable, and they are made to a transparent source of knowledge, represented in *RDF*, in opposition to the black-box reasoning of *GPT-3*. So, when interpretability is a requirement, using *GPT-3* directly is not a solution.

Despite slight improvements in the few-shot scenario with *davinci-003*, differences between *davinci-002* and *davinci-003* engines are minimal. However, we note that *davinci-002* insists on generating *Wikidata* queries, instead of *DBpedia*, which ends up producing erroneous queries. This was also why *DBpedia* was specified in half of the prompts but, apparently, it made no noticeable difference on the quality of the generated *SPARQL*.

Fine-tuning the *davinci* engine led to improvements in the generated *SPARQL*. This was somewhat expected because it was trained in 340 question-query pairs, whereas the few-shot approach only saw five and the zero-shot none. Performance could be possibly improved if more training examples were used, but this would have to resort to a different dataset.

Differences in *SPARQL* generation are, however, not reflected when comparing the results of the generated queries, where the performance of the few-shot approach is comparable to the fine-tuned model. This may be due to different queries that lead to similar results. However, we recall that the best-scored answers were obtained by querying *GPT-3* directly, without *SPARQL* and *DBpedia*.

7 Conclusion and Future Work

GPT-3 has been used for many tasks, and *SPARQL* generation has been attempted with different approaches. Yet, until recently, *GPT-3* had not been explored for the automatic generation of *SPARQL* queries.

Ideally, this would combine the best of text generation with *KB-QA*. Current text generation models are known for their capacity of adapting to many tasks, taking advantage of zero and few-shot learning. However, their inference is not transparent for humans, which hinders their application to critical domains. On the other hand, both *SPARQL* queries and *LD* can be easily scrutinised.

We tested different *GPT-3* engines in zero and few-shot learning with ten different prompts. We also fine-tuned a model for *SPARQL* generation. Results were analysed from the perspective of valid queries produced and their resemblance with correct ones. The evaluation was complemented by scoring the results of running the generated queries and comparing them to those obtained when the original question is asked directly to the model, which also generates an answer in *NL*.

Briefly, in the zero-shot scenario, *GPT-3* generates many invalid queries. Performance increases with the five-shot approach, and even more with fine-tuning, but *BLEU* scores show that generated queries are still far from the gold ones. On the other hand, the results of queries by the few-shot approach are comparable to those of queries by the fine-tuned model. Nevertheless, answers generated directly by the model are the best, even if still far from the gold answers.

Overall, the results were poor and show that we were far from the aforementioned ideal combination. Yet, we learned about the performance of *GPT-3* for this specific task and reported on insights that will hopefully open the door to further exploration of zero and few-shot learning for *SPARQL* generation, using recent *LLM*s. This work was developed as a course mini-project at the University of Coimbra, and some experiments were left to do due to lack of time and resources. For instance, the reported performance could possibly be improved with simple changes, such as: considering the type of question when selecting the training examples for the few-shot approach; as others have done [17, 19], pre-processing SPARQL queries for making them closer to *NL* (e.g., replace $?x$ variables or brackets, respectively by tokens var_x or bra_left); and, most of all, using the correct DBPedia version. The fine-tuned model could be further improved if more training data is used, but this would have to resort to larger datasets (e.g., *LC-QuAD* [20]). Moreover, there are many *LLM*s left to explore, e.g., *OPT-175B* [27], or the recent *GPT-4* [14].

References

1 Ankush Agarwal, Raj Gite, Shreya Laddha, Pushpak Bhattacharyya, Satyanarayan Kar, Asif Ekbal, Prabhjit Thind, Rajesh Zele, and Ravi Shankar. Knowledge Graph–Deep Learning: A Case Study in Question Answering in Aviation Safety Domain. *arXiv preprint arXiv:2205.15952*, 2022. `doi:10.48550/arXiv.2205.15952`.

2 Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, LNCS, pages 722–735. Springer, 2007. `doi:10.1007/978-3-540-76298-0_52`.

3 Mahdi Bakhshi, Mohammadali Nematbakhsh, Mehran Mohsenzadeh, and Amir Masoud Rahmani. SParseQA: Sequential word reordering and parsing for answering complex natural language questions over knowledge graphs. *Knowledge-Based Systems*, 235:107626, 2022. `doi:10.1016/j.knosys.2021.107626`.

4 Petr Baudiš. YodaQA: a modular question answering system pipeline. In *POSTER 2015-19th International Student Conference on Electrical Engineering*, pages 1156–1165, 2015.

5 Romain Beaumont, Brigitte Grau, and Anne-Laure Ligozat. SemGraphQA@ QALD5: LIMSI participation at QALD5@ CLEF. In *Working Notes of CLEF 2015 – Conference and Labs of the Evaluation Forum*, 2015.

6 Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from Question-Answer pairs. In *Proceedings of 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013.

7 Tom et al. Brown. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

8 Yongrui Chen, Huiying Li, and Zejian Xu. Convolutional Neural Network-Based Question Answering Over Knowledge Base with Type Constraint. In *China Conference on Knowledge Graph and Semantic Computing*, pages 28–39. Springer, 2018. `doi:10.1007/978-981-13-3146-6_3`.

9 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, Minneapolis, Minnesota, 2019. ACL.

10 Dennis Diefenbach, Andreas Both, Kamal Singh, and Pierre Maret. Towards a question answering system over the Semantic Web. *Semantic Web*, 11(3):421–439, 2020. `doi:10.3233/SW-190343`.

11 Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. Answering Natural Language Questions by Subgraph Matching over Knowledge Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):824–837, 2018. Conference Name: IEEE Transactions on Knowledge and Data Engineering. `doi:10.1109/TKDE.2017.2766634`.

12 Hai Jin, Yi Luo, Chenjing Gao, Xunzhu Tang, and Pingpeng Yuan. ComQA: Question Answering Over Knowledge Base via Semantic Matching. *IEEE Access*, 7:75235–75246, 2019. Conference Name: IEEE Access. `doi:10.1109/ACCESS.2019.2918675`.

13 Endri Kacupaj, Hamid Zafar, Jens Lehmann, and Maria Maleshkova. VQuAnDa: Verbalization QUestion ANswering DAtaset. In Andreas Harth, Sabrina Kirrane, Axel-Cyrille Ngonga Ngomo, Heiko Paulheim, Anisa Rula, Anna Lisa Gentile, Peter Haase, and Michael Cochez, editors, *The Semantic Web*, pages 531–547, Cham, 2020. Springer International Publishing.

14 OpenAI. GPT-4 Technical Report, March 2023. arXiv:2303.08774 [cs]. `doi:10.48550/arXiv.2303.08774`.

15 Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, page 311, Philadelphia, Pennsylvania, 2002. ACL. `doi:10.3115/1073083.1073135`.

16 Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online, 2020. ACL. `doi:10.18653/v1/2020.emnlp-main.437`.

17 Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, Andre Valdestilhas, Diego Esteves, and Ciro Baron Neto. Sparql as a foreign language. In *Proceedings of the Posters and Demos Track of the 13th International Conference on Semantic Systems - SEMANTiCS2017*, 2017.

18 Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. Evaluation of chatgpt as a question answering system for answering complex questions. *arXiv preprint arXiv:2303.07992*, 2023.

19 Hieu Tran, Long Phan, James Anibal, Binh T Nguyen, and Truong-Son Nguyen. SPBERT: an Efficient Pre-training BERT on SPARQL Queries for Question Answering over Knowledge Graphs. In *Neural Information Processing: 28th International Conference, ICONIP 2021, Proceedings*, pages 512–523. Springer, 2021.

20 Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs. In Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin, editors, *The Semantic Web – ISWC 2017*, LNCS, pages 210–218, Cham, 2017. Springer International Publishing. `doi:10.1007/978-3-319-68204-4_22`.

21 Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. Question Answering over Linked Data (QALD-5). In *Working Notes of CLEF 2015 – Conference and Labs of the Evaluation Forum*, volume 1391 of *CEUR Workshop Proceedings*, page 10. CEUR-WS.org, 2015.

22 Ricardo Usbeck, Ria Hari Gusmita, Axel-Cyrille Ngonga Ngomo, and Muhammad Saleem. 9th Challenge on Question Answering over Linked Data (QALD-9). *Language*, 7(1):58–64, 2018.

23 Mark D. Wilkinson et al. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1):160018, 2016. Number: 1 Publisher: Nature Publishing Group. `doi:10.1038/sdata.2016.18`.

24 Kun Xu, Sheng Zhang, Yansong Feng, and Dongyan Zhao. Answering Natural Language Questions via Phrasal Semantic Parsing. In Chengqing Zong, Jian-Yun Nie, Dongyan Zhao, and Yansong Feng, editors, *Natural Language Processing and Chinese Computing*, Communications in Computer and Information Science, pages 333–344. Springer, 2014. `doi:10.1007/978-3-662-45924-9_30`.

25 Zhengyuan Yang, Zhe Gan, Jianfeng Wang, Xiaowei Hu, Yumao Lu, Zicheng Liu, and Lijuan Wang. An Empirical Study of GPT-3 for Few-Shot Knowledge-Based VQA. *Proceedings of AAAI Conference on Artificial Intelligence*, 36(3):3081–3089, 2022. `doi:10.1609/aaai.v36i3.20215`.

26 Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. Neural machine translating from natural language to SPARQL. *Future Generation Computer Systems*, 117:510–519, 2021. `doi:10.1016/j.future.2020.12.013`.

27 Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open Pre-trained Transformer Language Models, 2022. arXiv:2205.01068 [cs]. `doi:10.48550/arXiv.2205.01068`.

28 Weiguo Zheng and Mei Zhang. Question Answering over Knowledge Graphs via Structural Query Patterns, 2019. arXiv:1910.09760 [cs]. `doi:10.48550/arXiv.1910.09760`.

A Framework for Fostering Easier Access to Enriched Textual Information

Gabriel Silva
IEETA, DETI, University of Aveiro, Portugal
LASI – Intelligent System Associate Laboratory, Coimbra, Portugal

Mário Rodrigues
IEETA, ESTGA, University of Aveiro, Portugal
LASI – Intelligent System Associate Laboratory, Coimbra, Portugal

António Teixeira
IEETA, DETI, University of Aveiro, Portugal
LASI – Intelligent System Associate Laboratory, Coimbra, Portugal

Marlene Amorim
GOVCOPP, DEGEIT, University of Aveiro, Portugal

Abstract

Considering the amount of information in unstructured data it is necessary to have suitable methods to extract information from it. Most of these methods have their own output making it difficult and costly to merge and share this information as there currently is no unified way of representing this information. While most of these methods rely on JSON or XML there has been a push to serialize these into RDF compliant formats due to their flexiblity and the existing ecosystem surrounding them.

In this paper we introduce a framework whose goal is to provide a serialization of enriched data into an RDF format, following FAIR principles, making it more interpretable, interoperable and shareable. We process a subset of the WikiNER dataset and showcase two examples of using this framework: One using CoNLL annotations and the other by performing entity-linking on an already existing graph. The results are a graph with every connection starting from the document and finishing on tokens while keeping the original text intact while embedding the enriched data into it, in this case the CoNLL annotations and Entities.

2012 ACM Subject Classification Information systems → Document representation; Information systems → Ontologies

Keywords and phrases Knowledge graphs, Enriched data, Natural language processing, Triplestore

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.2

Supplementary Material
Software (Dev Repository): `https://github.com/gabrielrsilva11/GraphBuilderAPI`

Funding This research is funded by National Funds through the FCT - Foundation for Science and Technology (UI/BD/153571/2022). It is also funded, Research Unit IEETA, by National Funds through the FCT - Foundation for Science and Technology, in the context of the project UIDB/00127/2020.

1 Introduction

With the the expansion of the internet and IoT, the world saw a dramatic increase in the amount of data that is generated every day [12]. While some of this data comes structured, what we observe, especially in the last decade, is a huge amount of unstructured data that was previously mostly ignored due to the difficulty in processing it. This difficulty in processing comes not only in the form of lack of processing power but also in the fact that the information in this type of data is enconded in natural language. This data is scattered

© Gabriel Silva, Mário Rodrigues, António Teixeira, and Marlene Amorim;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 2; pp. 2:1–2:14
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

throughout the web and comes from many sources. Tweets, forums, comments, anything that is written in natural language can be a valuable source of data to be converted into information. For example, twitter, can be used to identify adverse drug reactions [8] or analyse comments to have a better understanding of patient feedback [13].

There are many methods to enrich this data that have been developed over the years, Named Entity Recognition (NER), keyword extraction, topic modeling, among many others to fit the needs of those working with this data. Another of such methods is text annotations [23]. Text annotations are labels that can be added to specific parts of a text to provide additional information about the content. These annotations can include information such as part-of-speech tags, named entities, or even more complex structures such as a syntactic trees. This enriched data comes in different formats, for example, NER identifies which words are recognized as named entities while something like syntactic annotations need something more structured (like a table or a tree) for their representation. As a result of these different processing methods of large amounts of data that is enriched by different frameworks or humans is a complex process which can consume a lot of resources to accommodate.

Traditionally these annotations have been made and shared using, mostly, XML or JSON. Both the Text Encoding Initiative [1] and the ISO TC37 SC4 WG1 [2][27] have XML defined guidelines for publishing/sharing data with text annotations. However, there is a growing trend to integrate this enriched data with knowledge graphs in favour of JSON or XML formats due to the flexibility and the existing tools for the semantic web [7].

In this work we aim to tackle common problems that are found while working with enriched data. In general, these problems are:

- Difficulty in managing the enriched data with possibly many levels/layers and diversity of formats for each.
- Difficulty in sharing the results of the data enrichment.
- Lack of a common, unified way of representing the enriched data.
- Processing and exploration of the enriched data requires costly development of custom computational solutions.

Having identified these problems we set a few objectives for this work: (1) Creating a framework that can integrate different annotations within a single knowledge graph; (2) Create a knowledge graph that represents a unified way of representing documents (keeping every connection possible) along with their annotations; (3) Make this knowledge graph shareable and integratable with other software according to the user needs.

To tackle these problems our goal is to serialize enriched data into an RDF/OWL compliant format. Achieving this serialization will make systems that this format more interpretable, interoperable, and integrable with the already existent semantic web ecosystem.

By integrating this enriched data with knowledge graphs (both new and existing) we can create richer relationships that have the potential to help improve algorithms. It will help build more robust and accurate applications as well as sharing the information obtained.

The framework should make it possible to go directly from documents to a knowledge graph. Building a framework capable of doing this would avoid certain issues like having different data enrichment tools generate different representations. There is also the added benefit of being more convenient to whoever uses the framework.

---

[1] `https://tei-c.org/`

[2] `https://www.iso.org/committee/48104.html`

The work will be developed using CoNLL and Entity Linking as our main use-cases to showcase the framework as well as the resulting knowledge graph. The decision to use CoNLL was because, as mentioned before, it is a widely used and accepted standard despite the existing challenges. Entity linking will be to show the extensibility of the framework and how it can be adapted to different annotations.

Paper structure. Following this introductory section, the paper structure is as follows: Section 2 will focus on the framework and the different modules it comprises. How they were developed and what each module does. Section 3 we present the use-cases of the framework: CoNLL annotations and Entity Linking. Section 4 is the related work chapter, and Section 5 will present a conclusion and future work.

2 Our proposal: A framework for accessing enriched textual information

This section will focus on the framework that was developed as well as the decisions behind each module and how the requirements set at the beginning of the work were accomplished. It starts by describing the system and how its modules work, followed by what was used during the implementation as well as its issues. In the end we present a use-case of the framework which will be entity-linking.

2.1 Requirements

The requirements set for this framework relate to both the general problems identified as well as the concrete objectives defined for this work. As such, the requirements are:

- Standards adoption – Adopting standards, such as RDF/OWL or CoNLL, makes the tool be able to be integrated with an already existing suite of software.
- Extensible and Flexible – There are a lot of different text annotations that users might want to contemplate using and as such it should be possible to extend the usability of the tool to cater to these different annotations.
- Keep all the connections intact – That means linking the text to sentences, sentences to words and annotation to words and make it easy to navigate both ways. This will help keep the structure of the text intact and recreate the original document if needed. Ease of navigation in the knowledge graph is a must to develop different applications of the tool.
- Storage/Software Agnostic – The goal here is for the users to be able to integrate the tool with whatever stack they are using making it as easy to use as possible for everyone. Creating and uploading a knowledge graph should not be dependent on, for example, a single triple storage system.
- Abide by FAIR [28] principles – The four FAIR principals are the following: findable, accessible, interoperable and reusable. These are principals we want to abide by because of how important and connected to the problems we found they are, especially the last three.

These were the major requirements that were identified as well as the rationale behind each one of them.

2.2 Proposal/System

The major goal of this framework is to provide professionals with an extensible, easy-to-use library which can be adapted to their needs and relies on standard practices and tools only.

We propose a framework with an initial implementation as a python library that takes a document and transforms it into a direct RDF serialization of enriched data that follows the FAIR principles [28]. The first serialization, and the default one implemented in the library, is done using CoNLL.

By serializing the enriched data into RDF while keeping all the connections between elements intact we create an easy way to navigate this information as well as build upon it.

2.3 Framework overview

This framework uses standard resources that are not unknown to anyone that has ever worked with NLP before. SpaCy and Stanza, which are the two main drivers of this work, are widely used both in industry as well as research. The serialization is done to RDF which is another well-known standard and one that is widely used, not only for NLP tasks. The main difficulty found while developing the library was making it flexible so that users can, for example, write their own queries.

The Fig. 1 presents a simple overview of the framework and its different components for our use-cases.

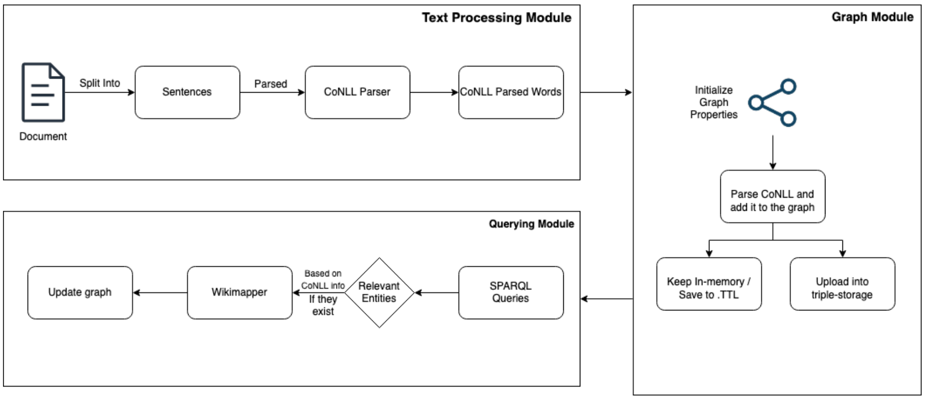


Figure 1 Overview of the current framework architecture and the different modules.

The framework consists of 3 main modules: (1) Text Processing Module; (2) Graph Module; (3) Querying Module. These are the models without which the framework would not function. Each of them handles a different part of the pipeline needed to go from enriched data to the final RDF serialization.

The text processing module objective is to take the raw document and apply the information enrichment methods into it. For example, in our use case, and as seen in Figure 1 we start off by processing the document, running it through a CoNLL parser (our enrichment method) and we then build the knowledge graph based on this.

The Graph module, as the title implies, is responsible for the serialization of the enriched data into an RDF compliant format. This involves parsing the enriched data, creating the necessary connections to maintain the connections in the document intact as well as either generating an RDF file or uploading it into a triple-storage system.

The last module is the querying module. This module is responsible for querying the data and displaying its information or updating an existing graph to add information to it. In our use-cases this is when entity linking is performed to showcase the flexibility of the framework.

The implementation of each module will be presented in the next subsections.

Table 1 Explanation of what each CoNLL data property represents.

Attribute	Description
id	The index of the word in the sentence.
word	The text of the word
lemma	The lemma of the word
pos	Part-of-speech of the word
feats	Morphological features of the word
edge	Universal dependency relation of the word
head	ID of the syntactic head of this word

2.4 First implementation of the Framework

In this section we will discuss how each module was implemented and the functionalities present in each one.

2.4.1 Text processing module

The text processing module is responsible for reading the documents and parsing them into the CoNLL format.

The first step is to split the text into sentences so that we can keep the connection from text into the different sentences. Followed by using SpaCy [18] and Stanza [25] (formerly StanfordNLP) to create the CoNLL representation.

This module allows us to choose the language in which the document comes in and the output is a pandas dataframe in a standard tabular CoNLL format. Table 1 shows each CoNLL attribute and their meaning. From here on out this is the representation we will use to create our graph.

2.4.2 Graph Module

In order to build the graph we first start off by defining the required classes, data properties and object properties. There are three classes defined: Text, sentence and word. This is so that we can preserve every connection possible to be able to navigate the graph whichever way we want and make it so that it is possible recreate the original document, even if just an unformatted version of it.

The data properties are the ones that come from the parsed CoNLL data. These include: edge, feats, lemma, id, pos, poscoarse, word and another data property called “sentence_text” that connects the first word of the sentence with the sentence itself. The head property from CoNLL will be defined as an object property instead of data since it isn’t related with the data itself but with the structure of the text and the syntactic tree.

When it comes to object properties most of these were created because of either the structure of the text or with navigation in mind. We create the following object properties: head, depGraph, nextSentence, nextWord, containsSentence, previousWord, fromSentence. Each of these properties are used in different one or various classes/instances. For text the property containsSentence is used to know which sentences belong to a document/text. For sentences we have nextSentence, this is used to know which sentence came next in the text, and depGraph that represents the start of the syntactic tree of that sentence (the root node of the CoNLL representation). The rest of them are related to words, nextWord and previousWord are related to the order in which the orders appear in the sentence and depGraph are the words that depend on the current word. Figure 2 has a simplification of these connections and showcases how we can navigate in the graph. X is the number of the sentence and Y is the number of the word, so Sentence_1 would be the first sentence and Word_1_3 would be the third word of the first sentence. After adding these properties and classes all that is left is to create the graph. We go through the CoNLL tabular data, parse it, and start by adding the sentences and appending the words to these sentences with the respective tags.

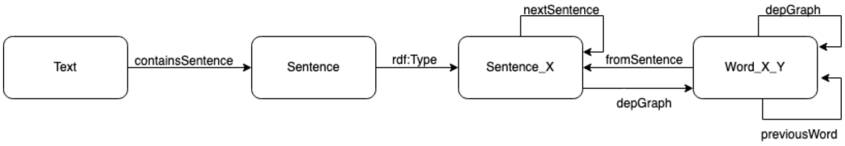


Figure 2 Simplification of the connections existent in the knowledge graph.

It is also possible to choose how you want your graph. There are two methods available, build it in-memory and export a TTL (Terse RDF Triple Language), a format used to express RDF data, file or upload it into a triple storage system as long as there is an available end-point. Ensuring the need for an endpoint instead of supporting specific triple-storage systems makes the system more agnostic and able to be integrated with different systems to suit different users needs.

The entity linking portion of the work is just a demonstration of a use-case of this framework. We start off by identifying relevant word tags, such as, obj, obl or nsubj and their dependents. From there on we utilize wikimapper which is a project that helps mapping page titles with the corresponding wikipedia articles. If there are any relevant entities and corresponding wikipedia articles we update the sentence to link these to the wikipedia pages and therefor get a more complete knowledge base.

2.4.3 Querying Module

The final module of this system is the Querying Module. This is a module that comes with some pre-built queries and the option to build your own. Currently this module also includes the entity linking module.

The current built-in queries are able to: (1) insert triples into a storage; (2) Find sentences that start by a given string; (3) Find sentences that contain a specific string or list of strings; (4) Find sentence by id;

These queries are built with SPARQL [3], a communication and querying protocol, and work for both triple-storage methods and the in-memory one. There are also functions that use these queries to accomplish other tasks, for example, building sub-graphs with all the dependencies starting from the root, find the syntactic path of a given subset of words to the root node, finding the node of a given word.

This module operates mostly in the same way for every problem a user might have. It starts off with a query to find out the useful nodes, for example, querying by sub-strings or by an attribute. We build a sub-graph of the queried information and then apply our navigation methods to find what the user wants to know.

---

[3] `https://www.w3.org/TR/sparql11-query/`

3 Use-Cases

To exemplify using this library we make use of two use-cases. The first use-case, which comes built-in with the framework, is the creation of a knowledge graph with CoNLL annotations. This example will show how we keep all the relations in a document and how we add our own annotations to it. The second use-case is entity linking. Using the previously created knowledge graph we perform entity linking and append the extracted entities to the corresponding sentences.

We used the WikiNER [9] data for the Portuguese language and split it into a small subset of data. We extracted 5121 sentences and processed them, built a knowledge graph with CoNLL data (first use-case) and performed entity linking (second use-case).

3.1 CoNLL

Looking at one of the most popular annotation methods and one of our use-cases, CoNLL [2], we can already identify some challenges when working with it. While this is a format that is widely adopted by several systems and used by researchers everywhere it is not the most human-readable format which makes interpreting the results hard, using tools to parse the result table may lead to transformation errors or loss of information and sharing results is not the easiest task.

To build the knowledge graph, we started off by processing the information as shown in Figure 1. The input is a text file with each line being a sentence. The library starts off by reading the file and converting the sentences into CoNLL using a combination of SpaCy [18] and stanza [25]. This is done in chunks to avoid using more memory than what the system would be capable of handling. As the chunks are parsed, they are being added to a new knowledge graph. In this case we are not doing this in-memory but using a triple-storage system called Virtuoso. We parse the text into CoNLL format and then, the resulting tabular data is converted into triples to be uploaded to our system by an endpoint.

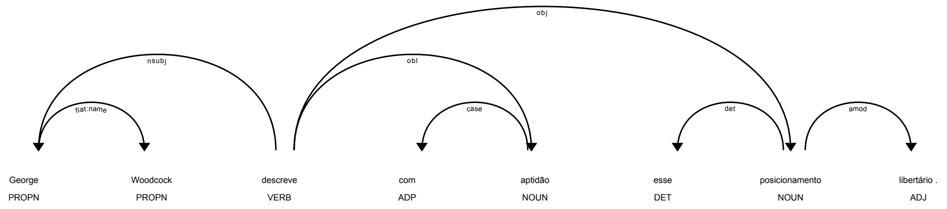
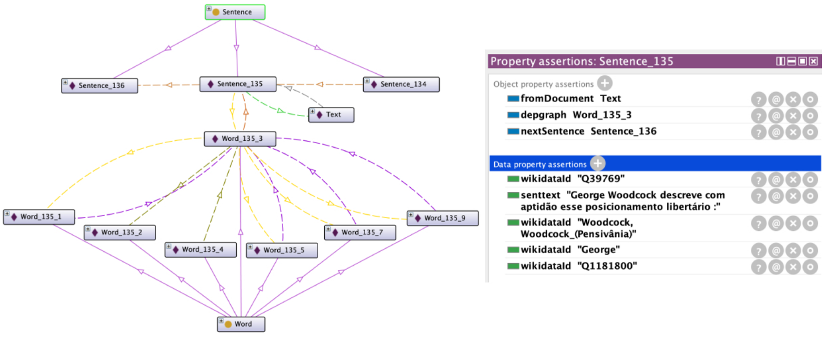


Figure 3 Example of the universal dependencies parsing for the sentence “George Woodcock descreve com aptidão esse posicionamento libertário .”.

To highlight the flexibility of the system we wanted to first create the CoNLL graph based on this subset of data and then add the entity linking part of the work. The processing and upload of the data took 4 hours to complete and the final result was a graph, with every connection intact (document -> sentence -> word -> CoNLL attributes). Figure 3 shows an example of the type of annotations, in this case the universal dependencies, present in our data properties that come from CoNLL.

The result, shown in Protégé using OntoGraf, and for the sentence “George Woodcock descreve com aptidão esse posicionamento libertário .” is shown in Figure 4. The framework is made available as a Python library and to process a document the user has to mention:

the desired relationships uri, an endpoint (if uploading to a triple storage), the language of the document and the document or folder of documents. Then all that is left is to instantiate our *CreateGraph* class and call the *create_graph* method. The output will either be an RDF file or the graph uploaded into a triple storage. A more in-depth explanation of the process is in Section 2.4.



Property assertions: Sentence_135

Object property assertions
- fromDocument Text
- depgraph Word_135_3
- nextSentence Sentence_136

Data property assertions
- wikidataId "Q39769"
- senttext "George Woodcock descreve com aptidão esse posicionamento libertário :"
- wikidataId "Woodcock, Woodcock_(Pensivânia)"
- wikidataId "George"
- wikidataId "Q1181800"

Figure 4 Example of a sentence and its connections seen in Protégé [20] with OntoGraf as well as its properties.

3.2 Entity Linking

The way we perform entity linking here was by using the universal dependency relations found by CoNLL. We look at the 3 nominal core arguments of a sentence, namely, nsubj, obj and iobj. Nsubj is identified as being the syntactic subject of a clause. Obj is the object of a verb. Iobj is an indirect object that is a core argument of the verb but not a subject (nsubj) or a direct object (obj) [10]. Not only do we look at these but also at the relations that are related with these three relations.

For example, in the phrase “George Woodcock descreve com aptidão esse posicionamento libertário”. “descreve” is our verb, “George” our nsubj and “posicionamento” our obj. Instead of looking only at “George” and “posicionamento” to perform the entity linking we also look at ”George Woodcock” and “esse posicionamento libertário”. We do this because, as seen in Figure 3, “Woodcock”, “esse” and “libertário” are associated with our nsubj and obj. This will lead to more robust entities being found in Wikidata as well as a more complete knowledge base.

This relation information is already stored in our knowledge graph. To access it we use a query SPARQL to fetch a sentence and then navigate on it. We start off with query to fetch a single sentence and build its sub-graph. The following query will fetch all the data related to a single sentence and from this data we build a sub-graph that we can navigate. In this query str_id represents the id of a sentence.

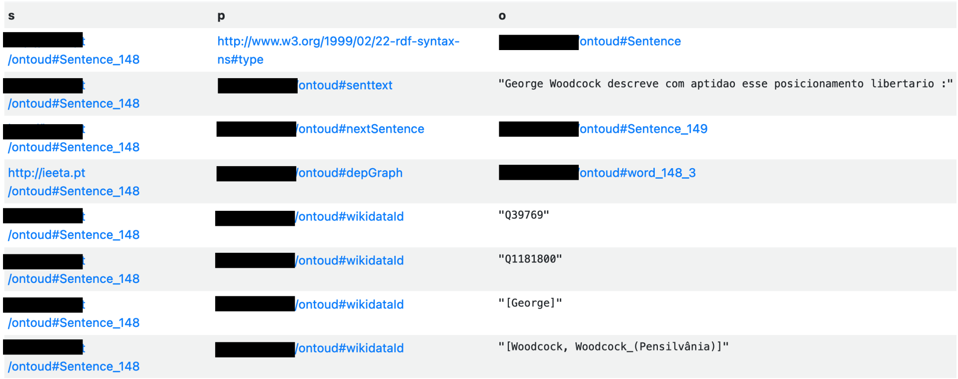
```
PREFIX dbp:  <http://dbpedia.org/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX defpref: <http://example.pt/framework#>

SELECT ?s ?p ?o
WHERE {
    <""" + str_id + """> defpref:depgraph* ?s .
    ?s ?p ?o .
}
```

From there on it is a matter of navigating between the links and properties of the graph that are shown in Figure 2. We define a root node (the word that is one of the relations previously mentioned) and navigate using the “depGraph” link to fetch all the dependencies of said node and build our partial sentence.

Using Wikimapper [4] we find if these entities exist on the portuguese version of wikipedia or not and fetch their IDs and related names. The final step is to append this ID and names to the sentence using the connection “wikidataId”. Figure 5 shows the Virtuoso SPARQL viewer for the following query:

```
PREFIX dbp:  <http://dbpedia.org/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX defpref: <http://example.pt/ontoud#>

SELECT ?s ?p ?o
WHERE {
    defpref:Sentence_148 defpref:depgraph* ?s .
    ?s ?p ?o .
}
```

This query will return the triples related with “Sentence_148” which corresponds to the sentence in Figure 3.



s	p	o
[redacted] /ontoud#Sentence_148	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	[redacted]ontoud#Sentence
[redacted] /ontoud#Sentence_148	[redacted]ontoud#senttext	"George Woodcock descreve com aptidao esse posicionamento libertario :"
[redacted] /ontoud#Sentence_148	[redacted]/ontoud#nextSentence	[redacted]ontoud#Sentence_149
http://ieeta.pt /ontoud#Sentence_148	[redacted]/ontoud#depGraph	[redacted]ontoud#word_148_3
[redacted] /ontoud#Sentence_148	[redacted]/ontoud#wikidataId	"Q39769"
[redacted] /ontoud#Sentence_148	[redacted]/ontoud#wikidataId	"Q1181800"
[redacted] /ontoud#Sentence_148	[redacted]/ontoud#wikidataId	"[George]"
[redacted] /ontoud#Sentence_148	[redacted]/ontoud#wikidataId	"[Woodcock, Woodcock_(Pensilvânia)]"

Figure 5 Example of some sentences and the corresponding wikidata IDs found taken from Virtuoso using a SPARQL query.

4 Related Work

In this section we will present the related work. The proposed framework can be used in several areas of work, however, the most directly related ones are annotations (text, semantic), linked data, and since it was our use case, entity linking.

A brief summary of the developments and current state-of-the-art will be presented here in next subsection for each of these topics.

---

[4] `https://github.com/jcklie/wikimapper`

4.1 Annotations

Works that follow the Linguistic Annotation Framework which was developed within the ISO TC37 SC4 WG1 are a great starting point for anyone looking into developing a new framework for annotations. This workgroup has put out several documents that can serve as an entry-point, covering most of the language processing field [26]. They have developed standards for morpho-syntactic representation (ISO 24611) [5] and syntactic structures (ISO 24615) [6] and representation of annotated content [27] (ISO 24612) [7] are some of the works published by this work group that are relevant to this work. In general these frameworks provide general requirements that should be followed when creating new annotations framework. For example, the representation of annotated content presents requirements such as expressive adequacy, media independence, semantic adequacy, incrementability, separability, uniformity, openness, extensibility, human readability, processability and consistency [27].

The POWLA (Portable Linguistic Annotation with OWL) [4] is one such framework that follows these guidelines and requirements. It is a framework designed to represent linguistic data structures in a LOD/OWL-compliant way. It is a direct RDF implementation of the PAULA datatypes and annotations. The authors also argue for the benefits of representing annotations in an RDF format over the at-the-time state-of-the-art systems (XML and GrAF).

There are other types of annotations that are not related to semantics but are also relevant to the work developed.

Sentiment annotation consists of looking at a sentence sentiment and assigning it a value of positive, negative or neutral. Assigning these marks is usually done to sentences as a whole but specific words can also convey a negative or positive meaning. As the author of [17] mentions it is not always clear to human annotators the emotion that is being portrayed so there is a need to create more complex annotation schemes that contemplate different parameters especially when considering text such as transcriptions or speeches.

Document Classification is another type of annotation. This one works at a document level instead of sentence or words and the goal is to classify a document into a set of existing categories [14].

4.2 Linked Data

Linked Data and NLP are two highly interconnected fields [7]. The focus of linked data is to create a web of interconnected data which can be queried and analysed in a structured way which pairs well, and even enriches, the goal of NLP which is to better understand and interpret human language [15]. In recent years, the trend has been to use more RDF/OWL based formats due to their interoperability and flexible yet standardized way of representing data [1]. We have seen researchers share more data in this format. For example, CovidPubGraph [24], Protein Ontology [3].

However, NLP is a big field of study and there are many tools which have different and varied outputs making it so that working on large project with many components becomes a complex task due to having to manage all these tools and different standards. There have been attempts to create a standard way to share NLP data by serializing it into RDF, such as the NIF (NLP Interchange Format) [11]. NIF is an RDF/OWL-based format whose goal

---

[5] `https://www.iso.org/standard/51934.html`
[6] `https://www.iso.org/standard/62508.html`
[7] `https://www.iso.org/standard/37326.html`

was to achieve interoperability and reusability between NLP tools, language resources and annotations. This format specifies several rules to follow as well as defining formal URIs that can be used to maintain consistency between tools and help researchers get over the hurdle of different formats for each tool and even achieve better results due to having the possibility to enrich their data with LOD (Linked Open Data).

There have also been other projects that attempt to convert formats, for example, csv or tsv into ontologies. The W3C keeps a web page of what they call “ConverterToRdf”[8] as well as “RDFImportersAndAdapters”[9] which lists different formats and the tools that can be used for each format, however, this list does not seem to be updated as some of these projects are now offline. This list also does not make any mention of text annotations which is what this work is going for.

The only project that we found who attempted to convert text annotations such as CoNLL into an RDF format was CoNLL-RDF [5, 6]. This is a Java framework where the user provides their own CoNLL file, and the program outputs the RDF representation relying on NIF. This project, however, has some key differences to the framework proposed. The first one being accepting free text / documents instead a CoNLL file. The proposed framework deals with text and serializes it into an ontology with all the CoNLL information in it. This makes it so that the attributes will always be consistent across different corpora. Providing your own file into CoNLL-RDF may break serialization if you use an unknown tagger or there is any error in the generated file. Our work also provides some pre-built SPARQL queries for some useful text processing as well as the option to integrate with different triple-storage systems (Jena, Virtuoso, etc ...) if there is an available endpoint. With CoNLL-RDF the only output is a RDF or TTL file which you can then upload it into said triple-storage systems, however, if your corpora is too big this isn’t an ideal solution.

4.3 Entity Linking

While entity linking is not the main scope of this work it is one of the use cases identified for this work. By using known and established knowledge bases we can improve the quality of our data and of our own knowledge bases. While here we link entities to their corresponding Wikipedia pages, EEL (Entity Extraction and Linking) is a vast field in which a wide variety of approaches are taken.

Representative systme in this area are J-NERD which performs Entity Extraction and Linking with respect to YAGO2/Wordnet and Wikipedia [22], Babelfy which combines Wikipedia WordNet and Babelnet into a semantic network to be used as a multilingual reference knowledge base [19], AIDA-Light focuses on scalability by using a two-step process and uses YAGO2 and Wikipedia [21]. Besides these systems there have also been investments in API/Web Services, for example, IBM has AlchemyAPI[10] which offers a suit of different NLP services (including EEL or Yahoo! Content Analysis API[11] which also includes entity/concept detection. For a more comprehensive study of such systems and a much more in-depth survey of EEL it is recommended to look at [16] and for neural based approaches the survey [29].

---

[8] `https://www.w3.org/wiki/ConverterToRdf`

[9] `https://www.w3.org/wiki/RDFImportersAndAdapters`

[10] `https://www.ibm.com/watson/alchemy-api.html`

[11] `https://developer.yahoo.com/contentanalysis/`

5 Conclusion

This paper presents a framework whose goal is to make working with NLP data easier. It offers a common, shareable, unified way of representing data without the need to write immense amounts of custom code. The framework is already capable of creating an RDF-compliant way of working with NLP data, knowledge base completion, text statistics (by navigating the graph) and appending annotations post graph creation. In this version of the framework the requirements that were set at the start were also met. We abide by the FAIR principles [28] by doing the following: it is findable and accessible by being online and available through pypi or github and the dataset being public. It is interoperable by virtue of using a standard format that any triple storage can use. It is reusable due to the data annotation being CoNLL which is a widely known and used format as well not making use of any proprietary frameworks or standards. All the connections from document to the word are intact and navigable both ways. The graphs can be built in-memory or uploaded to a triple storage that accepts SPARQL. We showed that it is extensible and flexible by adding our own annotations after creating the CoNLL graph. We adopt standards and frameworks, such as RDF and CoNLL, that allow integration with already existing software.

While the framework implementation currently only supports CoNLL annotations by default for enriched data, it is already capable of other functions, including (1) Uploading data to a triple-storage as long as there is an available endpoint; (2) Pre-built SPARQL queries to work with the data; (3) Basic entity linking to wikipedia pages.

These functionalities already showcase how powerful the framework is, we also see no issues as to why it could not support other standards, such as POWLA [4] or other types of non-syntactic annotations as long as they are related to either a document, a sentence or words. It is important to make the library adaptable and extensible since enriched data methods are always evolving and the needs of professionals are different depending on their use-cases.

As an example of a use-case we processed a sub-set of the WikiNER dataset for the Portuguese language, we built the knowledge graph with CoNLL annotations and performed a basic form of entity-linking on it. This was a subset with 5121 sentences and it took about 4 hours to process the whole dataset and upload it to Virtuoso[12]. To create the graph in-memory and export as a TTL file it took about 20 minutes. After the graph was created and to show how we can easily extend it with other functionalities we performed entity linking to get a more complete graph. In order to find entities we looked at three main universal dependency relations: obj, obl and nsubj as well as their dependencies to look for entities. When we have these relations and dependencies we use Wikimapper to find, if it exists, the corresponding Wikidata ID and titles, appending them to the sentence it belongs.

These use-cases show the flexibility of the framework and the type of work it can do. We opted to work with a triple-storage system, however it could also be done in-memory but for large documents this is not advised due to memory constraints.

5.1 Future Work

There is still plenty of work that can be done with the current state of the library. The first thing we plan on doing is optimizing the library. We can introduce parallelization when processing large amounts of text. While most of the overhead when working with a triple storage vs in-memory comes from network calls (4 hours vs 20 minutes processing time) for large texts this will be beneficial.

---

[12] `https://virtuoso.openlinksw.com/`

When it comes to enriched data, adding built-in support to other annotations methods and allow users to choose which ones they want to keep in their serialization is another step that is planned.

Allowing users to export only whichever parts of the graph that they want is also planned if, for example, only part of the graph needs to be shared, or the graph without any custom annotations that might have been done.

References

1 Michael Bergman. Advantages and Myths of RDF. *AI3, April*, 2009.

2 Sabine Buchholz and Erwin Marsi. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June 2006. Association for Computational Linguistics. URL: `https://aclanthology.org/W06-2920`.

3 Chuming Chen, Hongzhan Huang, Karen E. Ross, Julie E. Cowart, Cecilia N. Arighi, Cathy H. Wu, and Darren A. Natale. Protein ontology on the semantic web for knowledge discovery. *Scientific Data*, 7(1):337, October 2020. `doi:10.1038/s41597-020-00679-9`.

4 Christian Chiarcos. POWLA: Modeling Linguistic Corpora in OWL/DL. In Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti, editors, *The Semantic Web: Research and Applications*, pages 225–239, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

5 Christian Chiarcos and Christian Fäth. CoNLL-RDF: Linked Corpora Done in an NLP-Friendly Way. In *International Conference on Language, Data, and Knowledge*, 2017.

6 Christian Chiarcos and Luis Glaser. A Tree Extension for CoNLL-RDF. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 7161–7169, Marseille, France, May 2020. European Language Resources Association. URL: `https://aclanthology.org/2020.lrec-1.885`.

7 Philipp Cimiano, Christian Chiarcos, John P. McCrae, and Jorge Gracia. *Modelling Linguistic Annotations*, pages 89–122. Springer International Publishing, Cham, 2020. `doi:10.1007/978-3-030-30225-2_6`.

8 Anne Cocos, Alexander G Fiks, and Aaron J Masino. Deep learning for pharmacovigilance: recurrent neural network architectures for labeling adverse drug reactions in Twitter posts. *Journal of the American Medical Informatics Association*, 24(4):813–821, 2017.

9 Silviu Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 708–716, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL: `https://aclanthology.org/D07-1074`.

10 Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL: `http://www.lrec-conf.org/proceedings/lrec2014/pdf/1062_Paper.pdf`.

11 Sebastian Hellmann, Jens Lehmann, Sören Auer, and Martin Brümmer. Integrating NLP Using Linked Data. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, pages 98–113, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

12 Martin Hilbert. Big Data for Development: A Review of Promises and Challenges. *Development Policy Review*, 34:135–174, January 2016. `doi:10.1111/dpr.12142`.

13 Mustafa Khanbhai, Patrick Anyadi, Joshua Symons, Kelsey Flott, Ara Darzi, and Erik Mayer. Applying natural language processing and machine learning techniques to patient experience feedback: a systematic review. *BMJ Health Care Inform.*, 28(1):e100262, March 2021.

14 Vandana Korde. Text Classification and Classifiers:A Survey. *International Journal of Artificial Intelligence & Applications*, 3:85–99, March 2012. `doi:10.5121/ijaia.2012.3208`.

15 Elizabeth D Liddy. Natural language processing, 2001.

16 Jose L Martinez-Rodriguez, Aidan Hogan, and Ivan Lopez-Arevalo. Information extraction meets the semantic web: a survey. *Semantic Web*, 11(2):255–335, 2020.

17 Saif Mohammad. A practical guide to sentiment annotation: Challenges and solutions. In *Proceedings of the 7th workshop on computational approaches to subjectivity, sentiment and social media analysis*, pages 174–179, 2016.

18 Ines Montani, Matthew Honnibal, Matthew Honnibal, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020. `doi:10.5281/ZENODO.1212303`.

19 Andrea Moro, Alessandro Raganato, and Roberto Navigli. Entity Linking meets Word Sense Disambiguation: a Unified Approach. *Transactions of the Association for Computational Linguistics*, 2:231–244, 2014. `doi:10.1162/tacl_a_00179`.

20 Mark A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015. `doi:10.1145/2757001.2757003`.

21 Dat Ba Nguyen, Johannes Hoffart, Martin Theobald, and Gerhard Weikum. AIDA-light: High-Throughput Named-Entity Disambiguation. *LDOW*, 1184, 2014.

22 Dat Ba Nguyen, Martin Theobald, and Gerhard Weikum. J-NERD: Joint Named Entity Recognition and Disambiguation with Rich Linguistic Features. *Transactions of the Association for Computational Linguistics*, 4:215–229, 2016. `doi:10.1162/tacl_a_00094`.

23 Eyal Oren, Knud Möller, Simon Scerri, Siegfried Handschuh, and Michael Sintek. What are semantic annotations. *Relatório técnico. DERI Galway*, 9:62, 2006.

24 Svetlana Pestryakova, Daniel Vollmers, Mohamed Ahmed Sherif, Stefan Heindorf, Muhammad Saleem, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo. CovidPubGraph: A FAIR Knowledge Graph of COVID-19 Publications. *Scientific Data*, 9(1):389, July 2022. `doi:10.1038/s41597-022-01298-2`.

25 Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages, 2020. `arXiv:2003.07082`.

26 Laurent Romary. Standards for language resources in ISO – Looking back at 13 fruitful years, 2015. `arXiv:1510.07851`.

27 Laurent Romary and Nancy Ide. International Standard for a Linguistic Annotation Framework, 2007. `arXiv:0707.3269`.

28 Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1):160018, March 2016. `doi:10.1038/sdata.2016.18`.

29 Özge Sevgili, Artem Shelmanov, Mikhail Arkhipov, Alexander Panchenko, and Chris Biemann. Neural entity linking: A survey of models based on deep learning. *Semantic Web*, 13(3):527–570, April 2022. `doi:10.3233/sw-222986`.

A Pseudonymization Prototype for Hungarian

Attila Novák ✉
Faculty of Information Technology and Bionics, Pázmány Péter Catholic University, Budapest, Hungary

Borbála Novák ✉
Faculty of Information Technology and Bionics, Pázmány Péter Catholic University, Budapest, Hungary

Abstract

In this paper, we present a pseudonymization prototype for Hungarian, an agglutinating language with complex morphology, implemented as a web service. The service provides the following functions: entity identification and extraction; automatic generation and selection of replacement candidates; automatic and consistent replacement and reinflection of entities in the final pseudonymized document. The named entity recognition model applied handles names of persons well, and it has decent performance on other entity types as well. However ID-like entities need to be handled separately to achieve proper performance (not handled in the current prototype version). For automatic replacement candidate generation, a simple entity embedding model is used. We discuss the performance and limitations of the prototype in detail.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases named entity recognition, morphological reinflection, pseudonymization, entity embedding model

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.3

Funding The research presented in this paper was implemented with support provided by grant FK 125217 of the National Research, Development and Innovation Office of Hungary financed under the FK 17 funding scheme.

1 Introduction

Machine learning-based NLP models are often domain dependent to some extent in the sense that their performance often depends on how similar the features (topic, style, vocabulary) of the text to which the model is applied are to the features of the texts used to train the model. The performance of models can often be significantly improved if (ideally a significant amount of) in-domain training data is available.

However, access to types of texts containing sensitive personal data (such as medical or crime-related information) is severely restricted, and this can be a serious obstacle to the development of high-quality models for handling texts in such domains. Consistent automatic replacement of personal data in texts with similar but fictitious data is a possible solution to this problem. The type of solution of which we outline a working research prototype in this paper could provide a general solution to the legal problems that hinder the publication and use of texts containing sensitive data, and thus contribute significantly to the development of high-quality language models in these domains.

The objective of the research presented here was to develop a prototype that uses a high-precision and high-coverage named entity recognition algorithm to identify names and other personal data together with their entity type in text, and then associates fictitious but natural-looking names and data with them, replacing the occurrences of names and data (including suffixed forms) in the text in a consistent way. This way, we obtain texts that no longer contain real personal data and are therefore no longer constrained by the restrictions pertaining to the original data. These can thus be made available for training or fine-tuning

© Attila Novák and Borbála Novák;
licensed under Creative Commons License CC-BY 4.0

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 3; pp. 3:1–3:10
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

language models that can handle texts in the given domain. This type of models can also be used to replace the identified entities and other data with unique identifiers to provide proper data masking. Typically, pseudonymization means that a given code key can be used to recover the original information, and, indeed, the systematic recoverability of information is often a desired objective. When unique identifiers are used to replace the original names and data, these are recognizably different from real data. However, to achieve our original objective (i.e. to obtain a restriction-free version of a corpus of texts in a domain typically restricted by the presence of sensitive information), we need a solution that results in an output that is essentially indistinguishable from the original in terms of lexical distribution, thus we need to replace names with false names of similar distribution rather than with identifiers. This solution is usually referred to as data masking. For this purpose, the code key that could be used to recover the original data is unnecessary, and it can and should be discarded or at least is to be kept separately and subject to technical and organizational measures to ensure non-attribution to any identified or identifiable person.

Reversal of pseudonymization may be possible in some edge cases using externally available information even in the absence of the code key, if unmodified information in the context can be used to infer the identity of the person in an unambiguous manner. For this not to be the case, data masking in general needs to be applied not only to names of persons but also to other entities like institutions, places, facilities etc. in order to mask the contexts as well.

What makes the proper data masking process non-trivial in the case of morphologically complex languages like Hungarian is that nouns may have dozens-to-hundreds of possible inflected and derived forms that all should be recognized, disambiguated and consistently replaced and reinflected in context.

2 Method

The prototype system we present in this paper contains the following components (for detailed discussion of components and functions see the subsections below):

1. A named entity recognition (NER) model to identify the entities to be replaced and the type of each entity.
2. A model for disambiguated morphological analysis and lemmatization to identify the lexical form of entities to be replaced and the actual morphological form the name to be used for substitution must take in the given context.
3. A simple extraction module that compiles a collection of extracted entities (in a normalized/lemmatized form) along with their types and frequencies.
4. A model that can suggest suitable replacement candidates for the identified entities. Ideally, replacements should be done in a manner that is consistent throughout the document.
5. A model that replaces the identified entity names and reinflects them to match the grammatical context. For this, a morphological generator model is used.
6. The prototype is implemented as a dockerized web service taking json input and returning json output.

The web service call API has the following functions:

1. Function *spans* to preprocess text and extract entities (these are added to the input json).
2. Function *suggest* to suggest replacements (entities are enriched with suggestions and replacements) creating a replacements configuration. If the input does not contain preprocessed data and extracted entities, function *spans* is performed before suggesting replacements.

3. Function *replace* to automatically replace entities in text based on the replacements configuration reinflecting them in context. *spans* and/or *suggest* are also performed, if needed. The replacements configuration can be modified manually or in an automatic manner by an external process before calling *replace*.

2.1 The named entity recognition (NER) model

The NER model we used is based on the Hungarian named entity corpus *NerKor+Cars-OntoNotes++* [3], a 1.04M-token corpus covering 85 thousand entities of a relatively fine-grained 28-class entity type set. The entity classes include the 18 entity types covered by the *Ontonotes 5* corpus [8], and further types (such as *media, social media, awards, motor vehicles, projects*) differentiated when creating the corpus. We used this corpus to finetune a transformer encoder model to perform token classification. It is based on the Hugging Face Transformers tool set, and it is available at the Hugging Face Hub[1].

The key entity types targeted in the current prototype application are: *persons* (PER), *organizations* (ORG), *geopolitical entities* (GPE): i.e. names of settlements, countries and geopolitical regions like counties, and *facilities* (FAC), which include streets/roads and other public spaces. The latter two types are parts of addresses, an important target data type for the data masking task. The *NerKor+Cars-OntoNotes++* annotation does not cover nested entities, thus addresses are not annotated as a whole, only their parts are identified by the named entity annotation model. Nevertheless, given the unnested annotation, since the consistent replacement of names (including settlement names) in the document is desirable, annotation of addresses as a sequence of settlement and street address is not problematic. The *NerKor+Cars-OntoNotes++* corpus (in contrast to earlier NER datasets) also identifies derived forms of names.

The model also identifies *dates, times, time durations, numerical values, quantities, amounts of money* and *certain types of ID's* in addition to named entities. These entities are not only annotated but are also extracted. While the replacement of these types of entities is necessary in a full-fledged data masking annotation solution, the current prototype, while it identifies and annotates most of these, it does not automatically suggest a replacement for them (while it does generate automatic replacement candidates for named entities).

2.2 A morphological analysis and lemmatization

The models used for disambiguated morphological annotation and lemmatization are the *emMorph* morphological analyzer [4, 5] integrated with the *PurePos/emTag* tagger [6]. We used the e-magyar/emtsv pipeline [7] to integrate the morphological analyzer, the tagger and the named entity recognizer. However, we used improved versions of all tools instead of the ones originally shipped with e-magyar.

The most important improvement concerns the NER model: the model in e-magyar distinguishes only four entity types: persons, organizations, locations and miscellaneous (all other types of named entities), and it was trained on a much smaller 226k-token corpus in a limited domain (business news). Thus the original NER model does not properly identify or differentiate some entity types important in a data masking application (e.g. geopolitical entities, facilities and other geographical locations are not differentiated, and no derived

---

[1] `https://huggingface.co/novakat/nerkor-cars-onpp-hubert`

forms of named entities are recognized).[2] In addition, it has suboptimal performance on generic (non-business-domain) texts due to its training data being limited to a single domain: the BERT-based model originally featured in e-magyar has an F_1 score of only 0.8439 on the union of test sets of the business news NER corpus and NerKor corpus, while for the model trained on NerKor (using the same architecture), F_1 was found to be 0.9197 on the same joint test set.

We have also retrained the tagger model on an improved version of its original training corpus where annotation errors due to earlier erroneous conversion of morphosyntactic annotation were fixed. We also extended the stem database of the morphological analyzer to improve coverage of named entities both in analysis and generation.

2.3 Marking and extraction of entities

All entities identified by the NER model are marked in the text to be processed, they are normalized to a lexical form based on the lemmatization provided by the morphological analyzer, and the normalized form of the entities is extracted as a json dictionary including frequency and entity type data. Normalization affects the rightmost element of multi-word entities, as case marking and other inflections are attached to the head of the noun phrase, which is on the right in Hungarian. Normalization is needed to create a single representative lexical form for all inflected forms of each entity to make the consistent replacement and reinflection of the entities possible. The entity annotation including morphological analysis on heads of noun phrases is added to the original text as markup. This representation is used later to replace and reinflect entities.

2.4 The model for replacement suggestion

The dictionary of extracted entities serves as a basis for the configuration of the replacements to be performed. Replacement candidates are automatically added to this data structure when the *suggest* function of the web service call API is called.

In the current prototype, we used a simple static word/entity embedding model to automatically populate the replacements configuration candidates section. The embedding model was trained using an annotated 2-billion-word web-crawled corpus. The annotation followed the format presented in [2]: inflected words are represented by a sequence of two tokens: one consisting of the lemma and the PoS tag, and another independent token representing the morphological endings. The following example shows the representation of the sentence *Szeretlek, kedvesem.* ‘I love you, my darling.’:

```
szeret[/V] [Prs.1Sg>2] ,[Punct] kedves[/N] [Poss.1Sg]
love       [I, you]    ,        darling    [my]
```

This representation, while no information is lost, improves the quality of the word embedding model compared to one created from surface word forms in several ways: by assigning a separate representation to lexical items of different part of speech, by effectively reducing data sparseness problems following from the great variety of rare inflected word forms, and thus by improving the representation of lemmata.

---

[2] In Hungarian, derivational suffixes are used to derive adjectives from e.g. names of locations. These are not capitalized: *budapesti* ‘of/in/to Budapest’

This scheme was extended to include a representation of entities. In addition to morphological annotation, the corpus was also annotated using the NER model presented in Subsection 2.1. Lemmata of heads of entities were also annotated by the entity type in addition to PoS. Sentences containing multi-word entities were represented twice in the training data: once with the whole entity represented by a single token (with underscores between words of the phrase) and once by each non-terminal surface word form of each multi-word entity appearing as an independent token. This made it possible to generate independent representations for the whole entity and its constituent parts.

```
Iványi_Márta[/N]=[PER] [Nom] -[Punct] szoprán[/Adj] [Nom]
Iványi Márta[/N]=[PER] [Nom] -[Punct] szoprán[/Adj] [Nom]
Márta Iványi                 -        soprano
```

This feature of the embedding model makes it possible for the data masking tool to handle different entity types in a different manner. E.g. replacement candidates for names of persons are generated by handling surnames and given names independently. This ensures that e.g. kinship relations among persons mentioned in the text reflected by identical surnames are preserved by handling the surnames consistently. In contrast, facilities (e.g. street names) or the names of organizations are treated by the algorithm as one unit, thus it does not make up completely fictitious street or organization names.

The embedding model was trained using the fastText CBOW algorithm [1]. For the entity types handled by the automatic replacement model of the prototype, a random shuffled sample (currently 5 items) of the top (currently 50) nearest neighbors according to cosine similarity from the embedding model are added as replacement candidates to the replacements configuration with the second candidate selected (the first candidate is always the original entity itself). For quantities, date and ID entities, currently no additional replacement candidate is generated (see limitations of the current model below in Section 3). The replacement candidates generated for the female name *Bulcsu Mariann* are shown as an example below (with the candidate *Putz Evelin* selected).

```
"Bulcsu Mariann[/N]": {
  "allsugg": [["Bulcsu","Putz","Gutbrod","Maczucza","Südy","Gálos"],
     ["Mariann","Evelin","Zita","Judit","Krisztina","Erika"]],
  "frq": 1,
  "repl": "Putz Evelin",
  "norm": "Bulcsu Mariann",
  "type": {"PER": 1}
},
```

2.5 Replacement and re-inflection of entities

Entities in the annotated original text are replaced based on the replacements configuration passed to the *replace* API function. The normalized form of entities as marked in the annotation is first replaced by the normalized form of the selected replacement candidate, and then the latter is re-inflected using a morphological generator model created from the morphological analyzer. The morphological generator uses the lemma of the head and the disambiguated morphosyntactic analysis in the original annotation to generate the contextually appropriate inflected form of the replaced entity.

3 Limitations

The current prototype implementation has a number of limitations.

It does not automatically replace numeric, identifier or date-type entities, nor does it recognize all of these types (especially some types of identifiers such as vehicle registration numbers or telephone numbers). The latter is a major limitation from a GDPR point of view, which can be handled by either creating extra training data for the underlying NER model and retraining it, or by applying an independent identifier recognition model. Automatic replacement of numerical, date-type etc. entities identified by the current NER model can be simply solved by implementing and calling a module that generates and injects a replacement for entities of these types in the entity replacements configuration before invoking the replacement function. Most of these identifiers are easily detectable using simple regex-based patterns. We did not deal with this issue in the current prototype.

The system does not currently detect and consistently handle informal references to entities or accidental misspellings or other name variations. E.g. nicknames referring to persons in the text are not identified, and the suggested replacement for them is not consistent with the the replacement of the full name (e.g. the suggested replacement for a name like *Johnny* may be e.g. *Frankie* while the replacement candidate for *John* may be e.g. *Michael*).

The automatic replacement of names referring to geographic entities are not currently handled consistently either. E.g. while the original text may mention settlements which are close to each other geographically and also the name of the county[3] where all of them are located may be mentioned, the automatically selected replacements may not have the same properties.

Addresses are not handled completely consistently, either. The zip code may be inconsistent with the rest of the address, and the substituted settlement may not have the independently replaced street name. These inconsistencies in the output may negatively impact the quality of a complex language model trained on the output of the data masking tool, thus a better model for the replacement of place names and addresses would be desirable in a follow-up model version. It is a question to what extent it is desirable to mask settlement names. It is possible that masking street addresses provides satisfactory data masking to prevent possible data leaks.

In addition, some personal data may “leak” through the system due to imperfect recall of the named entity recognizer. However, the subsequent identification of such entities in the text is only possible in certain presumably rare cases: if almost all occurrences of a name are replaced except one, the inconsistent contextually unanchored single occurrence of the name may be inferred to be the original name.

Note that for some linguistic tasks requiring domain adaptation, consistent replacement of names is not necessarily required: a solution where names in the document are replaced randomly may be sufficient. While the resulting text is not suitable for training e.g. of models for co-reference resolution, it is certainly not a problem in this case if some data has not been replaced by chance, because there is no way of deducing from the text which elements of the original text remained unchanged.

A further problem may be the (lack of) handling of nested entities (when one name element contains another name element). The frequency of such errors can be estimated by measuring them on test data, and their actual impact can be assessed by manual inspection of automatically pseudonymized example texts.

---

[3] A county is a large administrative area in Hungary: Hungary consists of 19 counties.

4 Evaluation

We performed evaluation of the prototype on a 14.5k-word sample of police witness interview reports, which had been manually pseudonymized before. Manual pseudonymization was performed at the data processing company of the Ministry of the Interior, thus we were provided test data that had already been manually made GDPR-compliant. Nevertheless, the data was not made publicly available. The training data for the NER model did not contain any data similar to this genre.

Table 1 Performance of the NER model on the test corpus. Scores were reported by the CONLL-2003 NER evaluation script.

Frq. Ratio/Acc	Type	P	R	F_1
99.48%		94.74	93.33	94.03
37.59%	PER	99.01	99.01	99.01
14.26%	CARDINAL	97.44	98.70	98.06
12.96%	GPE	96.88	88.57	92.54
6.85%	FAC	94.59	94.59	94.59
6.67%	DATE	87.80	100.00	93.51
4.07%	ORG	86.67	59.09	70.27
3.89%	ORDINAL	100.00	100.00	100.00
3.70%	TIME	95.24	100.00	97.56
2.41%	DUR	100.00	100.00	100.00
1.67%	ID	100.00	55.56	71.43
1.30%	QUANTITY	100.00	100.00	100.00
1.11%	MONEY	100.00	100.00	100.00
1.11%	TEL	0.00	0.00	0.00
1.11%	CAR	100.00	33.33	50.00
0.93%	LOC	35.71	100.00	52.63
0.37%	AGE	100.00	100.00	100.00
0.00%	PROD	0.00	0.00	0.00

Table 1 shows the entity recognition performance on this test corpus. The overall F_1 score was 0.943, which is quite acceptable (only exact token span match is rewarded). The entity types in the table are ordered by their frequency in the gold test data. The most common entities are *person* names, for which we get an almost perfect recognition performance. This is quite reassuring, as this is the most important entity type from a GDPR point of view. As for now, *cardinals* also subsume zip codes and house numbers, as these have not been distinguished in the original NER model. But they are easily identifiable given their distribution relative to other parts of addresses. The relatively lower recall for geopolitical entities is caused by a) some informal references to settlement names (where instead of the official name of a settlement, a colloquial form was used) that were mostly mistagged by the model as LOC (this is a minor issue, its impact on replacement is that the suggested replacement for the name was of the wrong semantic category), and b) by the model sometimes missing some derived forms of settlement names. The relatively low recall for ORG entities is due to references in the texts to specific police headquarters and their departments in all caps, which was sometimes left untagged or tagged as LOC (except for the settlement name within the name, which is identified as GPE). This is not a grave problem either with regard to the performance of the data masking application. The model cannot identify phone numbers and some types of ID's, as mentioned in Section 3 on limitations. Other errors included the model tagging some car occurrences in the corpus as products rather than assigning the more specific CAR tag.

We also evaluated replacement and reinflection performance. The test set contained 894 identified entities, of which 474 were replaced. The main error types are shown in Table 2. The second column shows the ratio of errors in the replacement configurations, the third column in the actual occurrences in the test corpus.

The majority of unreplaced entity occurrences was due to our decision not to change them in the current prototype (dates, times, quantities, ID's, etc.). 44 entities (8.5% of the 518 entities that should have been replaced) remained unreplaced due to some error in entity identification or automatic suggestion generation. This ratio definitely needs to be improved in a production version (e.g. by replacing/improving the static-embedding-based suggestion algorithm). The inconsistent replacement of nicknames mentioned in Section 3 affects 2.7% of entities to be replaced. The replacement was deemed 'improper' in 5% of the cases: the selected replacement is of a different name type than the original, or a foreign first name was suggested as a replacement of a Hungarian name.[4] Adding a simple filtering mechanism to the handling of persons names could alleviate these problems. In many cases, misclassification of entity type by the NER model could be identified as the cause of improper replacement. Another 0.6% was replaced inconsistently due to some other factor (e.g. nonstandard usage of street address).

Table 2 Error types and actual occurrences in entity replacement due to an error in the automatic entity detection or replacement candidate generation/selection.

error type	entries in the configurations	affected occurrences
all	40 (19.7)	87 (16.8)
unreplaced	23 (11.3)	44 (8.5)
improper	9 (4.4)	26 (5.0)
nickname	5 (2.5)	14 (2.7)
inconsistent	3 (1.5)	3 (0.6)

Reinflection errors in the final document can be traced back either to morphological analysis errors (often due to some spelling error, affecting 1.8% of replaced entities) or affect locative cases of settlement names (0.8%). In Hungarian, names of geopolitical entities and public places (like street names) take a locative form either in the superessive *(on)* (e.g. *Budapest, Magyarország* 'Hungary'), or in the inessive *(in)* case (e.g. *Madrid, Spanyolország* 'Spain'). This is a lexical property of the name, and replacement of a name with another that belongs to the other group results in improper case inflection. This problem will need to be addressed in a future model update.

5 In the context of ChatGPT

The prototype we presented in this paper is based on a traditional NLP pipeline (although some elements of the pipeline have a neural implementation). Although it was created before ChatGPT's earthquake-like debut, we felt compelled to check whether we can get ChatGPT perform the same task out of the box. However, we did not manage to prompt ChatGPT

---

[4] This is due to the fact that a specific subset of foreign first names (of celebrities) are popular among members of some specific social groups, while other similar foreign first names are not.

into performing a consistent and comprehensive pseudonymization of Hungarian texts, like the system presented here performs.[5] It either left most entities intact (despite an explicit request not to retain any original names or data), or it just used single letters to mask more (still not all) entities in the text.

6 Conclusion

We presented a pseudonymization/data masking prototype for Hungarian providing functions of entity identification and extraction, automatic generation and selection of replacement candidates, and automatic and consistent replacement and reinflection of entities in the final pseudonymized document. The named entity recognition model handles most relevant entity types well, however ID-like entities need to be handled separately to achieve proper performance (not handled in the current prototype). The simple entity embedding model used for replacement candidate generation has some limitations, however, we managed to handle the problem of replacing names consistently to a reasonable degree. Performance of the prototype is acceptable, although further improvement is needed to develop it into a fully-fledged, reliable data masking solution that also outputs completely consistent text with names having a completely natural distribution.

References

1 Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. `doi:10.1162/tacl_a_00051`.

2 Attila Novák and Borbála Novák. Cross-lingual generation and evaluation of a wide-coverage lexical semantic resource. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL: `https://aclanthology.org/L18-1007`.

3 Attila Novák and Borbála Novák. NerKor+Cars-OntoNotes++. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference (LREC 2022)*, pages 1907–1916, Marseille, France, June 2022. European Language Resources Association. URL: `https://aclanthology.org/2022.lrec-1.203`.

4 Attila Novák. A new form of Humor – Mapping constraint-based computational morphologies to a finite-state representation. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1068–1073, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL: `http://www.lrec-conf.org/proceedings/lrec2014/pdf/207_Paper.pdf`.

5 Attila Novák, Borbála Siklósi, and Csaba Oravecz. A new integrated open-source morphological analyzer for Hungarian. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1315–1322, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL: `https://aclanthology.org/L16-1209`.

---

[5] We did not perform exhaustive prompt engineering. We tried (the Hungarian equivalent of) the following prompt variants: *a. Replace all names and sensitive data in the text below consistently with a similar name.* b. a+*(not letters)*, c. b+*Do not retain any original names or data.*

6 György Orosz and Attila Novák. PurePos 2.0: a hybrid tool for morphological disambiguation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 539–545, Hissar, Bulgaria, September 2013. INCOMA Ltd. Shoumen, BULGARIA. URL: `https://aclanthology.org/R13-1071`.

7 Tamás Váradi, Eszter Simon, Bálint Sass, Iván Mittelholcz, Attila Novák, Balázs Indig, Richárd Farkas, and Veronika Vincze. E-magyar – A Digital Language Processing System. In Nicoletta Calzolari (Conference chair), Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 7-12 2018. European Language Resources Association (ELRA).

8 Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, and Ann Houston. OntoNotes Release 5.0, 2013. `doi:10.35111/xmhb-2b84`.

Generating and Ranking Distractors for Multiple-Choice Questions in Portuguese

Hugo Gonçalo Oliveira
Center of Informatics and Systems, University of Coimbra, Portugal
Department of Informatics Engineering, University of Coimbra, Portugal

Igor Caetano
Instituto Pedro Nunes, Coimbra, Portugal
Department of Informatics Engineering, University of Coimbra, Portugal

Renato Matos
Center of Informatics and Systems, University of Coimbra, Portugal
Department of Informatics Engineering, University of Coimbra, Portugal

Hugo Amaro
Instituto Pedro Nunes, LIS, Coimbra, Portugal

Abstract

In the process of multiple-choice question generation, different methods are often considered for distractor acquisition, as an attempt to cover as many questions as possible. Some, however, result in many candidate distractors of variable quality, while only three or four are necessary. We implement some distractor generation methods for Portuguese and propose their combination and ranking with language models. Experimentation results confirm that this increases both coverage and suitability of the selected distractors.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases Multiple-Choice Questions, Distractor Generation, Language Models

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.4

Supplementary Material *Software (Source Code)*: `https://github.com/NLP-CISUC/smartedu-aqg/blob/main/Generating_Ranking_Distractors_PT.ipynb`

Funding This work was funded by: project SmartEDU (CENTRO-01-0247-FEDER-072620), co-financed by FEDER, through PT2020, and by the Regional Operational Programme Centro 2020; and through the FCT – Foundation for Science and Technology, I.P., within the scope of the project CISUC – UID/CEC/00326/2020 and by the European Social Fund, through the Regional Operational Program Centro 2020.

1 Introduction

Recent breakthroughs in Natural Language Processing (NLP) made knowledge even more accessible with tasks like Question Answering. In most cases, however, this does not mean that training and assessing humans is no longer necessary. Here, another task that benefits from NLP is Question Generation (QG) [12]. As the name suggests, QG aims at creating questions automatically (e.g., from learning materials), thus reducing the time that educators spend in the production of tests and leaving more time for activities like class preparation or interaction with students.

Due to straightforward grading, multiple-choice questions (MCQs) are a popular kind of questions. In addition to the question stem, MCQs have a list of alternative answers, out of which one is correct and the others are distractors. The creation of MCQs has also been automatised [1] in a process that considers the generation of the distractors. Many distractor generation methods have been proposed, but they are rarely suitable to every type

© Hugo Gonçalo Oliveira, Igor Caetano, Renato Matos, and Hugo Amaro;
licensed under Creative Commons License CC-BY 4.0

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 4; pp. 4:1–4:9
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of question, making it necessary to combine different methods. At the same time, some of the methods, or their combination, may produce a large set of candidates, while, in most cases, only three or four distractors are necessary. A selection has to be done, but the produced distractors are often of variable quality, so a random selection is rarely the best option.

We compile a set of distractor generation methods common in the literature, describe their adaptation to Portuguese, and apply them to a set of machine reading comprehension questions. To minimise the impact of random selection, we further propose a straightforward method for ranking distractors. It is based on pretrained language models, namely BERT [8] or GPT2 [23], and their application to computing the likelihood of textual sequences. A manual evaluation of results for a set of questions confirms that such models are a good option for ranking the distractors. They can be applied to distractors by different methods, thus increasing the number of covered questions, as well as the proportion of good distractors.

In the remainder of the paper, after reviewing some related work, we describe the implemented methods for generation and ranking; we report on a performed experiment and its evaluation; we conclude with final remarks and possible future directions.

2 Related Work

When generating MCQs [1], distractors have to be generated in addition to the stem of the questions. The quality of distractors has been estimated with several automatic methods, including named entities (NEs) of the same category, relatedness in WordNet, semantic types in DBPedia, or distributional semantics [21]. Not surprisingly, most of the previous methods were also applied to distractor generation.

When the answer is a word of a specific part-of-speech (PoS) or a named entity (NE) of a specific category [29], context words of the same type can be used as distractors. When the answer is a number, distractors can be obtained by increasing or decreasing it [29].

Alternatively, distractors can be retrieved from external resources, such as WordNet [9] or DBPedia [14]. From the latter, words that share a hypernym with the answer (co-hyponym) [18, 29] or that are similar enough [29] can be used. If too many distractors are obtained this way, preference can be given to those that appear in the context [18]. From DBPedia, distractors can be obtained by removing restrictions in the SPARQL query that answers the question [26]. Concepts that share properties or are related with the answers have also been obtained from other ontologies [28]. The external resource can also be a model of distributional semantics, where words similar to the answer can be obtained from [28, 11]. Other methods include using words with similar spelling [11] or masked language modelling [2]. Transformers like T5 may also be fine-tuned for generating MCQs, including the distractors [16].

In some of the previous, distractors can be ranked, according to one or more of the following features: PoS similarity [25, 2], semantic similarity with the answer [11, 2, 25], proximity of frequency [11, 25], or confidence score of a language model [2]. In any case, distractors cannot be synonyms of the answer.

Specifically for Portuguese, there is some work on QG. The majority relies on linguistic knowledge, such as syntactic dependencies [6, 22] or semantic roles [10], sometimes focusing exclusively on named entities [22]. But there is recent work with neural [15] approaches.

For distractor generation in Portuguese, words that shared traces with the answer have been used [6]. Specifically for cloze-style questions, multiple approaches were applied for distractor generation [3], which could be words with similar features (e.g., PoS, frequency), words obtained by exploring common errors in Portuguese, or related words (e.g., hyponyms and hypernyms in lexical resources). In the scope of listening comprehension, distractors were obtained from phonetically-similar words [20].

3 Approach

Several distractor generation methods were compiled from the literature and adapted to Portuguese. In order to select a subset of distractors by the previous, we rely on language models for computing their likelihood as answers to the question. This section describes the distractor generation and ranking methods.

3.1 Distractor Generation

Five distractor generation methods were implemented in this work. Due to their specificities, they do not produce distractors for every single question-answer pair. Yet, the number of covered questions can be maximised by a combination of methods.

The first method, hereafter Ctx, is the only that selects distractors from a given context and it only applies if such a context is available. If the answer is a NE, other entities of the same category that appear in the context are selected and used directly as distractors. Otherwise, context words of the same PoS of words in the answer are selected to replace the latter and result in new distractors.

Since many answers are or include numbers (e.g., ages, years, quantities), a method (Nb) was implemented for generating distractors specifically for them. They are obtained by replacing each numeric token of the answer by a range of numbers resulting from the addition or subtraction of units.

Having in mind that distractors should be semantically-similar to the correct answer, the remaining generation methods resort to three different resources for getting words of the same category. One (WN) gets co-hyponyms, i.e., words that share a hypernym, from a WordNet-like [9] lexical database.

Since wordnets cover mostly lexicographic knowledge, for world concepts, we get distractors from DBPedia [14] (DBP), an open multilingual knowledge base extracted from Wikipedia. Words that share one or more properties are good distractor candidates. In a parallelism with WN, we focus on words of the same category.

Distractors are also obtained from the most similar words (Sim), according to a word2vec-like [17] model. To avoid the inclusion of alternative correct answers, synonyms and hypernyms of the answer are removed with the help of WordNet.

If no distractors are obtained for the full answer with the previous three methods, they are applied to each open token in the answer, which is then replaced by the retrieved words and used as distractors. Possible outputs of the described methods, when implemented according to section 4, are illustrated in Tables 1 and 2.

Table 1 Examples of context, question, answer, and distractors extracted from context.

Context	Question	Answer	Distractors	Type
O caixão de Bell foi construído com pinho Beinn Bhreagh ... pediu aos convidados para não usarem preto (a cor tradicional do funeral) ..., durante o qual o solista Jean MacDonald cantou um verso de "Requiem" de Robert Louis Stevenson: ...	Qual cantor se apresentou no funeral de Bell?	Jean MacDonald	Beinn Bhreagh, Robert Louis Stevenson	Ctx
			Arsène MacDonald, Romain MacDonal, Gabriel MacDonald, ...	DBP

Table 2 Examples of questions and answers in the dataset, followed by distractors generated by different methods.

Question	Answer	Distractors	Type
Que peça do uniforme foi substituída pelo boné de patrulha?	boina preta	balackava preta, gorro preta, fez preta, quepe preta, ...	WN
		jaqueta preta, gorro preta, camisola preta, boina branca, boina vermelha, boina prateada, ...	Sim
Onde está localizado o templo de Walhalla?	Baviera	Berlim Leste, Renânia do Norte-Vestfália, Baden, Hamburgo, Saxônia, ...	DBP
Quando Victoria pediu a Palmerston que retomasse seu escritório?	Junho de 1859	Junho de 1849, Junho de 1850, ... , Junho de 1868, Junho de 1869	Nb
		Janeiro de 1859, Agosto de 1859, Novembro de 1859	DBP
Qual é a substância mais comumente abusada durante a adolescência nos EUA?	álcool	água potável, anfetamina, café, leite, tabaco, ...	WN

3.2 Distractor Ranking

Given their specificities, the five distractor generation methods will not generate distractors for all types of questions. The problem is that, in many cases, there will still be many distractors, even if only three or four are necessary. At the same time, their quality will be variable. For instance, without further polishing, distractors might include typos (e.g., *ttulos soberanos*, *agencia de jornais*) or, after replacement: result in inconsistent gender / number (e.g., *boina adequado*); result from very generic connections (e.g., *Portas Citosina* or *Portas Teriflunomida* for *Portas USB*, because USB was an American Invention); be related to a different sense of the answer (e.g., *Anatomia de Yongying* for *Corpo de Yongying*); or simply result in odd mixes (e.g., *Oskar New York Times*). This is why, instead of just using a random sample of all the produced distractors, a method for either selecting the most promising, or for discarding problematic ones, can be useful. Here, we could opt for classifying distractors as good or bad. However, this discrimination is often subjective (see Section 4) and, even when distractors are good, they might have different "levels" of suitability. Therefore, we opt for ranking distractors and propose to use language models (LMs) in what they were originally developed for: computing the likelihood of text sequences. A sequence will consist of the question immediately followed by the answer, e.g., the first distractor in table 2 results in the following sequence: *Que peça do uniforme foi substituída pelo boné de patrulha? balackava preta.* For each question, a sequence like the previous is produced for each distractor, and distractors are ranked in descending order of the likelihood of their sequence. Considering that, in any case, selected distractors should be reviewed by a human, it should be easier to manually select distractors from a ranking than from a set, possibly containing dozens of options.

4 Experimentation

To test the distractor generation and ranking methods, they were applied to a selection of Portuguese questions and answers. Obtained distractors were then manually evaluated and some conclusions were taken. This section describes the data used, the implementation of the methods, and finally presents the results and their discussion.

4.1 Evaluation Data

Distractors were generated for a random selection of 124 context-question-answer tuples in the validation portion of a Portuguese translation of the SQuAD [24] dataset, produced by the Deep Learning Brasil group[1]. Since MCQs typically have short answers, the sample was restricted to questions of three-token answers or less. The first three columns in Table 1, context, question, answer, illustrate the entries of the dataset. The original version of SQuAD has been extensively used for training question answering and generation models and it seemed appropriate to our experimentation. In opposition to another popular dataset, RACE [13], it does not contain distractors, but, as far as we know, RACE is not available for Portuguese. In any case, it would be difficult to automatise the evaluation of generated distractors, because there are often many suitable options.

4.2 Implementation

To implement the distractor generation for Portuguese, several tools and resources were used. In the Ctx method, the context is first tagged with the spaCy[2] toolkit, using the largest available model for Portuguese, `pt_core_news_lg`. This enables the identification of NEs and of the words' PoS. Only words of open PoS were considered for replacement. The same model was used for obtaining the most similar words in the Sim method. In the Nb method, numeric tokens nt are identified with Python's `isnumeric()` function. Then, all the numbers in the $[nt-10, nt[$ and $]nt, nt+10]$ intervals are generated to be used as replacements. The WN method relied on the NLTK interface to wordnet[3]. For Portuguese, it resorts to OpenWordNet-PT [7]. For DBP, DBPedia was accessed through its SPARQL endpoint[4]. It first uses the `skos:broader` property, which links concepts with their broader categories, i.e., we get the labels of concepts that share a broader category with the answer. If no distractors are obtained, we do the same for the `dct:subject` property, which links concepts with related subjects, i.e., we retrieve the labels of concepts related to the same subjects as the answer.

For ranking distractors, three LMs were tested, all available from the HuggingFace `transformers` library[5]: BERTimbau [27], both base and large, a BERT model pretrained for Portuguese; and GPorTuguese-2[6], GPT2-small fine-tuned with 1GB of Portuguese text.

For the BERT models, we relied on the FitBERT[7] tool, also based on the `transformers` library. This tool relies on pre-softmax logit scores for ranking a list of options according to their suitability to replace a mask in a given masked sentence. In this case, the input

---

[1] https://drive.google.com/file/d/1Q0IaIlv2h2BC468MwUFmUST0EyN7gNkn
[2] https://spacy.io/
[3] https://www.nltk.org/howto/wordnet.html
[4] https://dbpedia.org/sparql
[5] https://huggingface.co/transformers/
[6] https://huggingface.co/pierreguillou/gpt2-small-portuguese
[7] https://github.com/Qordobacode/fitbert

sentence was the question followed by a mask, and the options were the generated distractors. With GPT2, the likelihood of each sequence of tokens was approximated by the exponential of the loss of the model for this sequence.

4.3 Evaluation

Distractor generation methods were applied to each question of the evaluation data and their results were ranked by each language model. For evaluation purposes, at most three distractors were selected from each generation and each ranking method. When a generation method resulted in more than three distractors, their selection was random. As for ranking methods, they were applied to the set of all distractors by all the methods, before the previous selection, out of which the top-3 were selected.

Distractors resulting from the previous process were then shuffled for manual evaluation, which was done by two judges, one expert in Natural Language Processing and a Data Science student. Given the context, the question, the correct answer, and list of distractors, judges were asked to classify each distractor as: (0) unsuitable, i.e., nonsense or a synonym of the answer; (1) close, but a minor edition is needed, e.g., changing the gender, number or tense of a word; (2) suitable. Both judges were aware of the distractor generation methods but, during the evaluation process, did not have access to the source of each distractor. In order to compute agreement, distractors for the first 25 questions (230) were evaluated by both judges. Considering the three classes, Cohen's *kappa* was 0.61 (substantial agreement), which increased to 0.77 when the unsuitable (0) and close (1) classes were merged.

With the distractors classified, we observed the coverage of each method, as well as on the proportion of suitable distractors generated. The coverage of each method approximates the proportion of distractors of the target type generated for each question, considering a maximum of three per question, and is given by the total number distractors of the type divided by the times the number of questions. Table 3 summarizes these results[8].

Table 3 Distractor Evaluation.

Method	**Coverage**	**0**	**1**	**2**
Ctx	42.2%	33.8%	16.7%	49.7%
Nb	21.8%	3.7%	3.7%	**93.0%**
WN	39.2%	24.7%	11.0%	64.4%
DBP	25.0%	19.4%	7.5%	73.1%
Sim	54.0%	43.4%	6.8%	49.7%
GPT2	**96.0%**	29.7%	13.3%	56.9%
BERT-base	**96.0%**	19.4%	8.4%	**72.2%**
BERT-large	**96.0%**	18.9%	6.7%	**74.4%**

Despite varying across methods, there is a significant proportion of unsuitable distractors with all methods but Nb. This is also the method with the greatest proportion of suitable distractors, followed by DBP, but, even if sometimes by a low margin, all provide at least around 50% suitable distractors. It is easy to generate distractors for numbers. With the current simplistic method, some situations could go wrong (e.g., negative quantities), but they were a minority in the evaluation sample. However, such questions account for only one fifth of the sample, and other methods must be used for the remaining questions.

---

[8] For the shared 25 questions, only the classifications of the first judge were considered.

Looking at the coverage, we confirm that no method applies to a large proportion of questions. Greatest coverages are by Sim and Ctx, but these are also the least accurate methods. With Sim, there is not much control on the obtained words, which are sometimes plurals of the answer, or words of the same family. As for Ctx, we checked that the majority of issues did not result from complete distractors obtained from context, but from replacements of words with the same PoS.

The ranking methods consider the generations of each method, thus significantly increasing the coverage and still having a better proportion of suitable distractors (except for Nb). The 4% of distractors missing with these methods occur in a minority of situations where no generation method could generate a distractor. Among the models used, BERTimbau is preferable to GPorTuguese-2. This is not necessarily due to the model architecture, but may be caused by the data they were pretrained on. BERTimbau was pretrained for Portuguese from scratch, whereas GPorTuguese-2 is GPT2, pretrained for English, then fine-tuned for Portuguese. Performance of the two versions of BERTimbau, base and large, are very similar.

5 Conclusion

We have described the implementation of several methods for generating distractors, to be used in the creation of MCQs in Portuguese. They are complementary but their combination and raking by a language model provides both the best coverage and accuracy. The utility of such a straightforward method was confirmed by an experimentation where distractors were generated for a selection of questions and then manually classified.

This research contributes to the development of SmartEDU, a platform that aims at accelerating the process of producing education materials [5], with a focus on MCQs and slide deck generation [4]. In the future, we will work on improving the current methods and how some deal with incorrect spellings, such as missing accents, missing characters, or unexpected characters-(e.g., *seculo 19*, *assitência de financiamento*, *-Assistência de financiamento*). Due to the low quality of some translations in the version of SQuAD used, we will consider experimentation in other datasets (e.g., factoid sentences and questions [10], or questions manually produced for SmartEDU). Moreover, we will devise the inclusion of additional methods and explore other language models, not only for ranking, but also for generating distractors. For English, several options are available, such as a T5 transformer fine-tuned for distractor generation [16], given a context, a question and an answer. A similar model could be trained for Portuguese, possibly taking advantage of SQuAD. Generating everything with a language model is indeed more flexible, requires less programming and access to less third-party tools and resources. With some recent models, it can be done with a simple instruction prompt [19], which may additionally include a few complete examples for guiding generation (e.g., few-shot learning). On the other hand, the proposed approach has the main advantage of being transparent. For instance, we can easily track the origin of the distractors and discriminate them by type.

We make the implementation of the generation and ranking methods available from the following notebook:
`https://github.com/NLP-CISUC/smartedu-aqg/blob/main/Generating_Ranking_Distractors_PT.ipynb`

References

1 Dhawaleswar Rao Ch and Sujan Kumar Saha. Automatic Multiple Choice Question Generation from Text: A Survey. *IEEE Transactions on Learning Technologies*, 13(1):14–25, 2018.

2 Shang-Hsuan Chiang, Ssu-Cheng Wang, and Yao-Chung Fan. Cdgp: Automatic cloze distractor generation based on pre-trained language model. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5835–5840, 2022.

3 Rui Pedro dos Santos Correia, Jorge Baptista, Nuno Mamede, Isabel Trancoso, and Maxine Eskenazi. Automatic Generation of Cloze Question Distractors. In *Second language studies: acquisition, learning, education and technology*, 2010.

4 Maria João Costa, Hugo Amaro, Bruno Caceiro, and Hugo Gonçalo Oliveira. SmartEDU: Accelerating slide deck production with Natural Language Processing. In *Proceedings of 28th International Conference on Applications of Natural Language to Information Systems, NLDB 2023*, volume 13286 of *LNCS*, page In press. Springer, 2023.

5 Maria João Costa, Renato Matos, Hugo Amaro, Bruno Caceiro, Alcides Marques, and Hugo Gonçalo Oliveira. SmartEDU: A platform for generating education-support materials. In *Proceedings of the Experiment@ International Conference 2023 (expat'23)*, 2023.

6 Sérgio dos Santos Lopes Curto. Automatic generation of multiple-choice tests. *Unpublished master's thesis). Universida de Técnica de Lisboa, Portugal*, 2010.

7 Valeria de Paiva, Alexandre Rademaker, and Gerard de Melo. OpenWordNet-PT: An Open Brazilian WordNet for Reasoning. In *Proceedings of 24th International Conference on Computational Linguistics*, COLING (Demo Paper), 2012.

8 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of 2019 Conference of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186. Association for Computational Linguistics, June 2019.

9 Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, 1998.

10 João Ferreira, Ricardo Rodrigues, and Hugo Gonçalo Oliveira. Assessing factoid question-answer generation for Portuguese (short paper). In *Proceedings of 9th Symposium on Languages, Applications and Technologies, SLATE 2020*, volume 83 of *OASIcs*, pages 16:1–16:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

11 Shu Jiang and John SY Lee. Distractor generation for Chinese fill-in-the-blank items. In *Proceedings of 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 143–148, 2017.

12 Ghader Kurdi, Jared Leo, Bijan Parsia, Uli Sattler, and Salam Al-Emari. A systematic review of Automatic Question Generation for Educational Purposes. *International Journal of Artificial Intelligence in Education*, 30(1):121–204, 2020.

13 Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale reading comprehension dataset from examinations. In *Proceedings of 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, 2017.

14 Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. DBPedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

15 Bernardo Leite and Henrique Lopes Cardoso. Neural question generation for the Portuguese language: A preliminary study. In *Progress in Artificial Intelligence: 21st EPIA Conference on Artificial Intelligence, EPIA 2022, Lisbon, Portugal, August 31–September 2, 2022, Proceedings*, pages 780–793. Springer, 2022.

16 Potsawee Manakul, Adian Liusie, and Mark JF Gales. MQAG: Multiple-choice question answering and generation for assessing information consistency in summarization. *arXiv preprint arXiv:2301.12307*, 2023.

17 Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of Workshop track of the International Conference on Learning Representations (ICLR)*, 2013.
18 Ruslan Mitkov, Ha Le An, and Nikiforos Karamanis. A computer-aided environment for generating multiple-choice test items. *Natural language engineering*, 12(2):177–194, 2006.
19 NEA Nasution. Using artificial intelligence to create biology multiple choice questions for higher education. *Agricultural and Environmental Education*, 2(1), 2023.
20 Thomas Pellegrini, Rui Correia, Isabel Trancoso, Jorge Baptista, Nuno Mamede, and Maxine Eskenazi. Asr-based exercises for listening comprehension practice in european portuguese. *Computer Speech & Language*, 27(5):1127–1142, 2013.
21 Van-Minh Pho, Anne-Laure Ligozat, and Brigitte Grau. Distractor quality evaluation in multiple choice questions. In *Artificial Intelligence in Education: 17th International Conference, AIED 2015, Madrid, Spain, June 22-26, 2015. Proceedings 17*, pages 377–386. Springer, 2015.
22 Juliana Pirovani, Marcos Spalenza, and Elias Oliveira. Geração automática de questões a partir do reconhecimento de entidades nomeadas em textos didáticos. In *Simpósio Brasileiro de Informática na Educação-(SBIE)*, page 1147, 2017.
23 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
24 Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016.
25 Siyu Ren and Kenny Q Zhu. Knowledge-driven distractor generation for cloze-style multiple choice questions. In *Proceedings of AAAI Conference on Artificial Intelligence*, volume 35, pages 4339–4347, 2021.
26 Dominic Seyler, Mohamed Yahya, and Klaus Berberich. Knowledge questions from knowledge graphs. In *Proceedings of ACM SIGIR International Conference on Theory of Information Retrieval*, pages 11–18, 2017.
27 Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. BERTimbau: Pretrained BERT models for Brazilian Portuguese. In *Proceedings of Brazilian Conference on Intelligent Systems (BRACIS 2020)*, volume 12319 of *LNCS*, pages 403–417. Springer, 2020.
28 Katherine Stasaski and Marti A Hearst. Multiple choice question generation utilizing an ontology. In *Proceedings of 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 303–312, 2017.
29 Cheng Zhang, Yicheng Sun, Hejia Chen, and Jie Wang. Generating adequate distractors for multiple-choice questions. *arXiv preprint arXiv:2010.12658*, 2020.

Web of Science Citation Gaps: An Automatic Approach to Detect Indexed but Missing Citations

David Rodrigues ✉
Iscte - University Institute of Lisbon, Portugal

António L. Lopes ✉
Instituto de Telecomunicações, Iscte - University Institute of Lisbon, Portugal

Fernando Batista ✉
Iscte - University Institute of Lisbon, Portugal
INESC-ID Lisbon, Portugal

Abstract

The number of citations a research paper receives is a crucial metric for both researchers and institutions. However, since citation databases have their own source lists, finding all the citations of a given paper can be a challenge. As a result, there may be missing citations that are not counted towards a paper's total citation count. To address this issue, we present an automated approach to find missing citations leveraging the use of multiple indexing databases. In this research, Web of Science (WoS) serves as a case study and OpenAlex is used as a reference point for comparison. For a given paper, we identify all citing papers found in both research databases. Then, for each citing paper we check if it is indexed in WoS, but not referred in WoS as a citing paper, in order to determine if it is a missing citation. In our experiments, from a set of 1539 papers indexed by WoS, we found 696 missing citations. This outcome proves the success of our approach, and reveals that WoS does not always consider the full list of citing papers of a given publication, even when these citing papers are indexed by WoS. We also found that WoS has a higher chance of missing information for more recent publications. These findings provide relevant insights about this indexing research database, and provide enough motivation for considering other research databases in our study, such as Scopus and Google Scholar, in order to improve the matching and querying algorithms, and to reduce false positives, towards providing a more comprehensive and accurate view of the citations of a paper.

2012 ACM Subject Classification Applied computing → Publishing; General and reference → Verification; Information systems → Digital libraries and archives; Information systems → Enterprise applications; Applied computing → Digital libraries and archives; Information systems → Data cleaning

Keywords and phrases Research Databases, Citations, Citation Databases, Web of Science, OpenAlex

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.5

Funding This work is funded by FCT/MCTES through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the scholarship reference Iscte_SIIC/01/2022, and projects UIDB/50021/2020 and UIDB/50008/2020.

1 Introduction

The number of citations on a research paper is of great value for the researcher and their work, since usually, the more citations a paper has, the higher chances of it being worth reading and having helpful information. Thus, the authors of said paper can also get prestige from the number of citations of their paper and will be more highly regarded. This is useful not only for the author to get recognition by his peers, but also for performance evaluation processes in the researchers' institutions that usually include publications' citations as one of the main metrics. This raises the importance of knowing the right amount of citations that any given paper has.

© David Rodrigues, António L. Lopes, and Fernando Batista;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 5; pp. 5:1–5:11
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, it is not always easy to keep track of citations, even with the existence of big citation databases, considering they too have their shortcomings. Each database has their own source list, which means that some papers that are indexed in a database, may not be found in another, therefore, there usually are unique citations for each database. But sometimes, the unique citations we find, in reality, are missing citations that one of the other databases did not find, because the citing paper is indexed in the database, but it is not tagged as citing the paper. That being the case, the database is missing information that another one has found, leaving us with incomplete data if only one database is used to check for the number of citations of any given paper. Therefore, it is very useful to devise a way to automatically find missing citations between databases, to get a closer depiction of the real number of citations in any given paper. The problem we aim to solve is to find these citations, where an article is indexed in multiple databases, but it does not count as a citation to the research paper we are studying in at least one of them. As a consequence of this problem, finding a way to automatically merge the information from multiple databases could get us a closer depiction of the real number of citations of a given work without having to go through the slow and tedious process of manually assessing these differences.

In this paper we present our first experiment, where we explore the potentially missing citations in Web of Science (WoS) using OpenAlex as a baseline and find some of the citing articles that WoS might be missing. There are two main reasons why we started by using OpenAlex as a baseline: first, it is a free tool that gives us all the information via an API, therefore it is easy to access and to gather information from it; secondly, OpenAlex gathers information from multiple other resources, which means it might have a more complete citation list for a given paper than other indexing databases. Consequently, OpenAlex allowed us to test our approach faster, and helped us make sure that we find missing citations in WoS, before moving on to Scopus and Google Scholar to complete our analysis.

In the next sections, we present the literature review, followed by a step by step description of the approach to achieve the goal of automatically finding missing citations. Section 4 analyzes the results of an experiment performed with a set of more than 3000 articles. Section 5 describes a manual validation process, using a randomly chosen portion of the data we gathered, in order to make sure that our approach was working correctly. Finally, Section 6 presents the major conclusions and pinpoints possible directions for further work.

2 Literature Review

Since each citation database shows strength covering different areas, looking only at one database can be misleading [7] because a database may not have some of the articles that could potentially cite the paper which a researcher is looking at since there are unique citations for each database that the others do not have [5]. Moreover, missing publications from indexed journals can aggravate the problem, for example, in a study from 2006 to 2017 in the journal of enfermeria nefrologica, only 50.2% of the papers were indexed by Scopus [6]. Like Scopus, the other databases also show lack of coverage in some areas, but all of them are working on enhancing their coverage, and getting all the articles they can in their databases, showing some improvement over the years [7]. Nevertheless, neither database is perfect, and for better results, more than one should be consulted.

The importance of having an accurate citation count is partially shown in the research conducted by [9], where they correlated the citation count of a paper with the amount of times that a paper would be consulted, as well as the attention level that the readers had while examining it, showing that the higher citation count, the higher chance of a researcher to read the paper and also pay closer attention to it.

There are two reasons why a paper might be missing citations in a database: either it is because the citing paper is not indexed in the database, or because there was some error while processing the publication's citing data. [1] found that there are a variety of errors that can justify missing citations, and those errors can be committed by both the databases or the authors of the articles. There can be errors in the DOIs or in the references of a paper.

Exploring these errors, [2] tries to automatically find and correct wrong DOIs in the databases, that were wrongfully inserted either by the authors or by the databases themselves. Although no concrete numbers of the total of errors were provided, we can see there are a lot of errors found by the study. Ovid Technologies publisher alone had over 370,000 outgoing citation errors in 2 years. Besides wrongfully written DOIs, [4] reports that databases sometimes, mistakenly, give the same DOI to different articles, and since the DOI should be unique for each article, these errors can make a difference in bibliometric analysis.

A follow-up study reported by [10], tries to find what were the differences of the references lists in Scopus and WoS. While comparing the reference list of around 100.000 papers in both databases, using as a baseline the Elsevier ScienceDirect Article Retrieval API to get the references, they found that WoS had 77.2% of the papers with the same number of references, while 19.3% had fewer references. On the other hand, 96.4% of Scopus papers had the same amount of references. They manually analyzed random papers with different results and saw a variety of different reasons why this happened. Since the different databases themselves, can extract different references list of a paper, this can also explain why there can be missing citations in some databases, because if they don't extract a reference, or do it incorrectly, then the paper that it is being cited won't be found in the database as a cited paper and the citation will be missing.

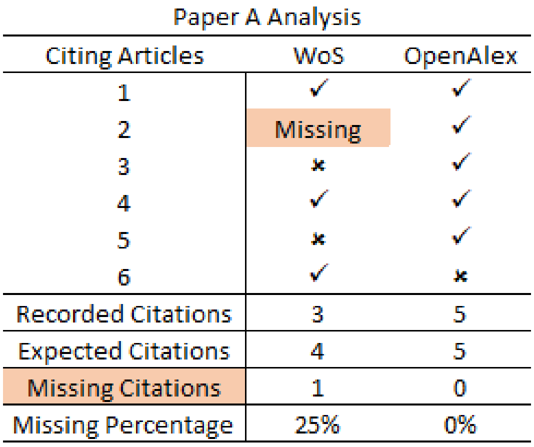
Having now a bigger understanding of some of the problems we might face, and what causes them, we look for research that has been conducted where we can take inspiration from. The research conducted by [5], showed us that using queries to try to find articles through their authors' names, is most of the time a futile endeavor, and might not the best approach, since there are a lot of varieties of the same author name, as well as some databases extract the authors' names very poorly. Also, [3] identifies the missing citations of several articles in an automated way. Our work follows a similar methodology to theirs, the main difference being the way we identify if a possible missing citation paper is present in the database where it is missing from. While they check if the paper's journal or conference title is present in the list of indexed sources of the database in question, we aim to look for the paper itself in the database, making sure it really is indexed by it.

Finally, an advantageous tool we found is OpenAlex, which is a "fully-open index of scholarly works, authors, venues, institutions, and concepts" that allows us, using the DOI of a given paper, to get all the information that they gathered about that paper. OpenAlex gets all their data from multiple services, like MAG and Crossref being the more important ones, and also from ORCID, ROR, DOAJ, Unpaywall, Pubmed, Pubmed Central, The ISSN International Centre and Subject-area and institutional repositories from a plethora of platforms [8]. Using this API, it is possible to get easy access to the information from all these other platforms from a single query.

3 Proposed Method

Our main goal is to automatically find missing citations between different citation databases. In this project, we will be looking at Web of Science, using OpenAlex as a baseline reference. We first need to find all citations that OpenAlex found, but WoS did not. Afterwards, with these potentially missing citations, we have to search WoS to check if each citing paper is indexed in it or not. If it is, then it is considered to be a missing citation.

Table 1 Example of a report for paper A.



Paper A Analysis

Citing Articles	WoS	OpenAlex
1	✓	✓
2	Missing	✓
3	✗	✓
4	✓	✓
5	✗	✓
6	✓	✗
Recorded Citations	3	5
Expected Citations	4	5
Missing Citations	1	0
Missing Percentage	25%	0%

An example of the information we want to find out is shown on table 1. If we analyze paper A, and WoS say that paper A is being cited by the articles 1, 4 and 6, and OpenAlex says that paper A is being cited by the articles 1, 2, 3, 4 and 5, there are 3 citing articles that OpenAlex found and WoS did not, which are 2, 3 and 5. The next step is to look for these 3 articles in WoS, in order to confirm if they are indexed by the database. And, for example, if only article 2 was found to be indexed by WoS, we can assert that article 2 is a missing citation of paper A on WoS.

In order to look for missing citations in an article, all we need is its DOI. The first step in the verification process is to clean the received data. Sometimes, the DOI can have extra information, such as the *https* link, the *doi.org* domain, or other invalid characters or blank spaces, so we standardize the DOI, so it is easier to work with it through the remaining steps.

Afterwards, we ask the institution's Current Research Information System (CRIS) API[1] for more information about this paper. From there, we can get the article's title, authors, year, the type of the article (journal, book, conference, etc.), as well as some information about the WoS representation of this paper, such as if the article is indexed in WoS, the url for the list of citing publications of our article and the unique identifier of WoS (accession number) for this article (WoS ID = WOS:xxxxxxxxxx).

The next step is to query OpenAlex for their information about this article. Once again, the query is done via the DOI, and all the information gathered about the article from the CRIS is compared with the one gathered from OpenAlex. With OpenAlex we can also get their list of citing publications, which comes with all the information they have about each of these citing publications, namely the DOI, title, publication year and the type.

After retrieving the list of citing publications from OpenAlex, we need the corresponding WoS list of citing publications to compare them both. Since the WoS API only provides the link for the page where the information is, we had to extract the information of each citing paper ourselves. We use the link provided by the CRIS in order to retrieve this information, but it would also be possible to get to the web page with the list of citing publications through the WoS unique identifier, since with that, we could build an URL that takes us directly to the WoS page about the paper, and from there we are only one click away from the list of citing publications.

---

[1] `https://ciencia.iscte-iul.pt/api/v2_6/doc`

Furthermore, we could also try to find the publication via the DOI and other information we got, but that method is more prone to errors, therefore, if possible, using the link to the page or the WoS ID is the better solution.

With the list of citing publications from both OpenAlex and WoS, we can look for possible missing citations in WoS. For this effect, we first try to match the OpenAlex publications with the WoS publications, and if both of these databases have the same publication tagged as citing our article, we don't need to do any further work for this article. If instead, the process does not find a match, then this publication is flagged as a possible missing citing publication for the article in WoS.

Having now all of these publications that are flagged as potentially missing citing publications in WoS, we need to check if they are in fact indexed in WoS. And only if they are, then they are considered missing citations. So once again, we must consult WoS to extract this information. This step is harder than the previous one, since before we already had a link to the page of paper we were looking for, while this time, we don't have any WoS information about this paper. In order to find the paper in WoS, we execute two different queries. First, we query using only the DOI of the publication '(query: DO=(doi))'. If this query does not provide results, we also try to find the publication via the title and publication year '(query: TI=(title) AND PY=(year))'. If any of the queries provides results, we extract the results from WoS the same way we extract the list of citing publications, and we double-check if the DOI is the same, or if the title, year and publication type are the same. The reason we don't use only the DOI to perform this check is because some publications don't have DOIs (or the database might be missing that information), but also because sometimes there are errors in the assignment of DOIs in databases, as seen earlier in the literature review. Hence, we might not find the correct publication via DOI, and do so through the title. If the queries' results are a match, then we are in the presence of a missing citation, and we tag it as such in the report we send back to the client. With every missing citation, we also provide the link of the publication we found that is supposed to be citing our article, so that the client can easily double check manually any missing citations they might want to check. These queries are very strict, and specially in the case of the title, if there is an error on a word, or any difference, they will not match, in the future we wish to improve these queries, using similarity techniques that will make it possible to find a match even in the presence of small errors. We would also like to try to expand these queries and use other information we might have on the paper to try to find it in the new database, although this must be carefully tested in order to limit the number of false positives that our application finds.

Through all this process, one of our main goals is not to overload the APIs or repeat processes unnecessarily. With that in mind, we use delays between every request to avoid overloading the systems. Moreover, we have a local database which is used as a cache, so if we are asked to analyze the same publication in a short period of time (the time period of when the data becomes outdated varies from each type of data, and ranges from 1 day to 1 month), we don't have to gather all the information from external sources to provide an answer. Therefore, for the entire process, every time we need any data, we first query our local database, and only if we don't have the data stored locally we will do the process described above in order to get the necessary data.

Finally it is important to state that in order to create a report for any publication, the publication has to be indexed in both databases, so if we can't find it either in WoS or OpenAlex, no report is created. Also, if the publication has no citations in either database, there isn't anything to look for, and no report is created. A diagram with the representation of our system's design can be seen in Figure 1.

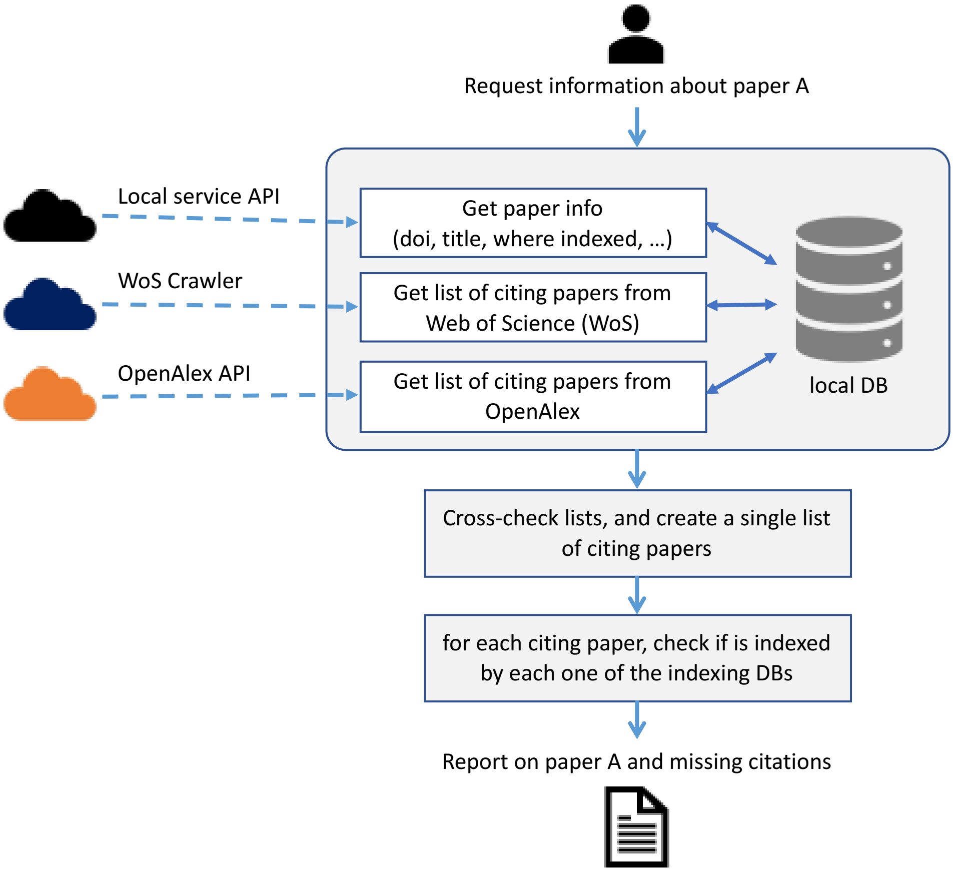

Request information about paper A
Local service API
WoS Crawler
OpenAlex API
Get paper info
(doi, title, where indexed, ...)
Get list of citing papers from
Web of Science (WoS)
Get list of citing papers from
OpenAlex
local DB
Cross-check lists, and create a single list
of citing papers
for each citing paper, check if is indexed
by each one of the indexing DBs
Report on paper A and missing citations

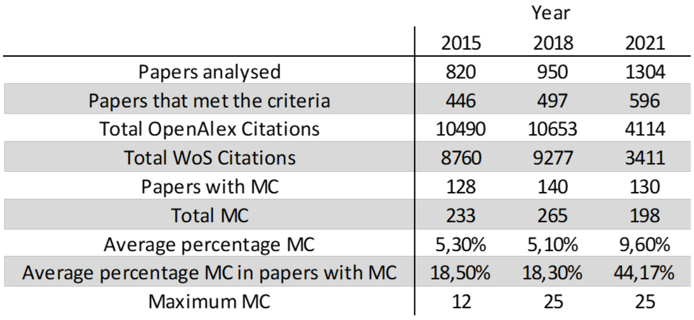
Figure 1 System's design.

4 Evaluation

In order to evaluate our proposed approach, we performed an experiment where we used the papers provided by our institution's CRIS as the dataset. From the 52.000 papers present in the dataset, we can only use 12.500 of them, since the others do not have a designated DOI.

For our experiment we decided to take a year-based approach, which means we carried out the analysis of all papers of a predetermined set of years. We chose the years 2015, 2018 and 2021. We chose these years for a multitude of reasons. First, we wanted years that would be relevant for our motivation, and since our university only examines the papers from the last 10 years to evaluate their researchers' performances, we did not want to look at papers that were older than 10 years. We also wanted to analyze multiple years, not only so we would have more data, but also so we could compare the results from each year and see if we could notice anything changing or even improving over time. The year of 2018 was the first year we explored, since it was before the COVID pandemic that started in 2019, and that was an untypical year that probably would not give us a trustworthy result. The year of 2021 was chosen because we also wanted a more recent year, but not so recent that the publication would not have had time to impact the scientific community, since if from the moment a publication is published, it still has to be read by other members of the community, that have to write and publish their own work. Finally, the year of 2015 was chosen to keep the 3 year gap between 2018 and 2021. A summary of the results from these 3 years is present at Table 2.

Table 2 Results per year.



	Year		
	2015	2018	2021
Papers analysed	820	950	1304
Papers that met the criteria	446	497	596
Total OpenAlex Citations	10490	10653	4114
Total WoS Citations	8760	9277	3411
Papers with MC	128	140	130
Total MC	233	265	198
Average percentage MC	5,30%	5,10%	9,60%
Average percentage MC in papers with MC	18,50%	18,30%	44,17%
Maximum MC	12	25	25

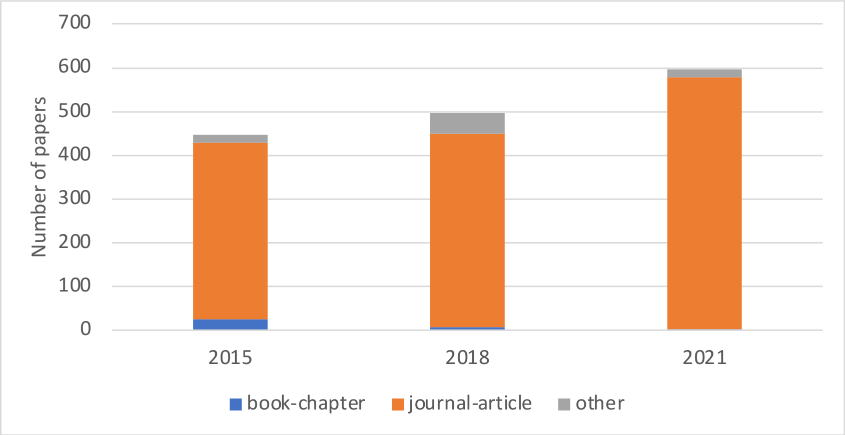


Figure 2 Type of publications being analysed.

Figure 2 shows the number of papers being considered in our experiments for each year, by publication type. It is interesting to observe that most of the papers are of the type “journal-papers”, a tendency that is increasing over the years.

In total, 3074 papers were analyzed by our system, where 820, 950 and 1304 are from the year 2015, 2018 and 2021, respectively. For all the analyzed papers, a report is only created if the paper is indexed in both databases (WoS and OpenAlex) and if the paper has at least one citation in either of them, therefore, for each of the years, only 446, 497 and 596 (respectively) reports were created, giving us a total of 1539 reports of different papers. We can see that the values are increasing over time, not only in the number of papers, but also in the number of reports created, especially in 2021, where there was an increase in over 350 papers and almost 100 more reports when compared to 2018, whereas if we compare 2018 with 2015, the increase was only of 120 papers and 51 reports. While the number of reports did not increase as much as the number of papers, that is probably due to the fact that since 2021 papers are much more recent and maybe some of them are not indexed in both databases yet or they do not have any citations. If we did the same analysis in one year, the data from 2021 would be the one that would change the most.

All the data we are presenting next comes from the papers where a report was created. The total number of citations in WoS for each year were 8760, 9277 and 3411 respectively. The huge drop in the number of citations in 2021, comes from what has been said before, which is that since the papers have had less time to be cited, the number of citations is gonna be lower. The low number of citations, when compared to the other years, also greatly influences the percentages that are presented in Table 2.

In 2015, we found missing citations in 128 out of the 446 papers with a total of 233 Missing Citations (MC). This means that 28,70% of the papers had MC, averaging 1,82 per paper. The numbers in 2018 are very similar, with 140 papers out of 497 having MC (28,17%) and with a total of 265 MC, with 1,89 per paper, a slight decrease in the number of papers with MC but a slight increase of MC per paper. Once again the year of 2021 stands out with only 130 papers out of 596 with MC (21,81%) and with 198 MC, with an average of 1,52 per paper.

$$Average\_percentage\_of\_MC = MC/(MC + Citations)$$

To calculate the average percentage of MC we chose to use the formula presented above, which will tell us, out of all the citations that a paper should have (the ones the database caught and the ones it missed), how many are missing, giving us results from 0% to 100%. We also counted the average of MC for all the papers, and only for the papers that had MC. Out of all the papers, the percentage of missing citations both in 2015 and 2018 rounded the 5%, while in 2021 it was 9,6%. If we look only for the papers with MC in their reports, in 2015 and 2018 we had 18,5% and 18,3% respectively, while in 2021 we got an astounding 44,17%. In 2021 the number shoots up, especially because of the low number of citations in 2021. Once again we argue that these numbers in 2021 are higher because the papers did not have as much time to be cited yet, and the papers that already cited them are still being worked on by WoS, and because the papers are recent, the WoS process is still behind in referencing the citations on the papers. In Figure 3 we present the distribution of papers according to the percentage of missing citations. Although the average number of MC per paper is low (between 1,5 to 1,8), there were some cases where a high number MC were identified as it can be seen in Figure 4. In 2015 a paper had 12 MC, and in 2018 and 2021 both had a paper each with 25 MC each. Only 8% of the analyzed papers had over 3 MC, and 7 papers had 6 or more MC, showing us that most of them have 3 or less MC.

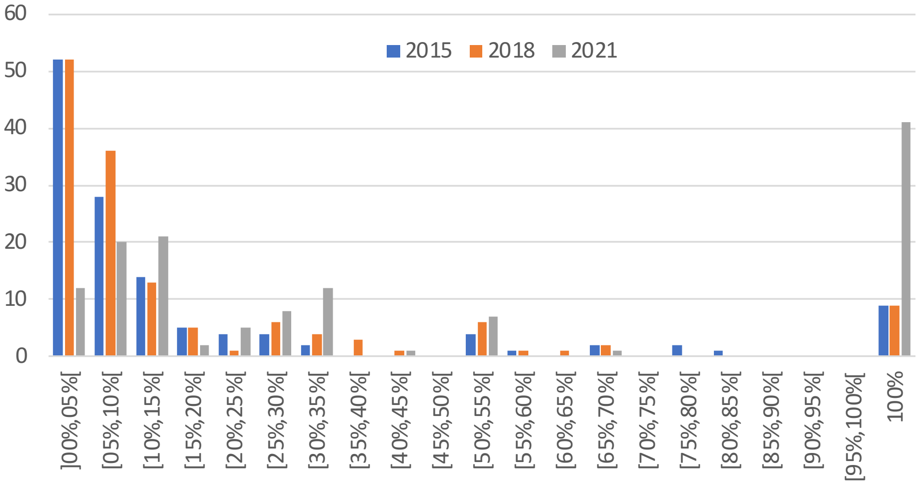


Figure 3 Distribution of papers according to missing citations percentage per year.

5 Results Validation

In order to have more trustworthy results, we manually checked a sample of the reports that we created. Part of that sample was retrieved at random, but another part was retrieved from reports that stand out because of unusual or unexpected results. We manually checked all

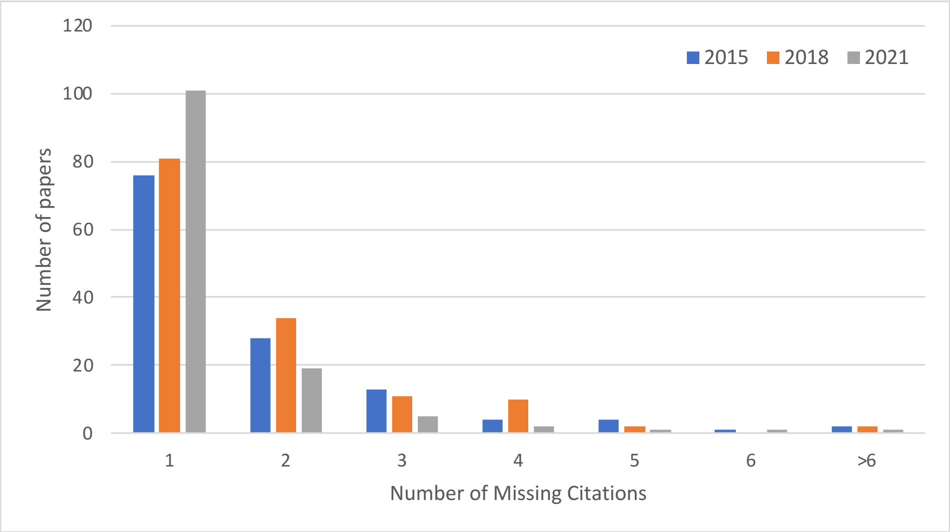


Figure 4 Distribution of papers according with the amount of missing citations per year.

reports that had over 3 MC and where those 3 MC represented at least half of the citations of the paper. This is because if a paper had 4 MC out of 100, then it was quite plausible that WoS might be missing a few citations, while if there were 4 MC out of 5 total citations, then something might be wrong, since it means that WoS did not find most of the citations that it should have.

The manual verification was processed as such for any given report:

- We had the DOI of the paper that was initially provided to our system, as well as the OpenAlex and WoS web links for this paper. Both were double checked to make sure they related to the same paper.
- For each missing citation identified by the system, we checked the OpenAlex and WoS web links for the citing paper that our system matched, in order to make sure that the OpenAlex publication was the same we found in WoS.
- If the original paper being analyzed was the same in both databases, and all the missing citations were correctly matched with a paper indexed in WoS, then it was a missing citation.

We manually verified 40 random papers, from which 36 had correctly identified their 51 MC, while the other 4 reports had some errors.

In one of these reports with errors, out of 7 MC found, 2 of them were wrongly matched, meaning that the OpenAlex citing publication was not the same as the WoS publication we found through our queries. Upon closer inspection we found that the problem was that these 2 publications had the same DOIs in the different databases, and after a search for the DOI in google, it looked to us that WoS had the publication with the wrong DOI.

We also found 2 cases, where one of the missing citations was due to OpenAlex saying that the publication was citing itself, and since the publication was indexed in WoS, it was tagged as a missing citation. Therefore we have a total of 2 wrong MC in these 2 reports.

Finally the remaining report was wrong not due to the fact that it incorrectly matched citing papers, but because of an error in the data received by the CRIS, where the DOI given to the publication, provided wrong information about the publication, which led to the comparison of two different articles, in WoS and OpenAlex.

In this manual search, from the 40 analyzed papers, only 10% had some kind of mistake, one of the cases was due to WoS having assigned the wrong DOI to a publication, and the other one was due to the fact that we tagged the publication as citing itself. The latter is an easy fix we can do in our application so it won't happen again, while the former is a mistake that we cannot work around that easily, since it is a problem in WoS side. Finally the mistake from wrong data provided by the CRIS is out of our reach to fix, since the information in this database is provided mostly by the authors of the publications and it is prone to have some errors.

Afterwards we confirmed the reports that showed over 3 MC which represent at least 50% of the citations of the report itself. Out of 14 total reports, 13 of them had no errors, correctly identifying 82 MC. On the other hand, the remaining report was wrong, due to the fact that the publication we got from OpenAlex, was different from the one in WoS, making our system compare citation lists from two different papers. Therefore this report incorrectly gave us 25 MC. This error came from the CRIS database, which had 2 different DOIs for the same publication, and the DOI being analyzed did not match the information provided by the CRIS.

In general, we are satisfied with the system's performance as it has only encountered a few errors, primarily caused by inaccurate data. Although we can address some of these errors, there are others that are beyond the scope of our approach. As a result, there is a slight chance of misclassifying a Missing Citation as such. To mitigate the it's impact, our reports include links to both databases for any matched papers, allowing for manual verification. This process significantly reduces the effort required compared to starting from scratch, thereby saving researchers a considerable amount of time.

6 Conclusions and Future Work

This paper proposes an automated approach that leverages two indexing databases to locate missing citations in one of them. We focus on the case study of Web of Science (WoS) as the primary citation database, while utilizing OpenAlex as a reference point for comparison. Our method involves identifying all citing papers associated with a given target paper from both research databases. Subsequently, we examine whether each citing paper is indexed in WoS but not referenced in WoS as a citation, indicating the presence of a missing citation. By conducting experiments on a dataset comprising 1539 papers indexed by WoS, we successfully uncovered 696 missing citations using our approach. Although a small part of these missing citations might be wrongfully identified, as shown above, the data suggests that most of them are indeed missing citations, which proves the validity of our approach and the fact that WoS has missing information for citing publications, specially if we take into consideration that only OpenAlex was used as a baseline, while later on, we intend to add both Scopus and Google Scholar, and we expect to detect a higher amount of missing citations in WoS, as well as finding missing citations in Scopus and Google Scholar themselves.

In our analysis we detected that in the later years, the percentage of missing citations skyrocketed, which makes sense since WoS might not have had the time to process the citations of more recent publications, while OpenAlex uses data from other platforms like DBLP and CrossRef, which might have faster algorithms to find missing citations, giving OpenAlex an advantage for more recent publications. Nevertheless, this can be a problem if an author is being evaluated on their more recent publications, making this process more critical for that circumstance. It could be interesting to follow up this research and see how the results have changed, and how long it takes for the WoS algorithm to catch up to the standards of older publications, since the publications analyzed from 2021, were between 1.5 and 2.5 years old when analyzed.

One aspect of our current work that we believe can be enhanced is the matching algorithms employed for publications. Merely augmenting query parameters is not sufficient, as the methodology outlined in this paper predominantly relies on the utilization of the DOI reference system, as well as basic searches that incorporate the title, authors, publication year, and publication type to establish connections between papers and their respective citations. In our forthcoming endeavors, we aim to introduce similarity metrics to refine paper searches, thereby potentially yielding more comprehensive and thorough outcomes.

The next step will be to add other indexing databases to our approach, such as Scopus and Google Scholar (GS), not only to try to find missing citations, but also because they might bring additional citing papers that WoS and OpenAlex might have missed. Especially in the case of GS, which has a broader indexing policy than WoS and Scopus. Moreover, a preliminary analysis has already shown us that there are cases where GS has more citations than OpenAlex for the same publication.

References

1 Robert A. Buchanan. Accuracy of cited references: The role of citation databases. *College & Research Libraries*, 67(4):292–303, 2006. `doi:10.5860/crl.67.4.292`.

2 Alessia Cioffi, Sara Coppini, Arcangelo Massari, Arianna Moretti, Silvio Peroni, Cristian Santini, and Nooshin Shahidzadeh Asadi. Identifying and correcting invalid citations due to doi errors in crossref data. *Scientometrics*, 127:3593–3612, 2022. `doi:10.1007/s11192-022-04367-w`.

3 Fiorenzo Franceschini, Domenico Maisano, and Luca Mastrogiacomo. A novel approach for estimating the omitted-citation rate of bibliometric databases with an application to the field of bibliometrics. *Journal of the American Society for Information Science and Technology*, 64(10):2149–2156, 2013. `doi:10.1002/asi.22898`.

4 Fiorenzo Franceschini, Domenico Maisano, and Luca Mastrogiacomo. Errors in doi indexing by bibliometric databases. *Scientometrics*, 102:2181–2186, 2015. `doi:10.1007/s11192-014-1503-4`.

5 Miguel A. García-Pérez. Accuracy and completeness of publication and citation records in the web of science, psycinfo, and google scholar: A case study for the computation of h indices in psychology. *Journal of the American Society for Information Science and Technology*, 61(10):2070–2085, 2010. `doi:10.1002/asi.21372`.

6 Erwin Krauskopf. Missing documents in scopus: the case of the journal enfermeria nefrologica. *Scientometrics*, 119:543–547, 2019. `doi:10.1007/s11192-019-03040-z`.

7 Henk F. Moed, Judit Bar-Ilan, and Gali Halevi. A new methodology for comparing google scholar and scopus. *Journal of Informetrics*, 10(2):533–551, 2016. `doi:10.1016/j.joi.2016.04.017`.

8 Jason Priem, Heather Piwowar, and Richard Orr. Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts, 2022. `doi:10.48550/ARXIV.2205.01833`.

9 Misha Teplitskiy, Eamon Duede, Michael Menietti, and Karim R. Lakhani. How status of research papers affects the way they are read and cited. *Research Policy*, 51(4):104484, 2022. `doi:10.1016/j.respol.2022.104484`.

10 Nees Jan van Eck and Ludo Waltman. Accuracy of citation data in web of science and scopus, 2019. `doi:10.48550/ARXIV.1906.07011`.

Querying Relational Databases with Speech-Recognition Driven by Contextual Knowledge

Dietmar Seipel
Department of Computer Science, Universität Würzburg, Germany

Benjamin Förster
Department of Computer Science, Universität Würzburg, Germany

Magnus Liebl
Department of Computer Science, Universität Würzburg, Germany

Marcel Waleska
Department of Computer Science, Universität Würzburg, Germany

Salvador Abreu
NOVA–LINCS, University of Évora, Portugal

Abstract

We are extending the *keyword–based query interface* DDQL for relational databases which is based on contextual background knowledge such as suitable *join conditions* and which was proposed in [10]. In the previous paper, join conditions were extracted from existing referential integrity (foreign key) constraints of the database schema, or they could be learned from other, previous database queries.

In this paper, we describe a *speech–to–text* component for entering the query keywords based on the system Whisper. Keywords, which have been recognized wrongly by Whisper can be corrected to similarly sounding words. Again, the context of the database schema can help here.

For users with a limited knowledge of the schema and the contents of the database, the approach of DDQL can help to provide useful suggestions for query implementations in SQL or Datalog, from which the user can choose one. Our tool DDQL can be run in a *docker image*; it yields the possible queries in SQL and a special domain specific rule language that extends Datalog. The Datalog variant allows for additional *user-defined aggregation functions* which are not possible in SQL.

2012 ACM Subject Classification Information systems → Relational database query languages; Information systems → Relational database model

Keywords and phrases Knowledge Bases, Natural Language Interface, Logic Programming, Definite Clause Grammars, Referential Integrity Constraints, Speech–to–Text

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.6

Supplementary Material *Software (Source Code)*: `https://gitlab2.informatik.uni-wuerzburg.de/Wissensbasierte-Systeme/declare/declare.git`
archived at `swh:1:dir:fc9f1d42f6306261bee5a7fae45776845d7b1025`

1 Introduction

The growing wave of *digitization*, which the smart world of the future is facing, could be met by concepts from *artificial intelligence (AI)*. The field of AI can be divided into symbolic and subsymbolic approaches, e. g., [12]. *Symbolic or knowledge–based* approaches model central cognitive abilities of humans like *logic*, deduction and planning in computers – mathematically exact operations can be defined. *Subsymbolic or statistical* approaches try to learn a model of a process (e. g., an optimal action of a robot or the classification of sensor data) from the data. Current knowledge–based information systems are increasingly becoming hybrid,

© Dietmar Seipel, Benjamin Förster, Magnus Liebl, Marcel Waleska, and Salvador Abreu;
licensed under Creative Commons License CC-BY 4.0

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 6; pp. 6:1–6:15
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

including different formalisms for knowledge representation. In this paper, we use concepts from AI and logic programming for answering non–expert queries to hybrid knowledge bases. Still, the most frequent formalism is relational databases, but it would be very interesting to include rule–bases, ontologies and XML databases as well.

It is becoming popular to consider natural language queries [1]. In a simple form, this concept is well–known from *keyword–based queries* in search engines like Google. It can be very helpful for users who are not so familiar with the database schema, and for users on mobile devices, where it is difficult to enter complex–structured queries. For a preceeding speech–to–text transformation, currently subsymbolic approches, e.g. voice/speech assistants such as the commercial systems Siri, Alexa, or Dragon NaturallySpeaking or the publicly available tools Whisper and Mozilla Common Voice/Deep Spech [13] are popular. In this paper, the complicated step of assigning a suitable semantics – i.e. of compiling textual keyword–based queries to correct complex–structured knowledge base queries, e.g. in SQL or Datalog – is done using a symbolic, declarative knowledge–based approach.

In a previous paper on DDQL at SLATE 2021 [10], we had presented the compilation of keywords to SQL or Datalog queries using concepts from logic programming [4, 6, 8]. In this paper, we will add a speech–to–text component that is based on the well–known tool Whisper for speech–to–text conversion. Subsequently, based on the context of the queried database, another text–to–text conversion has been developed to convert the keywords, which might have been understood a little bit wrongly by Whisper, to similar keywords that might have been intended by the user. This is done using concepts from language processing and the context of the database.

The rest of this paper is structured as follows: Section 2 gives an overview on database query languages and intelligent query answering. Section 3 presents a running example of a database instance, i.e. the representation by tables, and a set of relational database queries. Section 4 deals with the speech–to–text conversion with Whisper and the following text–to–text corrections based on the context of the database. Section 5 summarizes our system DDQL for answering keyword–based queries using technology from logic programming and deductive databases. We show how DDQL and Whisper are integrated in a *docker–based* approach to optimize the usability, portability and availability of the tool. Finally, Section 6 concludes with a summary.

2 Database Query Languages and Intelligent Query Answering

Natural language interfaces (NLI) are considered a useful end–user facing query language for knowledge bases, see Affolter et al. [1] and Damljanovic et al. [9]. This is especially true for complex databases and knowledge bases, where the intricacies of both the information schema and the technicalities of the query language – SQL most of the time – put the task of issuing useful queries well beyond the skill of most prospective, non–technical users. NLIs can usually be catgorized into *keyword–*, *pattern–*, *parsing–*, and *grammar–based* systems. Recent case studies are also reported by Stockinger [22] who argues that the trend of building NLI databases is stimulated by the recent success stories of artificial intelligence and in particular deep learning. An important keyword–based system is SODA [3]. Li and Jagadish [15] hold that NLIs are superior to other approaches to ease database querying, such as keyword search or visual query–building. They present the parsing–based systems NaLIR and NaLIX.

The main gripe with a natural language interface is that it is inherently difficult to verify reliably: an ambiguous sentence might be incorrectly parsed and its meaning evaluated, without the end user ever becoming aware of the situation. As a consequence, much effort has been placed into devising *user–friendly* ways of removing the ambiguity and translating the query to a semantically equivalent one in the native database query language. In practice, this entails presenting alternatives to the user and asking him to decide; the process may be iterated.

Doing so with SQL as the target seems a natural choice, but this hits many difficulties arising from the language's many quirks. This situation is exacerbated when one must present the query interpretation back to the user. Relying on a more abstract query language, such as a *first order predicate logic*–based one, turns out to be both easier and more effective, especially as the reflection of the user's utterance interpretation will be presented in a form which is closer to its presumed grammatical structure and therefore easier to recognize and understand. Besides convenience in presentation, relying on a logic representation for the queries and schema has several enabling benefits: a major one is that it provides a unifying framework for heterogeneous sources of information, such as SQL databases but also deductive databases, XML databases, ontologies queried in SPARQL or RDF datasets.

Interpreting a natural language sentence as a database query entails attempting to do several queries, ranging over the *schema* but also the *data* and even the query history. Contextual speech recognition is a very hard problem, which can be eased if one manages to make use of *background knowledge*. The inherent ambiguity in the task of parsing and tagging a sentence in natural language can be mitigated and complemented with concurrent knowledge base queries: domain knowledge may be used to constrain the admissible interpretations as well as to provide useful annotations. Having a logic–based framework also makes it easy to provide views, which may be further used in interpreting natural language queries. The logic dialect needs not be full first–order logic, as Datalog is sufficient to express queries originally formulated in simple natural language.

3 Relational Database Queries

It is difficult for database users to have to remember the strucuture of the database (the database schema) and the correct writing of the terms (table names and attributes) and the values in the tables. Nevertheless, they have a good notion of the queries that they would like to ask. We are proposing an intelligent expert tool for query answering based on the *deductive database system* DDbase of the declarative programming toolkit Declare [20]. We have developed a module DDQL, that can first parse the textual representation of the query using Declare's extended definite clause grammars (EDCG) [19] in Prolog based on the background knowledge of the database schema and the database, then hypothesize the intended semantics of the query using expert knowledge, and finally present possible queries and answers, so that the user can select one.

One could, e.g., imagine the following database queries to the well-known relational database COMPANY from [11] for exemplifying our approach.

$\mathcal{Q}_1$ *Give me the salary of Borg.*
$\mathcal{Q}_2$ *What is the salary of Research ?*
$\mathcal{Q}_3$ *Give me the sum of the salaries of the departments by name.*
$\mathcal{Q}_4$ *Give me the list of the salaries of the departments by name.*
$\mathcal{Q}_5$ *Give me the supervisor name of an employee by name.*

The database user does not say that *salary* is an attribute of a database table or that *Borg* is a value of another attribute. Moreover, there could be slight spelling mistakes.

The complete database schema will be given in Figure 2 in the appendix; the *ER diagram* contains 6 entity/relationship types and 8 referential integrity constraints between them (links given by arrows). Some corresponding database tables will be given in this Section.

In the background, the query compilation in Section 5.1 will extract undirected connected subgraphs from the ER diagram. The relationship types from the corresponding ER diagram of [11] are represented in the database schema as follows:

(a) in the table Employee, the 1:n relationship types Works_For and Supervision from the ER diagram are integrated as foreign keys Dno and SuperSsn (the Ssn of the supersisor), respectively;

(b) the manager of a department is given by the attribute MgrSsn in Department;

(c) the table Works_On gives the employees working on a project, and the attribute Dnum in Project gives the responsible department of a project; both tables are not shown here, but they can be seen in the ER diagram in the appendix.

Functionalities and existency constraints require: every employee works for exactly one department; every department must have exactly one manager; an employee can manage at most one department; every employee must work for at least one project; and every project must have exactly one responsible department. All constraints of the database schema can be used for optimizing queries.

In the following, we show a slightly restricted version of the database, where some entity types and attributes are not present or renamed.

Employee								
Fname	Lname	Ssn	BDate	Address	Sex	Salary	SuperSsn	Dno
John	Smith	4444	1955-01-09	731 Fondren, Houston	M	30000	2222	5
Franklin	Wong	2222	1945-12-08	638 Voss, Houston	M	40000	1111	5
Alicia	Zelaya	7777	1958-07-19	3321 Castle, Spring	F	25000	3333	4
Jennifer	Wallace	3333	1931-06-20	291 Berry, Bellaire	F	43000	1111	4
...	...	...	...	...	...	...	...	...
James	Borg	1111	1927-11-10	450 Stone, Houston	M	55000	NULL	1

The departments and their managers are given by the table Department with the primary key Dno. The 1:1 relationship type Manages is integrated as a foreign key MgrSsn together with the describing attribute MgrStartDate, the start date of its manager.

Department			
Dname	Dno	MgrSsn	MgrStartDate
Headquarters	1	1111	1971-06-19
Administration	4	3333	1985-01-01
Research	5	2222	1978-05-22

The multi–valued attribute Locations of the entity type Department yields a separate table Dept_Locations, which we do not consider here. The table Works_On shows the Hours that the employees work on the projects. The 1:n relationship type Controls between Department and Project is integrated as the foreign key Dnum in Project. Every project is located at one of the locations of its controlling department.

In this paper, we will only go into details for the shown tables Employee and Department.

4 Speech–to–Text with Whisper and Word Corrections

In this section, the possibility of using DdQl in cooperation with a speech–to–text tool is examined in more detail. For this purpose, Whisper, which was released in September 2022 [18], is presented and examined. For DdQl, it is crucial whether especially the keywords are correctly translated by a speech-to-text tool. Therefore DDQL can accordingly continue to work correctly with the text transcribed from the spoken sentence. Whisper is apparently better than the previous open-source speech-to-text tools, with a lower error rate regardless of context. This is also described in the paper on Whisper in addition to an explanation of the training model used to train Whisper. This training model is crucial for the high robustness of the resulting speech-to-text tool [2].

Apart from that, we have deliberately decided against using commercial speech-to-text variants. Experience has shown that voice assistants such as Amazon Alexa, Google Assistant or Apple's Siri work quite reliably, but they have the disadvantage that it is not possible to trace what happens to the data that is transmitted to the company's internal servers. Furthermore we do not want to use ChatGPT due to its fatal privacy issues. Apart from that, it would be possible to run a variant of ChatGPT locally. However, this would be accompanied by significantly increased hardware requirements.

4.1 Technology of Whisper

As already mentioned, Whisper is a software built by a training model for machine learning. This neural network is composed of a so called pipeline model [2]. This approach combined the parallel learning of the aspects *language identification*, *voice activity detection*, *transcription of the spoken to a written text* and *transcription from any available language into english text*. Therefore, the model uses several tokens during the training process to specify, which task especially is handled at the moment. All in all the model was trained with 680.000 hours of audio, of which 117.000 hours are 96 other languages than english. The resulting speech-to-text-tool Whisper is really robust and detached from any context of the spoken text, that would have been usually given by other training models [2].

4.2 Various Aspects of Speech–to–Text with Whisper

In the following, a case study will be presented to give an overview on various aspects of the extent to which Whisper would be suitable for use as a speech-to-text-tool for DDQL. The `employee` table (see Subsection 3) from the already mentioned `company` database gives the basis for this purpose.

The table name, the column names and then all attributes were read out loud three times and recorded in audio files. The values of the attributes `fname`, `minit` and `lname` were read out as triples. The audio files were then evaluated and transcribed individually by Whisper. The Whisper models `base`, `small` and `medium` were used one after the other. In summary, each entry from the table was read out loud three times, recorded and then transcribed by the three mentioned Whisper models. It would also have been possible to read out all the words in the `employee` table in one take to record and then have it transcribed by Whisper. According to the Whisper development team, however, this could have led to an unwanted side effect. Whisper could let already evaluated audio data influence later evaluations within an audio file [2]. Therefore, it was decided to record the different attributes and names individually in the respective audio files and then have Whisper transcribe them.

Since Whisper currently does not have the functionality to configure it easily so that it recognises certain words preferentially by the user, a representative case distinction is now presented below using the `employee` table as an example. This case distinction considers the possibilities that could occur, how Whisper mis-transcribes words and how they would then have to be further processed in order to ensure a meaningful further use by DdQl. Seen in this way, the mappings opposite to the direction of the arrow in the following represent the work of an intermediate tool between Whisper and DdQl in order to make the throughput of spoken queries to the desired results of DdQl as high as possible.

Since DdQl is case-insensitive, capitalisation is ignored in the following. Generally, Whisper transcriptions are complete sentences with punctuation. In the following, it is assumed that Whisper transcriptions are passed as word lists. Therefore punctuation on the edges of the whitespaces is also omitted, i.e. the characters '.', ',', ';', '!' and '?' inbetween words of the word lists are not shown.

The arrows ↦ of the mappings below correspond to the transcription by Whisper. Each left side of the arrows represent a table entry. The right side represents the possible Whisper transcriptions of each entry. The mentioned word lists are shown as words separated by whitespaces. Various transcriptions are separated by the character '|', while various mappings are separated by a semicolon. The mappings only show a selection of the wrong transcriptions of the `medium` model. This model, according to its own information [2], relatively often translated what was said into what was wanted.

In the following, we show for a selection of types of speech-to-text by Whisper and in the following Subsection 4.3 we present the subsequent text-to-text corrections by DdQl depending on the context of the queried database.

Case 1: Separation of Words by Whisper. In the most cases, Whisper basically transcribes the spoken words correctly. Apart from the correct order of letters, Whisper occasionally adds whitespaces or other characters inbetween letters of words, that originally are written as one word.

Examples:

`fname` (spoken f-name) ↦ `f name`; `minit` (spoken m-init) ↦ `m in it`;
`lname` (spoken l-name) ↦ `l name`; `dno` (spoken d-n-o) ↦ `d-n-o`;
`superssn` (spoken super-s-s-n) ↦ `super ssn`

Therefore an intermediate tool between Whisper and DdQl would have to ensure that the keywords are partially reassembled in order to then pass them to DdQl. Incidentally, this should also make it possible to spell out entries letter by letter.

Case 2: Tanscriptions of Similar or Phonetically Similar Words. Due to the fact that Whisper is derived from a speech-to-text driven learning environment, it sometimes transcribes words with smaller deviations. These discrepancies consist of either single or multiple letters and can also take place across several words, as in Case 1.

Examples:

`salary` ↦ `celery`; `bdate (spoken b-date)` ↦ `beat it` | `be date`;
`dno (spoken d-n-o)` ↦ `tno`; `franklin t wong` ↦ `franklin t wom`;
`alicia j salaya` ↦ `alicia j zelaya` | `alicia jade salaya`;
`joyce a english` ↦ `choice a english`;
`ahmad v jabbar` ↦ `amad vi jabar` | `amad vi javar` | `amad vi jabal`;
`james e borg` ↦ `james e bork`; `m` ↦ `um` | `mmm`; `null` ↦ `nah` | `no`

In this case, an intermediate tool between Whisper and DdQl would have to check for a written similarity or even phonetic similarity with the words from the exemplary `employee` table. If similarities are rated very highly according to a percentage value, these could then be passed on to DdQl as word lists accordingly.

Case 3: Speech-to-Text for Numbers. Here, it is shown how Whisper transcribes spoken numbers. Apart from characters like ’<whitespace>’, ’.’, ’,’ and ’-’ that could appear between the digits after the transcription, a closer look reveals that in the following examples with nine identical digits, an additional digit is added occasionally. Furthermore Whisper seems to translate single digits into written-out numbers or even words that are phonetically very close to these numbers.

Examples:

SSN (ID Number): `111111111` ↦ `1 1 1 1 1 1 1 1 1 1`;
`444444444` ↦ `4 4 4 4 4 4 4 4 4`;
`555555555` ↦ `5 5 5 5 5 5 5 5 5 5` | `5-5-5-5-5-5-5`;
`666666666` ↦ `6666666666`; `777777777` ↦ `7777 7777 7` | `7777 7777 7 7`;
Money: `30000` ↦ `3 0 0 0 0` | `30,000` | `3-0-0-0-0`;
`25000` ↦ `25,000` | `2500` | `2 5 0 0 0`; `43000` ↦ `43,000` | `4 3 0 0 0`;
Single Digits: `5` ↦ `five` | `fives`; `4` ↦ `four` | `for`; `1` ↦ `one`

An intermediate tool between Whisper and DdQl could eliminate the separation of the numbers by intermediate characters as described in Case 1. The occurrence of too many or too few digits could be fixed by checking for equality of the first digits compared to the entries of the exemplary `employee` table. This would also make it possible not to have to enter all the digits for a number. For the written-out representation of a number, the intermediate tool would have to pass the equivalent in digits on to DdQl. However, evaluating `for` as `4` and passing it on to DdQl as such would probably not make much sense, since the word `for` occurs frequently in English sentences and would therefor be evaluated as a number each time the speaker uses the word `for`.

Case 4: Transcription of Date Values. Whisper can transcribe dates read out loud number by number. The cases in which the months were pronounced or the spoken order of a date were differed is beyond consideration here.

Examples:

`1955-01-09` ↦ `1955.1.9`; `1945-12-08` ↦ `1945 12 8`;
`1958-07-19` ↦ `1958 7 19` | `1958 719`; `1931-06-20` ↦ `1931 620`;
`1952-09-15` ↦ `1952 9 15`; `1927-11-10` ↦ `1927 11 10`

An intermediate tool for transforming the recognized words to more suitable words would have to derive all possible dates that can be formed from the input and compare them with the values in the database.

Case 5: Transcription of Addresses. Here the transcription of addresses is looked on more closely. It can be seen that similar errors occur here, as they already emerged in the previous cases.

Examples:

`tx` (spoken texas) ↦ `texas`;
`3321 castle spring tx` ↦ `3,3,2,1 castle spring texas` |
`3 3 2 1 kassel spring texas` | `3321 carson spring texas`;
`291 berry bellaire tx` ↦ `291 berry bel air texas`

Additionally `tx` was always transcribed to `texas`, but this is because it was recorded that way. For an intermediate tool, several challenges would arise at the same time, as an address can consist of numbers, words and also abbreviations. Accordingly, an intermediate tool would have to combine several of the approaches from the previous cases. Apart from that, another table could be introduced that divides the address into its different components. This could be handled more easily by an intermediate tool, as described in Cases 1 to 4.

Case 6: Diverse. Proper Nouns like `ProductX`, abreviations like `SSN`, `MGRSSN`, and ambiguities like `SME` (subject meta expert, small medium enterprises) have to be handled. Moreover, `MGR` can abbreviate `Manager`. Nowaday, `Laser`, `X-Ray`, `LGBTQ`, `ChatGPT` and `Open AI` are commonly known names. Furthermore, as in the example above with `tx`, it should be ensured that common designations, such as `USA`, `EU` and `ASEAN` should be recognised and brought into a context processable by DDQL.

In conclusion, it can be said that an intermediate tool between Whisper and DDQL should be mostly dependent on data type in order to effectively enable corrections for DDQL. It is important that all data types that occur in the accessed database are considered. A person with expert knowledge about the database could then, for example in the case of this case study, set the intermediate tool accordingly so that addresses are recognised and handled like a compound data type.

The above mappings were composed mainly in relation to the transcription of the `medium` model of Whisper. In fact, the same categories of error types occurred in the `small` and `base` models, but with higher frequency and partially more bias. This is clearly due to the fact that `small` and `base` are smaller and less accurate speech-to-text models than `medium`, but thus are faster with respect to runtime [2]. When designing the intermediate tool between Whisper and DDQL, this trade-off of accuracy and runtime can be utilized and should be further investigated in the implementation.

4.3 Text–to–Text Corrections with Language Technology

As we have seen in the previous section, it is not only important to correctly interpret a user's request in natural language, but it is also of great importance to correct words incorrectly transcribed by a speech–to–text (STT) engine in the case of spoken requests.

We use a custom fuzzy matching approach, adjusted to the observed wrong transcriptions of Whisper, to counteract this problem. One main part of our fuzzy matching algorithm is based on the fact that STT engines try to identify the correct phonemes of spoken words, e.g. in an audio file, in order to map those phonemes to the correct spelling. Therefore it is much more likely, that words are transcribed incorrectly to similar sounding words, as we saw in, e.g., Case 2 (`salary` ↦ `celery`).

Our *phonetic matching* approach, to find homophones in the database, generates a phonetic key for each word, which tries to represent the most important parts of its pronounciation. This key generation is mainly derived from the known metaphone–algorithm [14].

Hereafter, for every case, mentioned in Subsection 4.2, it will be discussed how our fuzzy matching approach addresses those listed mistranscriptions.

Case 1: Separation of Words by Whisper. The phonetic key generation does not pay attention to any whitespaces, hyphen etc.. Therefore `dno` and `d-n-o` have the same phonetic key, same goes with `f name` and `fname`. If we are not able to find an exact match with a given value in the database, we compare the phonetic keys and pick those with the highest similarity. This reduces the amount of format issues significantly.

Nevertheless, we have to pay attention to those cases, in which word separations differentiate values in the database. For example if there are an `Anna Lyn` and an `Annalyn` in the database. A user may want to find `Anna Lyn`, but Whisper transcribes it incorrectly to `Anna Lin`. All three names have the same phonetic key. As it is more likely that Whisper uses the correct word separations, `Anna Lyn` will be preferred as a match.

Case 2: Transcriptions of Similar or Phonetically Similar Words. Here, several, but not all of the mistranscriptions have the same phonetic key than their origin. `salary` and `celery` have the same phonetic key of `SLR`. `bdate`, `beat it` and `be date` have also the same phonetic key of `BTT`. `franklin t wong (FRNKLNTWNK)` and `franklin t wom (FRNKLNTWM)` or `Pork (PRK)` and `Borg (BRK)` have different phonetic keys. In order to be able to rank the similarities of phonetic keys, we use a custom *Levenshtein Distance*, where we take into account, that some differences of phonetic keys are more likely to occur than others. It is much more likely, that a difference of `P` and `B` in the phonetic keys is based on a mistranscription of Whisper than `P` and `L`. Therefore differences like `P` and `B` contribute to a much lesser degree to the calculated distance than differences like `P` and `L`.

We also pay attention to those characters in the near surrounding of a detected difference. Keys like `NL` from the word `null` and `N` from `no` get a lower distance than `NK` from `nik` and `N`. It is much more likely, that Whisper mistranslates `null` to `no` than it does with `nik`.

These adjustments are based on the pronounciation itself and our testing with Whisper. Of course these rules are dependent of the used language. Right now, we only consider english as spoken language. It would also be possible to deal with other languages, but for every other language, an own phonetic algorithm has to be created.

Case 3: Speech–to–Text for Numbers. In the case of numbers, all spelled out digits are first converted to their corresponding digits. Then, based on the amount of consecutive digits or the given context, it is verified whether this could be a date. If this is not the case, all consecutive digits, which are separated only by whitespaces or similar, are merged. Then we search in the database for numbers that contain the given number, or if the given number contains a number in the database. This gives us the opportunity, to address numbers, which are not known in their entirety. If the user only knows the first few digits of the `SSN` of an employee, it is possible to ask the user if a certain `SSN`, containing the given number, was the `SSN` he meant. If there is more than one match with a high similarity, the user can choose for which of them he wants to get the according result.

It is not recommended to combine this approach with the Levenshtein Distance, because in our testing, Whisper usually did not mistranscribe one digit into another. It was more likely that Whisper translated a `4` into `for`. Hence we consider cases, in which `for` was not changed and cases, in which `for` was changed to `4`. Those results are preferred, that did not change anything.

Case 4: Transcription of Date Values. The usual way for a user to provide dates, is to pronounce the month or year directly instead of spelling it out digit by digit. This works quite well with Whisper. But if just the according digits are mentioned, we verify if a certain set of digits could be a date. The different aspects we consider are the length and the value of the number. From those possible date representations with the wrong format, we create all dates that are eligible, compare them with the values in the database and ask the user if he meant this specific date. `1931 620` for example, can only be `1931-06-20`.

Case 5: Transcription of Addresses. Many approaches discussed so far help us to interpret a misunderstood address the correct way. `berry bellaire tx (BRBLRTX)` and `berry bel air texas (BRBLRTXS)` have very similar phonetic representations, our algorithm will take this kind of mistranslations into account. As we are also looking for matching substrings, it would also be possible to respond to from Whisper transcribed queries like *Give me all employees, who live in berry bel air texas.*

Case 6: Diverse. Abbreviations that can occur in the database are addressed by a simple dictionary, which stores those words and their according abbreviation.

It is not only possible, to compare words according to spelling or pronounciation, it is also possible to compare words according to their meaning. If we have a look at the query *How much does the employee Borg earn*, then `earn` and `salary` have a similar meaning. This can give us the opportunity to interpret a wider range of variations in the formulation of queries in natural language correctly.

5 The Declarative Database Query Language DDQL

In DDQL, the generation of queries is based on declarative concepts from logic programming. The knowledge-based compilation of keyword–queries to Datalog and/or SQL is done in three steps. In experiments with the `company` database, useful queries were generated; if a database does not contain referential integrity constraints, then we will need query logs for deriving suitable join conditions.

The declarative programming toolkit Declare [20] and its deductive database system DDbase already have functionality for evaluating database queries formulated using extensions of Datalog. Even hybrid queries including different knowledge representation formalisms are possible in DDbase. Relational databases can be accessed using Datalog or SQL queries and ODBC; for XML processing, a query, transformation and update language FNQUERY is given in [21]. In DDbase, Datalog programs are *evaluated bottom–up* with *stratified fixpoint* computation, and they can – possibly, for simple forms of aggregation – be compiled to SQL queries; often Datalog programs can also be evaluated top–down like in Prolog. For the evaluation in logic programming, the field notation atoms are compiled to ordinary, logical Datalog atoms based on background knowledge about the database schema. The ordinary Datalog rules can be compiled to SQL with DDbase, if there is no default negation – and for stratified default negation or aggregation.[1] For non–stratified default negation, DDbase could use *answer set* solvers, cf. [5], if there are no aggregation literals.

5.1 Knowledge–Based Compilation of Queries

DDQL compiles a NL query $\mathcal{Q}_N$ to a Datalog query $\mathcal{Q}_D$ in three steps:

$$\mathcal{Q}_N \to Q_A \to \mathcal{Q}_F \to \mathcal{Q}_D.$$

The result tables are shown in a graphical interface in Figure 1. First, using Definite Clause Grammars (DCGs, see, e.g., [4, 8]), an annotated query $\mathcal{Q}_A$ is generated. Using, e.g., the following grammar rule in the extended DCG formalism introduced in [19], also the resulting parse tree can be computed:

```
query ==> aggregation , of , attribute , of , table .
```

---

[1] A stratified evaluation requires that none of the embedded atoms A_i is mutualle recursive with the head atom A. Then, the program is split into strata, such that default negated literals or aggregation literals refer to lower strata; the strata are evaluated successively, beginning with the lowest.

The DCG rules are fully interleaved with database access operations of DDbase using ODBC. E.g., the derivations of `attribute` and `table` can result in ODBC calls, or – for potential speed–up – calls to a cached collection of facts previously extracted from the database. Some words of the query – such as "`of`" and "`the`" – are ignored. DDQL generates the annotations one after the other on *backtracking*, starting with the most likely annotations. E.g., the query $\mathcal{Q}_1$ with the key words "`salary, of, Borg`" is first annotated to the following query $\mathcal{Q}_A$:

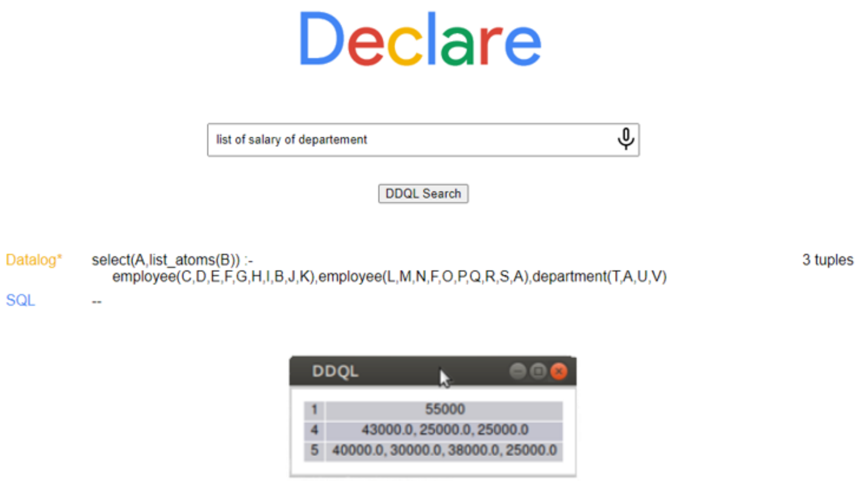
```
salary=company/employee/attribute
'Borg'=company/employee/row(@LNAME)
```

The keyword "`of`" is ignored. In DDQL, it can be detected easily from the database schema that `salary` is an attribute of the relation `employee`. The location of `'Borg'` has to be done based on the contents of the database, which is more expensive. This can be done depending on the context of the table `employee`; it turns out that it is the value of the attribute `LNAME`. After the first annotation has been done and the first solution to the query has been produced, DDQL uses backtracking to generate further annotations and solutions. Of course, then DDQL will also search for `'Borg'` in other tables of the database. In summary, the annotation $\mathcal{Q}_N \to \mathcal{Q}_A$ depends on – and can be refined by – the speech–to–text recognition by Whisper and the text–to–text corrections.

Then, the compilation of the annotated queries $\mathcal{Q}_A$ to SQL or Datalog is done by adding suitable *join conditions* using technology from DDbase based on the database schema. As an intermediate representation, conjunctive queries $\mathcal{Q}_F$ in Datalog are generated with atoms in field notation. The conjunctive queries are then refined and optimized to ordinary Datalog queries $\mathcal{Q}_D$ using background knowledge from the database schema or Datalog–likes rules. $\mathcal{Q}_D$ could be evaluated on a deductive variant of the relational database or a deductive database; an SQL variant of $\mathcal{Q}_D$ can be evaluated on the relational database.

In SQL, queries usually have the form `select-from-where-group by`: The `select` part specifies the attribute values that we are interested in and can contain aggregations of attribute values. The `from` part specifies the tables that should be used to obtain the answer. The intelligent `where` part describes the join-conditions for the envolved tables. For transforming $\mathcal{Q}_A$ to $\mathcal{Q}_F$ and $\mathcal{Q}_D$, in many cases, suitable join conditions can be inferred from the foreign key constraints given in the database schema. The schema of the database `company` contained many foreign key constraints. For databases without given foreign key constraints, join conditions can be inferred from previous SQL queries in the log file: a join condition can be assumed, if the primary key of a table (all attributes of the primary key) is joined with some attributes of another table. In Declare, the schema of a table can be extracted automatically from a running relational MySQL database systen and presented in XML to the user and analysed with Prolog.

Obviously, query $\mathcal{Q}_1$ can simply be computed from table `employee`, since `salary` is an attribute of `employee` and `Borg` is an attribute value in this table of the attribute `lname`. Sometimes the word `Borg` could have been entered or spoken wrongly as, e.g., `Bork` with an ending "`k`". Unless corrected by similarity search, the query would find no result. For $\mathcal{Q}_2$, the situation is more complicated. `Research` is an attribute value in the table `department`, and DDQL has to find the suitable join condition `employee.DNO = department.DNUMBER`. Query $\mathcal{Q}_3$ requires the aggregation function `sum` on the `salary` values, and the departments have to be `grouped by name`. The *relevant links* between the concepts mentioned in a user query might be an undirected tree, or even a *cyclic graph*. E.g., if the user asks for the salary of all employees working in a department that is controlling a given project and that is located in a given city, then two different link trees might be used.



list of salary of departement

Datalog*
```
select(A,list_atoms(B)) :-
    employee(C,D,E,F,G,H,I,B,J,K),employee(L,M,N,F,O,P,Q,R,S,A),department(T,A,U,V)
```
3 tuples

SQL --

1	55000
4	43000.0, 25000.0, 25000.0
5	40000.0, 30000.0, 38000.0, 25000.0

Figure 1 Graphical User Interface of DDQL: Query $\mathcal{Q}_4$.

5.2 The Graphical User Interface

A prototype of the graphical user interface (GUI) of DDQL is shown in Figure 1. The query keywords can be entered by speech after pressing the microphone icon or typed separated by blanks into the input box. Then, the a list of possible search queries in Datalog or SQL is generated, which might be further optimized. Currently, a corresponding SQL query is shown, if there are no aggregation functions.

Obviously, for user–defined aggregation functions, only the Datalog variant is possible. Figure 1 shows such a case with the user-defined aggregation function `list` for the query $\mathcal{Q}_4$. For this example, only one Datalog answer was found, which leads to the result table shown in the figure. Here, a unique answer was returned by DDQL. The join condition between the first and the third Datalog atom of the query is given by the middle atom for `employee`: The two occurences of `F` for the social security number identify the two `employee` atoms, and `A` in the `department` atom ensures the SQL condition `employee.DNO = department.DNUMBER`.

5.3 Integration of Whisper into Declare in a Docker-Based Tool

To integrate Whisper with DDQL, several prerequisites must be met. First, Whisper must be installed. For this, we need a current version of Python together with the PyTorch framework and FFmpeg. For DDQL, we need the Declare developers toolkit, which contains DDQL to be installed along with SWI–Prolog version 7.6.4 and ODBC to access the database. In addition, we need a Python script which was developed to provide the command line interface and connect Whisper and DDQL. To connect DDQL to the desired database, we need to manually create a DSN with the required information. Moreover, the client environment must be able to offer an X display for the graphical component of Declare to work.

Since the installation relies on a delicate balance of versions for all dependencies, the process is somewhat cumbersome. On top of that, several other inconveniences may occur. Whisper seems to have trouble with other installed Python packages. To avoid this, Whisper should be installed in an isolated virtual environment. Another inconvenience could be the operating system used itself. It can influence the installation of the individual components and can lead to unexpected behavior, since not all parts are tested on every variant of every operating system.

To increase usability as well as other features such as portability and availability, several approaches can be considered. As a first step, we have *dockerized* Declare[2] to conveniently provide a Declare distribution with all its dependencies. To use Whisper we either install it as mentioned in the description above or use an existing Docker image of Whisper. However, we need to install Python to run the script, which needs some small modifications, to provide the CLI for the tool. We also have to create the DSN to connect DDQL with the database. Another approach could be to combine all the required dependencies into a single Docker image, so that only Docker needs to be installed to use the tool via CLI. Then, the database connection can be created inside the Docker container by passing all the required information as environment variables. This maximizes the portability and availability of the program, as we can easily retrieve the Docker image from any machine and run it as needed. To further optimize the usability of this approach, we could integrate the python script into a web server that runs inside this single Docker container. With this step, we are able to install it both on a local machine or on a server to provide the tool to the user with an appropriate web page.

6 Conclusions and Future Work

In this paper, we have shown how queries to relational databases can be answered in keyword–based natural language interfaces using intelligent, cooperative techniques. The concepts mentioned in the query are linked based on contextual background knowledge, mainly from the database schema. In the future, also concepts from subsymbolic AI will be investigated and included where useful. For instance, by looking at the user behaviour from previous queries, we may derive heuristics for finding intended queries (e.g. linking atoms) using some form of *machine learning*.

We have containerized Declare – inluding DDQL – in a docker image and added a *voice recognition* part based on Whisper. In this paper, we have described the techniques used for recognizing the correct queries in the context of the database. DDQL can benefit from interleaving it with Whisper. Later, we are planning to use the voice recognition also for DDQL on mobile devices and the knowledge acquisition could be done with a voice assistant based on a domain–specific language, see [17]. Currently, we are training the Whisper tool on different databases with varying schemata and instances. We would also like to add voice output, i.e. spoken answers, such that the returned table could be read to the user. At the moment the tool Whisper can transcribe 99 languages and we are planning to add multilinguality to DDQL.

Graphical user interfaces (GUIs) might also be acted upon. Then, the GUI is the context; it might also be described in a structured form such as HTML 5 [16] or the XML user interface language XUL [7], which has been used by Mozilla for the firefox web browser; then text matching has to be done on the interface components with reflection over a GUI. In Declare, XUL interfaces can already be interpreted, and we are working on Prolog links in HTML.

---

[2] `https://hub.docker.com/r/declaredocker/declare`

References

1 Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. A Comparative Survey of Recent NLIs for Databases. *VLDB J.*, 28(5):793–819, 2019. `doi:10.1007/s00778-019-00567-8`.

2 Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, Ilya Sutskever. Robust Speech Recognition via Large-Scale Weak Supervision, 2022. arXiv preprint arXiv:2212.04356. URL: `https://arxiv.org/pdf/2212.04356.pdf`.

3 Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. SODA: Generating SQL for Business Users. *Proc. VLDB Endowment*, 5(10):932–943, 2012. `doi:10.14778/2336664.2336667`.

4 Ivan Bratko. *Prolog Programming for AI.* Addison–Wesley Longman, 4th edition, 2011.

5 Gerhard Brewka, Thomas Eiter, and Mirek Truszczynski. Answer Set Programming at a Glance. *Communications of the ACM*, 54(12):92–103, 2011.

6 Ceri, Stefano and Gottlob, Georg and Tanca, Laetitia. *Logic Programming and Databases.* Springer, 1990.

7 Christian Schneiker, Dietmar Seipel. Declarative Web Programming with Prolog and XUL. In *Proc. 26th Workshop on Logic Programming (WLP)*, 2012.

8 William Clocksin and Christopher S. Mellish. *Programming in Prolog.* Springer Science & Business Media, 2003.

9 Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. NLIs to Ontologies: Combining Syntactic Analysis and Ontology–based Lookup through the User Interaction. In *Extended Semantic Web Conf.*, pages 106–120. Springer, 2010.

10 Dietmar Seipel, Daniel Weidner, Salvador Abreu. Intelligent Query Answering with Contextual Knowledge for Relational Databases. *10th Symposium on Languages, Applications and Technologies (SLATE)*, 2021.

11 Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, 3rd Edition.* Addison–Wesley Longman, 2000.

12 Ben Goertzel. Perception Processing for General Intelligence: Bridging the Symbolic/Subsymbolic Gap. In *International Conference on Artificial General Intelligence*, pages 79–88. Springer, 2012.

13 Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep Speech: Scaling up End–to–End Speech Recognition. *arXiv preprint arXiv:1412.5567*, 2014.

14 Lawrence Philips. Hanging on the Metaphone, Computer Language, Vol.7, No.12, 1990.

15 Fei Li and H. V. Jagadish. Understanding Natural Language Queries over Relational Databases. *SIGMOD Rec.*, 45(1):6–13, 2016. `doi:10.1145/2949741.2949744`.

16 Matthew MacDonald. HTML 5: The Missing Manual, 2nd Edition. O'Reilly Media, Inc, 2011.

17 Falco Nogatz, Julia Kübert, Dietmar Seipel, and Salvador Abreu. Alexa, How Can I Reason with Prolog? (Short Paper). In *Proc. 8th Symposium on Languages, Applications and Technologies (SLATE 2019)*, 2019.

18 OpenAI. Introducing ChatGPT and Whisper APIs. URL: `https://openai.com/blog/introducing-chatgpt-and-whisper-apis`.

19 Christian Schneiker, Dietmar Seipel, Werner Wegstein, and Klaus Prätor. Declarative Parsing and Annotation of Electronic Dictionaries. In *Proc. 6th International Workshop on Natural Language Processing and Cognitive Science (NLPCS 2009)*, 2009.

20 Dietmar Seipel. Declare – A Declarative Toolkit for Knowledge–Based Systems and Logic Programming. `http://www1.pub.informatik.uni-wuerzburg.de/databases/research.html`.

21 Dietmar Seipel. Processing XML–Documents in Prolog. In *Workshop on Logic Programming (WLP 2002)*, 2002.

22 Kurt Stockinger. The Rise of Natural Language Interfaces to Databases. ACM SIGMOD Blog, 2019. URL: `https://blog.zhaw.ch/datascience/the-rise-of-natural-language-interfaces-to-databases/`.

A Appendix

We have used the relational database COMPANY from [11] for exemplifying our approach. The database schema in Figure 2 contains 6 entity/relationship types and 8 referential integrity constraints between them (links given by arrows). Some corresponding database tables have been shown in Section 3 earlier in this paper.

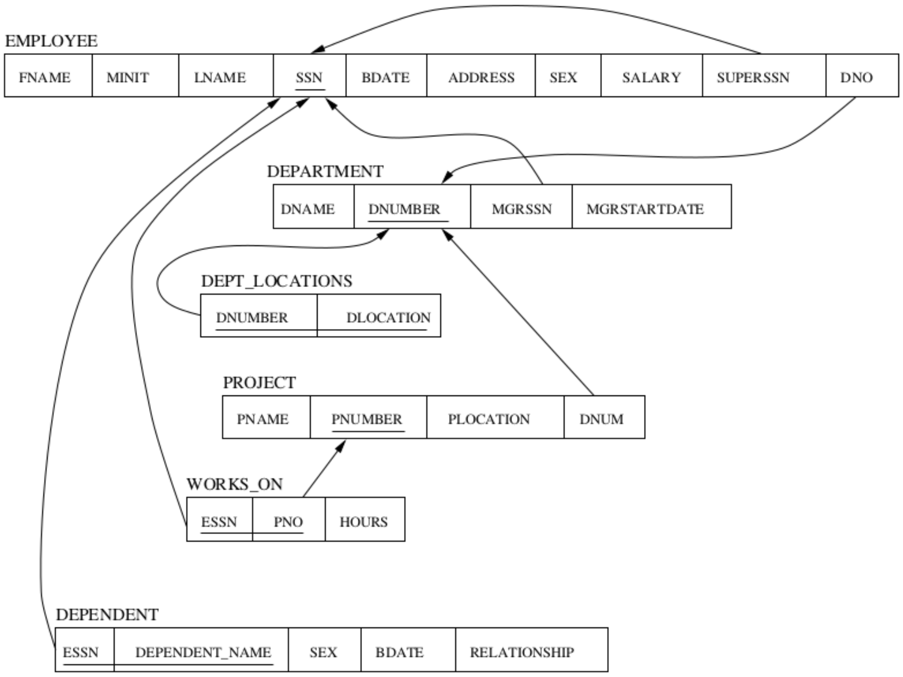


Figure 2 Referential Integrity Constraints for the Relational Database COMPANY.

Automatic Speech Recognition of Non-Native Child Speech for Language Learning Applications

Simone Wills
Radboud University, Nijmegen, The Netherlands

Yu Bai
Radboud University, Nijmegen, The Netherlands
NovoLearning, Nijmegen, The Netherlands

Cristian Tejedor-García
Radboud University, Nijmegen, The Netherlands

Catia Cucchiarini
Radboud University, Nijmegen, The Netherlands

Helmer Strik
Radboud University, Nijmegen, The Netherlands

Abstract

Voicebots have provided a new avenue for supporting the development of language skills, particularly within the context of second language learning. Voicebots, though, have largely been geared towards native adult speakers. We sought to assess the performance of two state-of-the-art ASR systems, Wav2Vec2.0 and Whisper AI, with a view to developing a voicebot that can support children acquiring a foreign language. We evaluated their performance on read and extemporaneous speech of native and non-native Dutch children. We also investigated the utility of using ASR technology to provide insight into the children's pronunciation and fluency. The results show that recent, pre-trained ASR transformer-based models achieve acceptable performance from which detailed feedback on phoneme pronunciation quality can be extracted, despite the challenging nature of child and non-native speech.

2012 ACM Subject Classification Human-centered computing → Human computer interaction (HCI); Applied computing → Education

Keywords and phrases Automatic Speech Recognition, ASR, Child Speech, Non-Native Speech, Human-computer Interaction, Whisper, Wav2Vec2.0

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.7

Category Short Paper

Funding The project ST.CART is funded by the European Regional Development Fund (ERDF).

Acknowledgements Special thanks go to all the children who participated, their parents, their teachers, and the schools.

1 Introduction

In the field of education there is considerable interest in developing and employing voicebots to support children in fundamental skills like learning to read or in acquiring foreign languages. However, this is one of the areas in which the performance of automatic speech recognition (ASR) systems is still lacking. In general, current speech-based systems work relatively well for adult native speakers, with the performance dropping considerably when it comes to other target groups such as elderly speakers, children, non-native speakers, and patients with pathological speech [4, 8]. There are two main factors which contribute to this; firstly the inherently challenging nature of the the speech produced by these speakers and secondly, the limited availability of the speech data needed to develop and improve these systems.

© Simone Wills, Yu Bai, Cristian Tejedor-García, Catia Cucchiarini, and Helmer Strik;
licensed under Creative Commons License CC-BY 4.0

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 7; pp. 7:1–7:8
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In terms of child speech, ASR performance is hindered by speech characteristics which include high degrees of variability, linguistic differences as a result of different stages of language acquisition, and the marked presence of disfluencies and hesitation. This is in addition to physiological differences, such as shorter vocal tracts, which cause a mismatch between the adult and child speech [15]. For applications geared towards supporting the acquisition of a second language (L2) or foreign language by children, the task is doubly challenging in that it deals not only with child speech, but that on non-native speech which is equally as challenging for ASR systems [9]. These challenges are exacerbated by the limited availability of speech data, a particularly prominent problem when working with languages other than English [13]. Collecting speech data is also often hampered by the difficulty in obtaining approval for data collection as well as difficulty in gaining access to the speakers, as they are generally less independent.

Existing ASR-based educational applications built for adult L2 learners have demonstrated the utility of ASR output to provide immediate feedback on correct pronunciation based on single utterances [10, 16]. Despite the challenges, there are few educational ASR-based educational products which have been developed for children learning foreign languages. [11] built a system for children with pronunciation difficulties to practice in their native languages, where children get feedback, in the form of correct / incorrect, from an ASR system . In [17] English pronunciation of short utterances is scored based on tone, volume, timbre and speed.

In this paper we report on research conducted within the framework of a project aimed at investigating the usability of ASR technology for developing innovative educational applications for children learning a second language. In general, ASR technology works better when the text to be recognized is known in advance, as in the case of read speech. However, using only read speech considerably limits the possibilities for developing educational applications, which would indeed be confined to learning to read. For developing more creative applications, less prepared and more spontaneous speech would be required, but this makes the ASR even more complex [12].

To gain insight into the potential of ASR for these tasks in a language other than English, we conducted experiments in which we analyzed extemporaneous speech produced by non-native children learning Dutch through traditional and more recent ASR systems. The research questions we addressed are: *RQ1* How do current state-of-the art ASR systems fare on extemporaneous, Dutch non-native child speech, and *RQ2* how does their performance compare to that on native and read speech? We also consider what kind of information can be automatically extracted from such recordings to gain insights into language proficiency, such as measures of pronunciation quality, articulation and fluency.

The paper is organized as follows. Section 2 describes the data we used, while Section 3 describes the ASR system evaluations. We present both the analysis and ASR performance results in Section 4, followed by a discussion thereof in Section 5. The paper concludes with final remarks and possible future work in Section 6.

2 Speech Material

In this study, we use speech material taken from the JASMIN corpus [7]. This is a Dutch-Flemish corpus of around 90 hours of contemporary Dutch speech from groups of speakers not represented in the existing Spoken Dutch Corpus (Corpus Gesproken Nederlands; CGN). This includes children, non-native speakers, and elderly people. One of the corpus aims was to collect speech from human-machine interactions. The corpus includes manual orthographic transcriptions and phonetic transcriptions produced automatically using ASR.

For our study we selected the Dutch human-machine dialogues of children aged between 11 and 18 years old, separated into two groups; native Dutch child speakers and non-native children learning Dutch. This corresponds to component-p of Group 2 and 3 in the corpus structure. The total selected data contains 9 hours and 21 minutes of dual-channel speech recordings, produced by 94 children (1).

In these dialogues children are questioned about activities they enjoy and are stimulated to provide answers on the fly. The dialogues are conducted in a Wizard-of-Oz scenario, that is that the role of the computer is played by a human being in disguise and the answers provided by this person are sounded through TTS to make them sound like computer-generated speech. The resulting speech is considered extemporaneous speech, as opposed to spontaneous, because the children were prompted through questions to produce answers rather than generating speech on their own initiative.

These recordings were selected because they reflect the realistic challenges which are encountered when applying ASR to turn-taking conversations with children, as one might expect in language learning practice. Additionally, these dialogues were designed such that the children would produce hesitations and dysfluencies, as is often the case in extemporaneous human-machine communication.

Table 1 Overview of Data.

	No. Participants	Female	Male	Age (Years)	Duration (Hour)
Native	41	21	20	12-18	4.48
Non-Native	53	28	25	11-18	4.87
Total	94	49	45	11-18	9.35

3 Methodology

3.1 ASR Systems

We compare the performance of two state-of-the-art open-source ASR systems: Wav2Vec2.0 [1] and Whisper AI [14]. Both systems employ openly available pre-trained models . Wav2Vec2.0 is a transformer-based model pre-trained on unlabelled audio, which can then be further fine-tuned. In this paper we use a cross-lingual model[1], which has been trained on multiple languages and fine-tuned on Dutch. The Whisper ASR system is a recent release by OpenAI, and is an encoder-decoder transformer model pre-trained on multilingual data. Word-level time-stamps are obtained using the WhisperX Python library [3].

3.2 Speech Characteristics

We are not only interested in how well the different ASR systems perform in decoding the audio, but whether this output can be reliably used to automatically extract information about the children's language proficiency. We calculated a number of measurements, for each speaker, relating to speech characteristics. The measurements and their calculations are presented in Table 2.

---

[1] `https://huggingface.co/FremyCompany/xls-r-2b-nl-v2_lm-5gram-os`

Table 2 Speech Characteristic Measurements.

Measurement	Calculation
Number of utterances	segments of speech surrounded by significant pause
Number of words	count of all word tokens
Number of phones	count of all phone tokens
Number of filled pauses	count of all filled pause words (e.g. "hmm", "uh", "ehm")
Vocab Size	count of unique word tokens
Average number of words per utterance	number of words / number of utterances
Average number of phones per utterance	number of phones / number of utterances
Total word duration (seconds)	sum of all word durations
Average word duration (seconds)	total word duration / total number of words
Articulation rate	number of phones / total word duration
Total speech duration (seconds)	sum of all utterance durations

We first calculated these measures through manual transcriptions and then used the Whisper output to calculate a subset of these measures, so that we can compare manual transcription-based measures to those based on ASR output. For the Whisper output, the text was also lower-cased and punctuation removed for measurements replying upon word forms, such as vocab size and filled pause identification.

3.3 Pronunciation Evaluation

Pronunciation information was extracted using the NovoLearning ASR system. This is a back-end ASR system in a Dutch automatic Reading Tutor application which "listens" and provides feedback to children. The ASR takes prompts and speech recordings as input and gives results on both word level and phone level. The ASR analyses speech and gives confidence score represented by probabilities on each phone of the words in the prompt [2]. The probability score ranges from 0 to 100. We passed the native and non-native dialogue speech with the manual transcripts to the NovoLearning ASR. Unrecognized words and filler words were filtered out. We calculated the mean confidence score and standard deviation (SD) of each phoneme for native and non-native speech. We used the phone probability scores to represent the pronunciation quality of each phoneme.

4 Results

4.1 ASR Performance

The word error rate (WER) for Whisper versus Wav2Vec2.0 on the native and non-native speech are shown in Table 3. WER measures the accuracy of the text output of an ASR system by computing the percentage of words which it transcribed incorrectly, determined through comparison with a 'gold standard' (typically manual) transcription. A higher WER indicates more inaccuracies. As seen 3 Whisper outperforms Wav2Vec2.0, so we further compared the performance of Whisper on Read vs. Dialogue speech. The WERs for Read speech are lower than those for Dialogue, with the WER for native Read speech being particularly low compared to the others.

Table 3 WERs (%) using two ASR models for native and non-native Dialogue Speech.

Model	Native	Non-native
Wav2Vec	44.80	44.60
Whisper	30.70	33.80

Table 4 WER (%) of Whisper for Read vs. Dialogue Speech.

	Read	Dialogue
Native	8.00	30.70
Non-native	24.80	33.80

Table 5 Speech Characteristics: Median and InterQuartile Range (IQR) values for manual transcriptions.

Speech Characteristic	Native		Non-native	
	Median	IQR	Median	IQR
Number of utterances	40	10	52	18
Number of words	105	60	141	84
Number of phones	375	147	482	251
Number of filled pauses	3	7	14	12
Vocab size	68	37	84	34
Average words / utterance	2.51	1.35	2.82	0.66
Average phones / utterance	9.39	3.99	9.44	1.90
Total word duration	39.97	13.31	53.40	31.31
Average word duration	0.38	0.22	0.38	0.37
Articulation rate	9.59	1.48	8.94	1.54
Total speech duration	48.93	15.79	68.94	29.96

Table 6 Speech Characteristics: Median and InterQuartile Range (IQR) values for Whisper output.

Speech Characteristic	Native		Non-native	
	Median	IQR	Median	IQR
Number of words	220	68.5	262	57
Number of filled pauses	1	1.5	3	4
Vocab size	129	20.5	145	24
Average words / utterance	4.25	0.70	4.17	0.85
Total word duration	57.95	22.95	72.66	13.30
Average word duration	0.26	0.04	0.28	0.02

4.2 Speech Characteristics

Table 5 and Table 5 present the speech characteristic measurements we calculated based on the manual and Whisper ASR-based transcriptions, respectively. Using the manual transcriptions, we see that for several measures, in comparison to native-speakers, the non-native speakers have higher median values (the number of utterances, words, phones, and filled pauses, total speech duration, total word duration, and vocabulary size). Articulation rate has a lower value, and average number of words and phones per utterance have similar values in the two groups. A subset of measurements calculated using Whisper output are given in Table 6. While the values are all higher when compared to the manual-based values, but the same trends are reflected in the data, with non-native speaker values being higher than native speakers' for the same characteristics as above.

4.3 Pronunciation Evaluation

Table 7 Mean and SD values for the phoneme confidence scores for native and non-native speech.

	Native		Non-native			Native		Non-native	
	Mean	SD	Mean	SD		Mean	SD	Mean	SD
a	95.69	12.85	92.32	15.61	b	97.07	8.38	96.65	8.87
aa	96.97	11.26	96.32	11.59	d	97.51	8.94	97.10	10.27
aw	95.12	11.67	88.13	18.29	f	96.30	7.55	93.83	12.47
ax	94.60	15.16	95.21	12.91	g	34.18	27.08	44.99	28.36
eh	90.99	19.97	90.58	19.27	hh	75.94	35.26	76.32	32.41
ei	96.73	10.84	96.79	8.83	k	98.23	7.26	97.52	9.43
eu	98.42	4.59	85.25	24.39	l	97.84	8.06	96.52	9.87
ey	90.73	22.71	90.75	21.43	m	97.27	8.10	96.18	9.70
ih	94.84	12.30	92.79	15.21	n	97.48	8.57	97.56	7.99
iy	98.12	6.14	92.70	15.72	ng	97.78	9.52	96.37	10.58
oh	92.08	17.38	87.04	19.91	p	98.05	7.15	97.48	9.30
ow	96.80	10.24	93.65	14.46	r	97.20	9.11	95.29	12.31
uh	93.28	12.72	86.64	19.77	s	98.74	4.51	96.57	11.20
uu	86.91	20.89	72.64	27.91	sh	95.67	8.43	72.16	28.79
uw	84.57	29.93	77.13	31.42	t	97.14	10.26	96.29	12.71
uy	97.31	6.64	87.25	19.10	v	98.43	6.17	97.40	9.94
					wv	94.68	15.63	93.68	15.52
					x	97.90	8.26	97.75	8.63
					y	89.42	25.05	85.99	26.53
					z	98.28	5.31	95.66	12.75
					zh	96.24	NaN	70.87	40.07

Table 7 shows that the mean probability scores of most phonemes are similar between native and non-native speech. Most phonemes are above 90. The probability of phoneme "g" is low in both groups (34.18 for native speech, and 44.99 for non-native speech). We looked into the words containing the "g" sound, most of them are English words since the "g" sound does not occur in Dutch words. The only two Dutch words containing the "g" sound are words "stokbrood", "honkbal" and its plural form "honkballen". The "g" sound in these words is a result of voice assimilation. The voiceless velar "k" changes to a voiced velar stop. However, the velar stop "g" is different from the English "g", which leads to low probability. the confidence scores for "hh" and "y" are slightly lower than those of other consonants in both groups. For native speech we observe slightly higher scores in vowels like "eu", "uu" and "uy". However, score differences for these vowels between the two groups are still below 20.

5 Discussion

In the present study we investigated the performance of the latest state-of-the-art ASR systems on Dutch child speech to determine to what extent this technology can be employed to process extemporaneous speech produced by non-native children learning Dutch. As to our research question on how these ASR systems fare on extemporaneous, Dutch non-native child speech (*RQ1*), the results indicate that recent, pre-trained transformer-based models like Wav2Vec2.0 and Whisper obtain reasonable performance. Of these two transformer-based models, Whisper achieves the better performance. While not further explored in this paper, Whisper also has the advantage of producing punctuation. As to our second research question (*RQ2*) regarding the performance difference between native and non-native, read and extemporaneous speech, the results show that ASR performance is better for native than non-native speech, as expected. Similarly, better results were observed for read speech over extemporaneous speech. In particular, the results for read speech appear to be much better for the group of native speakers. There are two possible explanations for this finding. As native speakers, these children can read much better in Dutch. In addition, their realizations of the Dutch sounds are more accurate, which leads to better ASR performance.

To gain a deeper understanding of these differences, we extracted information about several aspects of speech quality that provide insights into language proficiency. We first analyzed differences in temporal measures such as articulation rate, duration and fluency and vocabulary-based measures such as number of words and vocabulary size. Temporal and fluency measures indicate that native speakers are much more fluent than non-native speakers. They speak faster and produce fewer dysfluencies, which is in line with previous research findings [6]. With respect to the amount of speech produced, the results show that the non-native speakers are surprisingly more talkative. They seem to produce longer replies to the questions posed by the computer, which contain more words, but these words appear to be shorter and often interrupted.

For pronunciation quality, however, we did not see considerable differences between native and non-native speakers. ASR-based confidence scores for individual speech sounds do not seem to differ between the two groups of native and non-native speakers, which is surprising.

It is clear that further research is needed to gain more insights into the differences between native and non-native speech and the way in which these affect ASR performance. Nevertheless, is is interesting to see that many of the results obtained through manual transcriptions do not differ considerably from those based on ASR output. This means that this technology has the potential to facilitate future analyses of non-native speech in the context of language learning and language evaluation as well as the design and development of innovative language learning applications.

6 Conclusions

The results of the present study on the suitability of state-of-the-art ASR technology for Dutch non-native read and extemporaneous child speech allow us to conclude that in spite of the considerable challenges recent, transformer-based models open up new perspectives for applying this technology in educational applications. The use of ASR technology also makes it possible to extract additional measures that provide insights into language proficiency. As has been shown previously [5] completely correct performance is not necessarily required to be able to realize pedagogically sound language learning applications.

References

1 Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

2 Yu Bai, Ferdy Hubers, Catia Cucchiarini, and Helmer Strik. ASR-Based Evaluation and Feedback for Individualized Reading Practice. In *Proc. Interspeech 2020*, pages 3870–3874, 2020. `doi:10.21437/Interspeech.2020-2842`.

3 Max Bain, Jaesung Huh, Tengda Han, and Andrew Zisserman. Whisperx: Time-accurate speech transcription of long-form audio. *arXiv preprint arXiv:2303.00747*, 2023.

4 Mohamed Benzeghiba, Renato De Mori, Olivier Deroo, Stephane Dupont, Teodora Erbes, Denis Jouvet, Luciano Fissore, Pietro Laface, Alfred Mertins, Christophe Ris, et al. Automatic speech recognition and speech variability: A review. *Speech Communication*, 49(10):763–786, 2007. Intrinsic Speech Variations. `doi:10.1016/j.specom.2007.02.006`.

5 Catia Cucchiarini, Ambra Neri, and Helmer Strik. Oral proficiency training in dutch l2: The contribution of asr-based corrective feedback. *Speech Commun.*, 51:853–863, 2009.

6 Catia Cucchiarini, Helmer Strik, and Lou Boves. Quantitative assessment of second language learners' fluency: comparisons between read and spontaneous speech. *The Journal of the Acoustical Society of America*, 111 6:2862–73, 2002.

7 Catia Cucchiarini, Hugo Van hamme, Olga van Herwijnen, and Felix Smits. Jasmin-cgn: Extension of the spoken dutch corpus with speech of elderly people, children and non-natives in the human-machine interaction modality. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May 2006. European Language Resources Association (ELRA). URL: `http://www.lrec-conf.org/proceedings/lrec2006/pdf/254_pdf.pdf`.

8 Joost Doremalen, Catia Cucchiarini, and Helmer Strik. Optimizing automatic speech recognition for low-proficient non-native speakers. *EURASIP Journal on Audio, Speech, and Music Processing*, 2010, January 2010. `doi:10.1155/2010/973954`.

9 Roberto Gretter, Marco Matassoni, Daniele Falavigna, A Misra, Chee Wee Leong, Katherine Knill, and Linlin Wang. Etlt 2021: Shared task on automatic speech recognition for non-native children's speech. In *Interspeech*, pages 3845–3849, 2021.

10 Denis Liakin, Walcir Cardoso, and Natallia Liakina. Learning l2 pronunciation with a mobile speech recognizer: French/y/. *Calico Journal*, 32(1):1–25, 2015.

11 Ikuyo Masuda-Katsuse. Pronunciation practice support system for children who have difficulty correctly pronouncing words. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

12 Susana Perez Castillejo. Automatic speech recognition: Can you understand me? *Research-publishing. net*, 2021.

13 Martin Raab, Rainer Gruhn, and Elmar Noeth. Non-native speech databases. In *2007 IEEE Workshop on Automatic Speech Recognition & Understanding (ASRU)*, pages 413–418, 2007. `doi:10.1109/ASRU.2007.4430148`.

14 Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*, 2022.

15 Martin Russell and Shona D'Arcy. Challenges for computer recognition of children's speech. In *Workshop on speech and language technology in education*, 2007.

16 Cristian Tejedor-García, David Escudero-Mancebo, Valentín Cardeñoso-Payo, and César González-Ferreras. Using challenges to enhance a learning game for pronunciation training of English as a second language. *IEEE Access*, 8:74250–74266, 2020. `doi:10.1109/ACCESS.2020.2988406`.

17 Shelley Shwu-Ching Young and Yi-Hsuan Wang. The game embedded call system to facilitate english vocabulary acquisition and pronunciation. *Journal of Educational Technology & Society*, 17(3):239–251, 2014.

OCRticle - a Structure-Aware OCR Application

Sofia G. Rodrigues dos Santos
Informatics Department, University of Minho, Braga, Portugal

J. João Dias de Almeida
ALGORITMI/LASI, University of Minho, Braga, Portugal

Abstract

While there are currently many applications and websites capable of performing Optical Character Recognition (OCR), none of the widely available options offer structured OCR, i.e., OCR that maintains the text's original structure. For example, if a document has a title, after performing OCR on it, the title should have a different formatting, in order to distinguish it from the rest of the text.

This paper covers the topic of structure-aware OCR, first by describing the current state of OCR tools, then by showcasing a prototype tool capable of retaining the structure of articles scanned from an image.

2012 ACM Subject Classification Applied computing → Optical character recognition

Keywords and phrases OCR, Optical Character Recognition, Data Structure, Data Parsing, Document Structure

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.8

Supplementary Material *Software (Source Code)*: `https://github.com/RisingFisan/OCRticle`
archived at `swh:1:dir:651451c61ae5fca1265a703ed38eab264bb82551`

Funding This work has been supported by FCT Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

Currently, most Optical Character Recognition (OCR) tools are limited to a textual output. In other words, if someone is scanning an image that contains text with varying font sizes, indentation or colors, for example, inputting this image into an OCR tool will generate a string of pure text without any of that "extra" information. When scanning documents like newspaper or magazine pages, it becomes important to retain that information, otherwise it becomes much harder to identify features of the text like titles or captions.

The main goal of this paper is to provide a critical assessment of OCR tools, their strengths and limitations, and to showcase a prototype tool capable of performing OCR while keeping the original text's structure as intact as possible.

2 State of the Art

This section covers the current state of OCR tools and some of their main features. It also mentions and describes Tesseract, one of the most used OCR engines nowadays.

2.1 OCR Tools

Most available OCR tools can be described as "basic", and a simple online search for the term "online OCR" will reveal hundreds of websites that perform this type of OCR. Using these tools, a user can upload a file, usually an image, and the tool will process it and return a string of text that it found on the image. These tools offer very little customization or

© Sofia G. Rodrigues dos Santos and J. João Dias de Almeida;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 8; pp. 8:1–8:14
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

extra features, as they are meant to be used by anyone, regardless of previous knowledge in character recognition software, with most of them only allowing users to select the language in which the text is written and if the output file should be a text file, a PDF file or a Microsoft Word file, for example [12][14].

Table 1 contains a list of some popular OCR tools [10] and the main features supported by each. This list only includes tools that can be used for free. Paid tools tend to offer most, if not all of these features, but we did not find any additional feature offered by a paid tool that was not also part of a free tool. By default, all of these tools support textual output.

Table 1 Comparison of some OCR tools.

Tool	Recognize text in Portuguese	Auto-rotation	Table recognition	Restrict OCR to part of file/image	Create searchable PDF	Create DOCX file
OCRSpace [11]	X	X	X		X	
FreeOCR [3]						X
OnlineOCR [12]	X	X	X		X	X
Google Lens [4]	X	X		X		
Text Grab [19]				X		

One of these OCR tools that stands out from the rest is Google Lens [4]. This tool, available on mobile devices, allows the user to use their smartphone's camera to capture an image containing text. After taking this picture, the user can interactively select and copy the text recognized in the picture. Google Lens also excels at recognizing handwritten text and other kinds of text that traditional tools struggle with [20], mainly due to Google's investment in Machine Learning algorithms.

Some tools also offer the possibility to create a special PDF file containing the original scanned document, but with an invisible text layer above the images [5]. This way, users can still see the original document but also select the text found in it, as if it were a regular PDF file. This method is a possible solution to structure-aware OCR, and is used by search engines in order to find text in PDF files composed only of images [6]. However, the text itself is not structured, it only appears to be, thus it is not the best solution for structured OCR.

It's worth noting that different tools have different use cases, and therefore might not have the need for all of the features shown in Table 1. For example, Google Lens uses a smartphone's camera to capture and detect text, unlike the other tools, and Text Grab allows the user to take a screenshot of their computer and immediately recognize text from the captured image, so it makes sense that these tools wouldn't focus on features like creating searchable PDF files.

2.2 OCR Engines

Although the aforementioned tools perform optical character recognition, they are merely wrappers for OCR engines. An OCR engine is the software responsible for recognizing the characters in an image and converting them to text [1]. Applications like Text Grab use these engines to bring OCR functionality to their tools, and are meant to be more user-friendly and intuitive than purely using an engine.

One of the most widely used OCR engines is called Tesseract. It can be used directly via command line, through a 3rd party tool or by using an API written for a programming language [15]. Examples of Tesseract APIs include Python-tesseract [13] or Tesseract.js [17], which can be used by programmers to create applications with OCR functionalities.

Tesseract supports dozens of languages [18] and offers a wide assortment of options [15]. One of these is related to page segmentation, and affects how Tesseract detects text in an image. By default, Tesseract tries to divide the original picture into segments, which can be titles or columns of text, for example. Then, it performs OCR on each of those segments. If the output format is a textual format, this will mean that the final text will be split into segments. However, each one of these segments has the same font and size, making it impossible to distinguish between a title and a regular sentence, for example, without additional context. This default behavior can be modified or disabled, if one desires.

Additional options include output formatting, which can be a PDF file, a text file or an hOCR file (HTML compatible file, with additional information about the original text's structure), for example, or an option to let Tesseract detect words from a user-provided list.

Since Tesseract is mainly an OCR engine, its image processing capabilities [16] may not be adequate for all use cases, which leads many users to apply pre-processing effects to their images before inputting them in Tesseract, in order to improve the text detection [9]. These may include changes in contrast, brightness, size or even converting the image to black-and-white, to avoid issues with colored text/backgrounds.

3 Problem Definition

This paper proposes the creation of a program capable of performing Optical Character Recognition on a file, while maintaining its original structure. For this purpose, a prototype called **OCRticle** was developed, in order to showcase the benefits of this approach to OCR. Unlike already existing solutions that convert a file to a PDF file with invisible text, this tool creates a purely textual file. In order to have a structured textual file, this application uses Markdown [2], a markup language [7], in the generated files.

3.1 Program Specifications

OCRticle is a desktop application developed in the Python programming language. This decision stems from the authors' experience with Python and the ability to easily and rapidly create a command-line or graphical application with Python. Tesseract is used for the OCR component of the application, along with `pytesseract`, Python's Tesseract API [13].

The application focuses primarily on detecting text from newspapers or magazine pages, essentially pages with one or more articles, as its name implies. In practice, it can scan any type of document, but it might not be able to preserve its structure entirely.

OCRticle has a Graphical User Interface (GUI), developed using the Kivy framework [8], where a user can select a source image containing the article(s) they want to convert. Then, they can select each article's position within the image. This allows the tool to focus less on discerning between different articles and more on formatting each one independently. After this step, the tool performs OCR on each image section and display the results to the user, who can then save the detected text in a Markdown file or perform some minor adjustments to the final document's structure.

3.2 System Architecture

The system follows an architecture similar to the one shown in Figure 1.

OCRticle is composed of a front-end and a back-end. The front-end is responsible for user interaction and uses the Kivy module to render a GUI. After receiving user input, the front-end sends data to the back-end, the "brain" of the program. The back-end processes

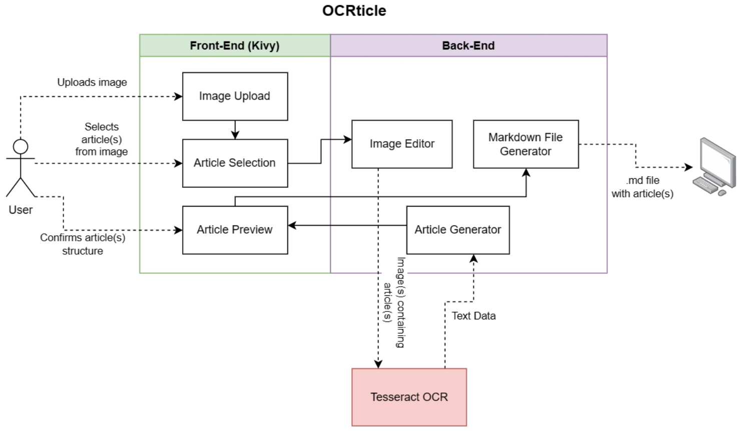


Figure 1 System architecture diagram. Interactions within OCRticle are represented with solid lines and interactions with the "outside" are represented with dotted lines.

data and generates results, in this case, a formatted article or group of articles. It also communicates with Tesseract, sending it images and receiving data about the text present in them.

4 Development

This section covers the development steps of OCRticle.

The first step in development consisted of creating a simple program capable of converting an image into an intermediate representation (IR). This IR holds information about the text and additional context about the OCR process, for example, the dimension of the bounding box where the text was located. The pytesseract module already has a method `image_to_data` which can convert an image into a data structure (e.g., TSV file, dictionary or `pandas` dataframe). However, this data structure contains more data than necessary, so it became necessary to simplify it and make it easier to navigate and access. Hence, this information is filtered and converted into instances of Python classes.

4.1 Classes

Four classes were created for this purpose: Article, Block, Paragraph and Line. The last three classes simply mirror the information returned by pytesseract, which splits text into pages, blocks, paragraphs, lines and words. Since this tool will only work with individual pages, the first category is not needed, and the fifth category, words, is represented as a list in the Line class, since there is no need to store extra information about each word. Besides a list of words, the Line class also contains information about its height. This information is useful because Tesseract might sometimes classify two lines as being part of different paragraphs, or two paragraphs as being part of different blocks, which is not always the case. If we assume that lines of the same paragraph and paragraphs of the same block should have text of the same size, by storing each line's height, we can compare the height of different lines and paragraphs, and infer if two lines/paragraphs should be part of the same paragraph/block. For this purpose, the Paraphaph class contains a `get_line_height` method, which returns the average height of all its lines. Similarly, the Block class contains an equal method, but one which returns the average height of its paragraphs' lines.

Blocks have an additional attribute, `type`, which can be one of four different values: TITLE, TEXT, QUOTE, or CODE. These types should match the role of the text inside the block in the original article. When saving the final file generated by OCRticle, each block type has a different representation, according to the Markdown syntax [2].

Figure 2 illustrates the internal representation of an article.

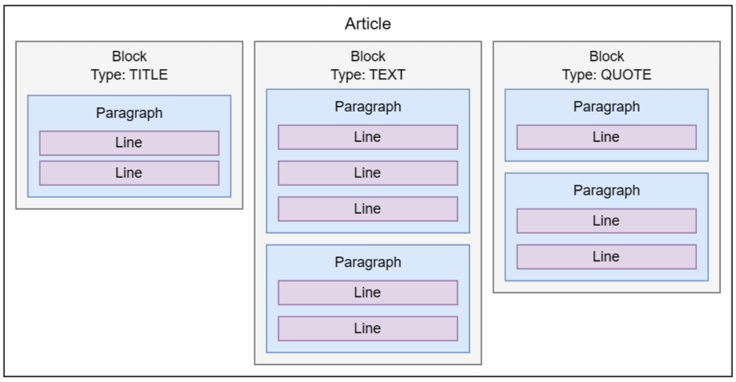


Figure 2 Article internal structure - example.

This particular article is composed of three blocks, one of which is the article's title, another a block of text, and the third one a quote.

4.2 Graphical User Interface

After these classes were developed, the main step in creating OCRticle arose, building the GUI.

4.2.1 Image selection and preprocessing

When opening the application, the interface contains a window where the user can select an image to be analyzed. This step can be skipped if OCRticle is opened through the command line and given a file path as an argument.

After selecting an image, the next window of the application allows the user to select the articles present in the image by drawing rectangles over them. Optionally, users can select areas for the program to ignore. For example, if the original articles contain images with text, one could select the image as an area to be excluded, so that Tesseract won't detect the text in the image. Additionally, if an image has many articles and the user only wants to run OCRticle on some of them, they can just select the articles that they want from the image, instead of having to perform OCR on the entire document or manually cropping the image.

There are also options to control the image's brightness, contrast, and saturation. This is particularly useful if the original image is not well-lit or has a lot of colors in it.

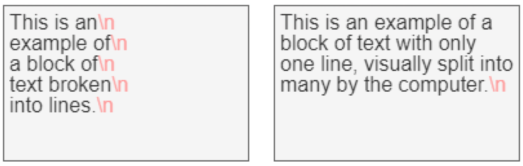
4.2.2 Text detection and formatting

After the user selects the articles in the image, OCRticle divides the image into segments, based on the drawn rectangles, and feeds those segments to Tesseract, which proceeds to detect the text within those images. Then, the tool creates instances of the Article class, each containing the text from a different article. When creating these instances, the program perform an optimization step, where it tries to group together different blocks or paragraphs, based on how they end. For example, if a paragraph ends with a hyphen and the next one begins with a lowercase letter and both paragraphs have the same line height, it probably means that Tesseract failed to detect both lines as part of the same paragraph, and the program will try to automatically fix that mistake.

In addition, OCRticle also tries to detect an article's title. To do this, it analyses every block from the scanned text and finds the one with the biggest font size, which it then tags as the article's title. In the final file, this is represented with a pound sign (#) before the block, which is used in Markdown to define a heading. If every block has a similar font size, OCRticle doesn't do anything, to avoid mislabeling a block, although this can still happen if the block with the biggest font size is not the article's title.

In case of mislabeling, OCRticle allows the user to manually label each detected block before saving the final file. These labels correspond to the block types mentioned previously. OCRticle also joins blocks that it believes are similar enough to be of the same type. For example, if an article has two columns, each with some paragraphs, and Tesseract returns one block for each column, OCRticle will try to figure out if those blocks have the same structure and should therefore be merged. If OCRticle doesn't merge two blocks, for some reason, the user can perform that merge automatically, though a button in the article preview screen.

Another feature offered by OCRticle, which is not present in most OCR tools, is the possibility to include or exclude line breaks inside each paragraph. Typical OCR tools, when scanning a document, will preserve each line in the original image as a separate line in the final text. However, in the original document, the text is only split into multiple lines due to the limited size of the paper. On a computer screen, which is typically wider than a piece of paper, it might not make sense to preserve the original line breaks. Therefore, after scanning a picture, there's an option in the application to remove the original line breaks, and keep each paragraph as a single continuous line. This behavior mimics computer text editors, like Microsoft Word, where the line breaks are artificially created by the software in order to make the text fit in the screen, while the text remains stored as a single line. Figure 3 illustrates this difference, with the newline characters highlighted in red.



```
This is an\n
example of\n
a block of\n
text broken\n
into lines.\n
```

```
This is an example of a
block of text with only
one line, visually split into
many by the computer.\n
```

Figure 3 Distinct options for line breaks in output file. The first option mimics the original image.

Line breaks will always be removed in titles because headings in Markdown must consist of a single line.

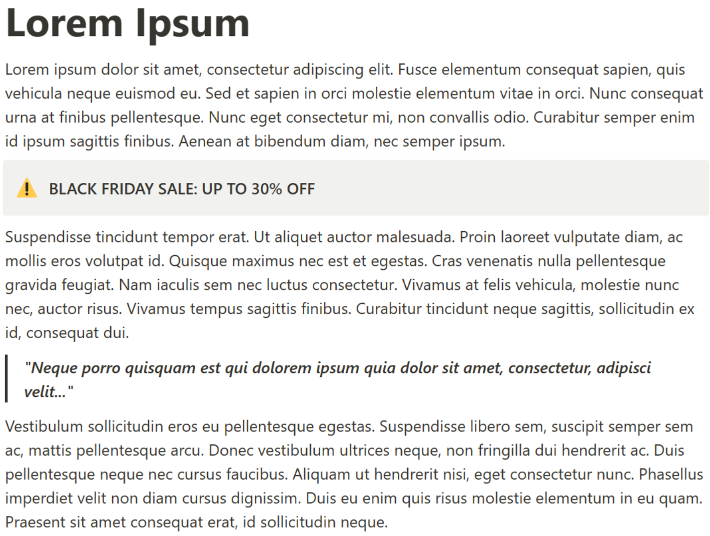
4.2.3 File saving

After a user has confirmed the labels for each articles' blocks, OCRticle allows them to save the formatted text as a Markdown file. This file can then be opened in a Markdown viewer and it will be displayed according to the preferences set by the user before saving.

5 Usage Example

This section illustrates a usage scenario for OCRticle.

In this scenario, a user called Sam wants to convert an image with an article to text while keeping the article's structure, so Sam uses OCRticle for this purpose. Sam's original image can be seen in Figure 4.



Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce elementum consequat sapien, quis vehicula neque euismod eu. Sed et sapien in orci molestie elementum vitae in orci. Nunc consequat urna at finibus pellentesque. Nunc eget consectetur mi, non convallis odio. Curabitur semper enim id ipsum sagittis finibus. Aenean at bibendum diam, nec semper ipsum.

⚠ BLACK FRIDAY SALE: UP TO 30% OFF

Suspendisse tincidunt tempor erat. Ut aliquet auctor malesuada. Proin laoreet vulputate diam, ac mollis eros volutpat id. Quisque maximus nec est et egestas. Cras venenatis nulla pellentesque gravida feugiat. Nam iaculis sem nec luctus consectetur. Vivamus at felis vehicula, molestie nunc nec, auctor risus. Vivamus tempus sagittis finibus. Curabitur tincidunt neque sagittis, sollicitudin ex id, consequat dui.

> *"Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit..."*

Vestibulum sollicitudin eros eu pellentesque egestas. Suspendisse libero sem, suscipit semper sem ac, mattis pellentesque arcu. Donec vestibulum ultrices neque, non fringilla dui hendrerit ac. Duis pellentesque neque nec cursus faucibus. Aliquam ut hendrerit nisi, eget consectetur nunc. Phasellus imperdiet velit non diam cursus dignissim. Duis eu enim quis risus molestie elementum in eu quam. Praesent sit amet consequat erat, id sollicitudin neque.

Figure 4 Image that Sam wants to convert to structured text.

Sam starts by opening OCRticle and selecting the image from their computer. This particular image is stored as a PNG file, but Tesseract accepts any kind of image format.

Then, OCRticle asks Sam to select the article or articles from the image. Since this image only has one article, Sam can simply press "Submit" and OCRticle assumes that the entire image contains just one article. However, since the image also contains an advertisement, Sam uses the "Exclude from article" drawing mode to draw a red rectangle over the banner, thus excluding it from being scanned by Tesseract.

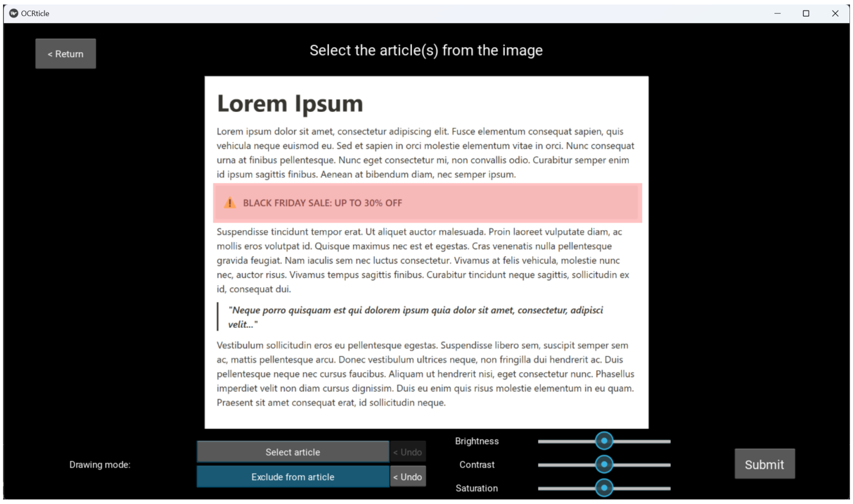
In addition, since the image has a purely white background and black text, Sam has no need to use the brightness, contrast or saturation sliders. However, these options would be useful if the image did not have a white background, or if it wasn't bright enough for the text to be easily read.

After pressing "Submit" on the article selection screen, Sam is taken to the next window, the article preview screen.

Here, the text from the image is divided into logical blocks, each representing a section of the original article. OCRticle correctly identified the article's title, but it didn't identify the quote block as being a quote, instead labeling it as "TEXT". Thus, Sam proceeds to click on the "TEXT" button besides the corresponding block, which opens a drop-down menu where they can select the "QUOTE" option, changing the block into a quote block.

The first two paragraphs of the text were split into different blocks because of the empty space in the image caused by the removal of the advertisement. Therefore, Sam presses the "Merge above" button on the second text block, in order to merge it with the first.

Sam also selects the "Keep line breaks" option, since they want the final text to match the original image as best as possible.



Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce elementum consequat sapien, quis vehicula neque euismod eu. Sed et sapien in orci molestie elementum vitae in orci. Nunc consequat urna at finibus pellentesque. Nunc eget consectetur mi, non convallis odio. Curabitur semper enim id ipsum sagittis finibus. Aenean at bibendum diam, nec semper ipsum.

⚠ BLACK FRIDAY SALE: UP TO 30% OFF

Suspendisse tincidunt tempor erat. Ut aliquet auctor malesuada. Proin laoreet vulputate diam, ac mollis eros volutpat id. Quisque maximus nec est et egestas. Cras venenatis nulla pellentesque gravida feugiat. Nam iaculis sem nec luctus consectetur. Vivamus at felis vehicula, molestie nunc nec, auctor risus. Vivamus tempus sagittis finibus. Curabitur tincidunt neque sagittis, sollicitudin ex id, consequat dui.

> ***"Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit..."***

Vestibulum sollicitudin eros eu pellentesque egestas. Suspendisse libero sem, suscipit semper sem ac, mattis pellentesque arcu. Donec vestibulum ultrices neque, non fringilla dui hendrerit ac. Duis pellentesque neque nec cursus faucibus. Aliquam ut hendrerit nisi, eget consectetur nunc. Phasellus imperdiet velit non diam cursus dignissim. Duis eu enim quis risus molestie elementum in eu quam. Praesent sit amet consequat erat, id sollicitudin neque.

Figure 5 Article selection screen. The red rectangle represents an "exclusion zone" that won't be considered by OCRticle.

Figure 7 shows the same screen as Figure 6, but now with Sam's changes applied to the article.

Finally, Sam presses the "Save article(s)" button, which takes them into the final screen of the application, where Sam is asked to save a file containing the text from the previous screen, correctly formatted. For this example, the generated file can be seen in Figure 8.

Both the article's title and the quote are correctly identified, according to Markdown syntax.

6 Case study

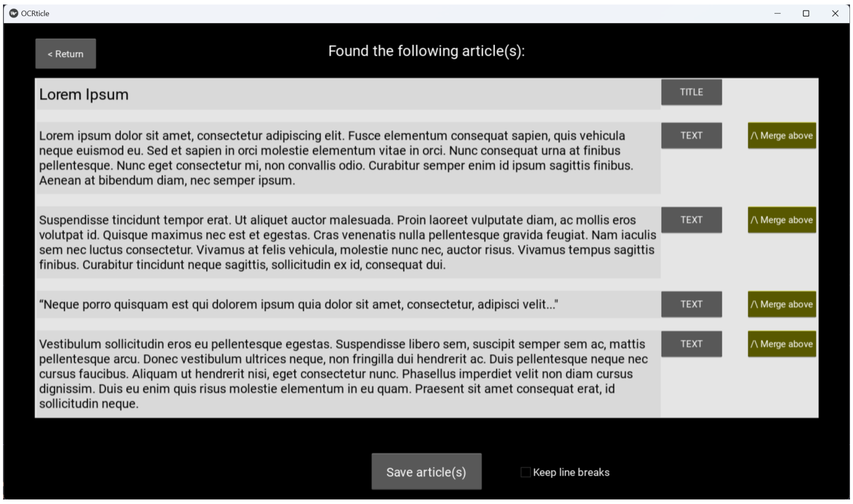
Throughout OCRticle's development, several tests were conducted with all kinds of images. In order to verify how OCRticle fared against real pictures, instead of just computer screenshots, we used pictures of newspaper clippings or pages. One of those tests, which was performed with a Portuguese newspaper from 1928, is described below.

Figure 9 shows the newspaper page in question, which was given to both Tesseract and OCRticle.

Due to the color of the paper and to the fact that Tesseract's Portuguese dictionary does not contain some of the old words used in these articles, the results from just using Tesseract for OCR are rather poor.

The full output was too large to be included in this paper, but this snippet, which corresponds to the top left article from Figure 9 shows clearly that plain Tesseract does a bad job at detecting text from this image:

```
Instalação da Comissão Administra-
realisada no dia à do corrente.

| Snr. Presidente da C. A. da
"pelo Snr. Governador Ci-
rmos do $ 3.º do art. 2.º
eto de 31 de Dezembro
"nomeado Administrador
ncelho, cargo que actual-
erce e assim nos termos
```



Found the following article(s):

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce elementum consequat sapien, quis vehicula neque euismod eu. Sed et sapien in orci molestie elementum vitae in orci. Nunc consequat urna at finibus pellentesque. Nunc eget consectetur mi, non convallis odio. Curabitur semper enim id ipsum sagittis finibus. Aenean at bibendum diam, nec semper ipsum.

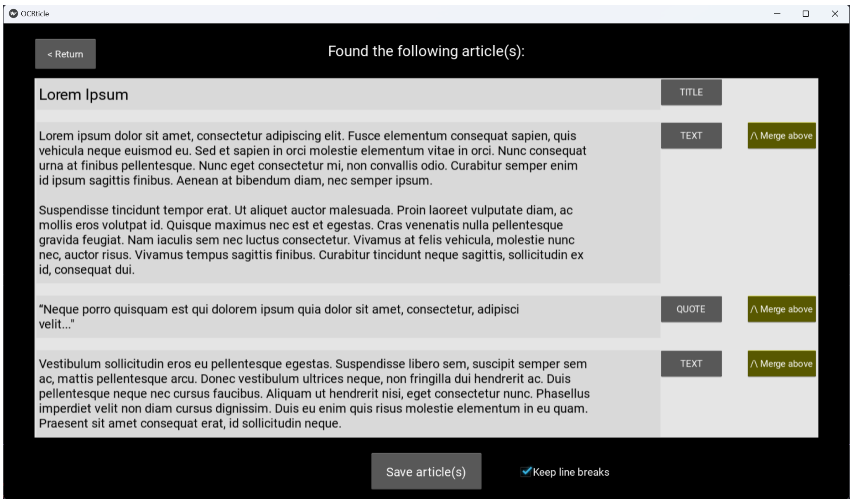
Suspendisse tincidunt tempor erat. Ut aliquet auctor malesuada. Proin laoreet vulputate diam, ac mollis eros volutpat id. Quisque maximus nec est et egestas. Cras venenatis nulla pellentesque gravida feugiat. Nam iaculis sem nec luctus consectetur. Vivamus at felis vehicula, molestie nunc nec, auctor risus. Vivamus tempus sagittis finibus. Curabitur tincidunt neque sagittis, sollicitudin ex id, consequat dui.

"Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit..."

Vestibulum sollicitudin eros eu pellentesque egestas. Suspendisse libero sem, suscipit semper sem ac, mattis pellentesque arcu. Donec vestibulum ultrices neque, non fringilla dui hendrerit ac. Duis pellentesque neque nec cursus faucibus. Aliquam ut hendrerit nisi, eget consectetur nunc. Phasellus imperdiet velit non diam cursus dignissim. Duis eu enim quis risus molestie elementum in eu quam. Praesent sit amet consequat erat, id sollicitudin neque.

☐ Keep line breaks

Figure 6 Article preview screen.



Found the following article(s):

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce elementum consequat sapien, quis vehicula neque euismod eu. Sed et sapien in orci molestie elementum vitae in orci. Nunc consequat urna at finibus pellentesque. Nunc eget consectetur mi, non convallis odio. Curabitur semper enim id ipsum sagittis finibus. Aenean at bibendum diam, nec semper ipsum.

Suspendisse tincidunt tempor erat. Ut aliquet auctor malesuada. Proin laoreet vulputate diam, ac mollis eros volutpat id. Quisque maximus nec est et egestas. Cras venenatis nulla pellentesque gravida feugiat. Nam iaculis sem nec luctus consectetur. Vivamus at felis vehicula, molestie nunc nec, auctor risus. Vivamus tempus sagittis finibus. Curabitur tincidunt neque sagittis, sollicitudin ex id, consequat dui.

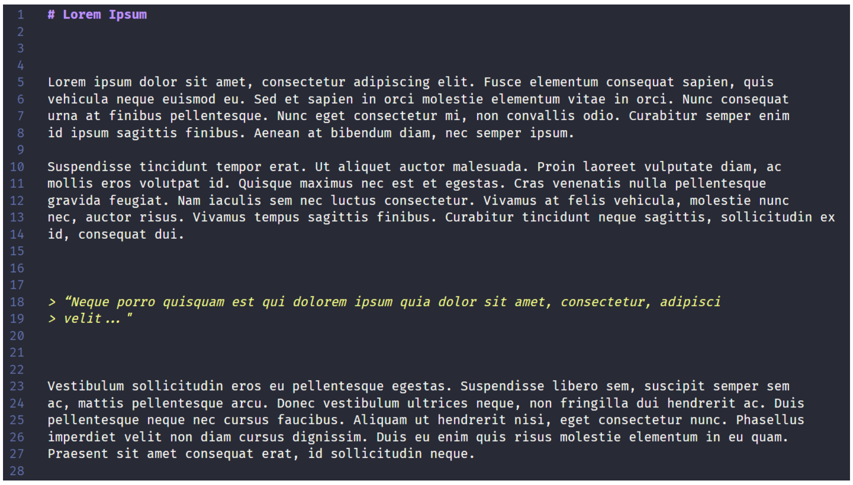
"Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit..."

Vestibulum sollicitudin eros eu pellentesque egestas. Suspendisse libero sem, suscipit semper sem ac, mattis pellentesque arcu. Donec vestibulum ultrices neque, non fringilla dui hendrerit ac. Duis pellentesque neque nec cursus faucibus. Aliquam ut hendrerit nisi, eget consectetur nunc. Phasellus imperdiet velit non diam cursus dignissim. Duis eu enim quis risus molestie elementum in eu quam. Praesent sit amet consequat erat, id sollicitudin neque.

☑ Keep line breaks

Figure 7 Article preview screen after Sam's changes.

```
1 presidiu a esta sessão.

Designou o dia para as ses-
uartas feiras pelas 15

gusto Barreira.

expostos; Dr.

jaldio:
ndes

juzir esta secção.

o - Mesa da Camara:

te Dr. Gonçalo Monteiro
; Vice-presidente Dr.
aquim Machado Guima-
```



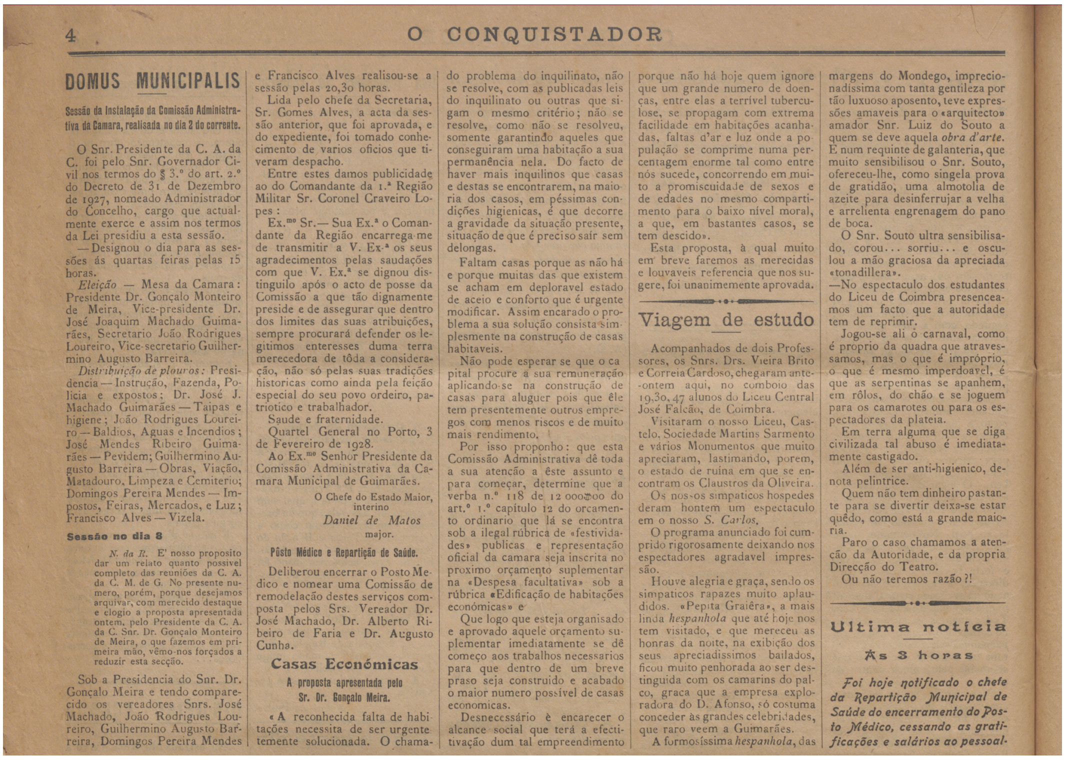
```
# Lorem Ipsum



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce elementum consequat sapien, quis
vehicula neque euismod eu. Sed et sapien in orci molestie elementum vitae in orci. Nunc consequat
urna at finibus pellentesque. Nunc eget consectetur mi, non convallis odio. Curabitur semper enim
id ipsum sagittis finibus. Aenean at bibendum diam, nec semper ipsum.

Suspendisse tincidunt tempor erat. Ut aliquet auctor malesuada. Proin laoreet vulputate diam, ac
mollis eros volutpat id. Quisque maximus nec est et egestas. Cras venenatis nulla pellentesque
gravida feugiat. Nam iaculis sem nec luctus consectetur. Vivamus at felis vehicula, molestie nunc
nec, auctor risus. Vivamus tempus sagittis finibus. Curabitur tincidunt neque sagittis, sollicitudin ex
id, consequat dui.



> “Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci
> velit … ”



Vestibulum sollicitudin eros eu pellentesque egestas. Suspendisse libero sem, suscipit semper sem
ac, mattis pellentesque arcu. Donec vestibulum ultrices neque, non fringilla dui hendrerit ac. Duis
pellentesque neque nec cursus faucibus. Aliquam ut hendrerit nisi, eget consectetur nunc. Phasellus
imperdiet velit non diam cursus dignissim. Duis eu enim quis risus molestie elementum in eu quam.
Praesent sit amet consequat erat, id sollicitudin neque.

```

Figure 8 File containing Sam's article.

```
Secretario Juão Rodrigues.
ice-secretario Guilher-

ibuição de plouros: Presi-
Instrução, Fazenda, Po-
José J.
ado Guimarães - Taipas e
e; João Rodrigues Lourei-
Ss, Aguas e Incendios ;
Ribeiro Guima-
Pevidem; Guilhermino Au-
Barreira - Obras, Viação,
ro, Limpeza e Cemiterio;
os Pereira Mendes - Im-
Feiras, Mercados, e Luz;
o Alves - Vizela.
```

On the other hand, with OCRticle, text detection improves a fair amount, substantially in some cases. The last two articles from the original image were detected and processed by OCRticle as follows (consecutive blank lines have been suppressed in order to preserve space):

```
rto, 3

ate da a Cates.

Maior,

tos

tie.

to Meão de 5 comr Dr. o Rigusto

as

gos com mais renc Por is: Comissão a sua ati para con verba n.º AE to ordina sob a ileg
des» put oficial da proximo | na «Desp rubrica «] económic:

Que los e aprovac plementar começo a:

# Viagem de estudo
```



4 O CONQUISTADOR

DOMUS MUNICIPALIS

Sessão da Instalação da Comissão Administrativa da Camara, realisada no dia 2 do corrente.

O Snr. Presidente da C. A. da C. foi pelo Snr. Governador Civil nos termos do § 3.º do art. 2.º do Decreto de 31 de Dezembro de 1927, nomeado Administrador do Concelho, cargo que actualmente exerce e assim nos termos da Lei presidiu a esta sessão.

— Designou o dia para as sessões ás quartas feiras pelas 15 horas.

Eleição — Mesa da Camara: Presidente Dr. Gonçalo Monteiro de Meira, Vice-presidente Dr. José Joaquim Machado Guimarães, Secretario João Rodrigues Loureiro, Vice-secretario Guilhermino Augusto Barreira.

Distribuição de plouros: Presidencia — Instrução, Fazenda, Policia e expostos; Dr. José J. Machado Guimarães — Taipas e higiene; João Rodrigues Loureiro — Baldios, Aguas e Incendios; José Mendes Ribeiro Guimarães — Pevidem; Guilhermino Augusto Barreira — Obras, Viação, Matadouro, Limpeza e Cemiterio; Domingos Pereira Mendes — Impostos, Feiras, Mercados, e Luz; Francisco Alves — Vizela.

Sessão no dia 8

N. da R. E' nosso proposito dar um relato quanto possivel completo das reuniões da C. A. da C. M. de G. No presente numero, porém, porque desejamos arquivar, com merecido destaque e elogio a proposta apresentada ontem, pelo Presidente da C. A. da C. Snr. Dr. Gonçalo Monteiro de Meira, o que fazemos em primeira mão, vêmo-nos forçados a reduzir esta secção.

Sob a Presidencia do Snr. Dr. Gonçalo Meira e tendo comparecido os vereadores Snrs. José Machado, João Rodrigues Loureiro, Guilhermino Augusto Barreira, Domingos Pereira Mendes e Francisco Alves realisou-se a sessão pelas 20,30 horas.

Lida pelo chefe da Secretaria, Sr. Gomes Alves, a acta da sessão anterior, que foi aprovada, e do expediente, foi tomado conhecimento de varios oficios que tiveram despacho.

Entre estes damos publicidade ao do Comandante da 1.ª Região Militar Sr. Coronel Craveiro Lopes:

Exmo Sr. — Sua Ex.ª o Comandante da Região encarrega-me de transmitir a V. Ex.ª os seus agradecimentos pelas saudações com que V. Ex.ª se dignou distinguilo após o acto de posse da Comissão a que tão dignamente preside e de assegurar que dentro dos limites das suas atribuições, sempre procurará defender os legitimos enteresses duma terra merecedora de tôda a consideração, não só pelas suas tradições historicas como ainda pela feição especial do seu povo ordeiro, patriotico e trabalhador.

Saude e fraternidade.

Quartel General no Porto, 3 de Fevereiro de 1928.

Ao Exmo Senhor Presidente da Comissão Administrativa da Camara Municipal de Guimarães.

O Chefe do Estado Maior,
interino
Daniel de Matos
major.

Pôsto Médico e Repartição de Saúde.

Deliberou encerrar o Posto Medico e nomear uma Comissão de remodelação destes serviços composta pelos Srs. Vereador Dr. José Machado, Dr. Alberto Ribeiro de Faria e Dr. Augusto Cunha.

Casas Económicas

A proposta apresentada pelo Sr. Dr. Gonçalo Meira.

«A reconhecida falta de habitações necessita de ser urgentetemente solucionada. O chamado problema do inquilinato, não se resolve, com as publicadas leis do inquilinato ou outras que sigam o mesmo critério; não se resolve, como não se resolveu, somente garantindo aqueles que conseguiram uma habitação a sua permanência nela. Do facto de haver mais inquilinos que casas e destas se encontrarem, na maioria dos casos, em péssimas condições higienicas, é que decorre a gravidade da situação presente, situação de que é preciso saír sem delongas.

Faltam casas porque as não há e porque muitas das que existem se acham em deploravel estado de aceio e conforto que é urgente modificar. Assim encarado o problema a sua solução consista simplesmente na construção de casas habitaveis.

Não pode esperar se que o capital procure a sua remuneração aplicando-se na construção de casas para aluguer pois que êle tem presentemente outros empregos com menos riscos e de muito mais rendimento.

Por isso proponho: que esta Comissão Administrativa dê toda a sua atenção a êste assunto e para começar, determine que a verba n.º 118 de 12 000\$00 do art.º 1.º capítulo 12 do orcamento ordinario que lá se encontra sob a ilegal rúbrica de «festividades» publicas e representação oficial da camara seja inscrita no proximo orçamento suplementar na «Despesa facultativa» sob a rúbrica «Edificação de habitações económicas» e

Que logo que esteja organisado e aprovado aquele orçamento suplementar imediatamente se dê começo aos trabalhos necessarios para que dentro de um breve praso seja construido e acabado o maior numero possível de casas economicas.

Desnecessário è encarecer o alcance social que terá a efectitivação dum tal empreendimento porque não há hoje quem ignore que um grande numero de doenças, entre elas a terrível tuberculose, se propagam com extrema facilidade em habitações acanhadas, faltas d'ar e luz onde a população se comprime numa percentagem enorme tal como entre nós sucede, concorrendo em muito a promiscuidade de sexos e de edades no mesmo compartimento para o baixo nível moral, a que, em bastantes casos, se tem descido».

Esta proposta, à qual muito em breve faremos as merecidas e louvaveis referencia que nos sugere, foi unanimemente aprovada.

Viagem de estudo

Acompanhados de dois Professores, os Snrs. Drs. Vieira Brito e Correia Cardoso, chegaram ante-ontem aqui, no comboio das 19,30, 47 alunos do Liceu Central José Falcão, de Coimbra.

Visitaram o nosso Liceu, Castelo, Sociedade Martins Sarmento e vários Monumentos que muito apreciaram, lastimando, porem, o estado de ruina em que se encontram os Claustros da Oliveira.

Os nossos simpaticos hospedes deram hontem um espectaculo em o nosso *S. Carlos*.

O programa anunciado foi cumprido rigorosamente deixando nos espectadores agradavel impressão.

Houve alegria e graça, sendo os simpaticos rapazes muito aplaudidos. «Pepita Graiêra», a mais linda *hespanhola* que até hoje nos tem visitado, e que mereceu as honras da noite, na exibição dos seus apreciadissimos bailados, ficou muito penhorada ao ser destinguida com os camarins do palco, graca que a empresa exploradora do D. Afonso, só costuma conceder às grandes celebridades, que raro veem a Guimarães.

A formosíssima *hespanhola*, das margens do Mondego, imprecionadíssima com tanta gentileza por tão luxuoso aposento, teve expressões amaveis para o «arquitecto» amador Snr. Luiz do Souto a quem se deve aquela *obra d'arte*. E num requinte de galanteria, que muito sensibilisou o Snr. Souto, ofereceu-lhe, como singela prova de gratidão, uma almotolia de azeite para desinferrujar a velha e arrelienta engrenagem do pano de boca.

O Snr. Souto ultra sensibilisado, corou... sorriu... e osculou a mão graciosa da apreciada «tonadillera».

—No espectaculo dos estudantes do Liceu de Coimbra presenceamos um facto que a autoridade tem de reprimir.

Jogou-se ali o carnaval, como é proprio da quadra que atravessamos, mas o que é impróprio, o que é mesmo imperdoavel, é que as serpentinas se apanhem, em rôlos, do chão e se joguem para os camarotes ou para os espectadores da plateia.

Em terra alguma que se diga civilizada tal abuso é imediatamente castigado.

Além de ser anti-higienico, denota pelintrice.

Quem não tem dinheiro pastante para se divertir deixa-se estar quêdo, como está a grande maioria.

Paro o caso chamamos a atenção da Autoridade, e da propria Direcção do Teatro.

Ou não teremos razão?!

Ultima notícia

Ás 3 horas

Foi hoje notificado o chefe da Repartição Municipal de Saúde do encerramento do Posto Médico, cessando as gratificações e salários ao pessoal.

Figure 9 Image of a 1928 Portuguese newspaper used for testing.

```
Acompanhados de dois Professores, os Snrs. Drs. Vieira Brito e Correia Cardoso, chegaram
 ante- -ontem aqui, no comboio das 19,30, 47 alunos do Liceu Central José Falcão, de
Coimbra.

Visitaram o nosso Liceu, Castelo, Sociedade Martins Sarmento e vários Monumentos que
muito apreciaram, lastimando, porem, o estado de ruina em que se encontram os Claustros
da Oliveira.

Os nossos simpaticos hospedes deram hontem um espectaculo em o nosso S. Carlos,

O programa anunciado foi cumprido rigorosamente deixando nos espectadores agradavel
impressão.

Houve alegria e graça, sendo os simpaticos rapazes muito aplaudidos. «Pepita Graiêra», a
mais linda hespanhola que até hoje nos tem visitado, e que mereceu as honras da noite, na
exibição dos seus apreciadissimos  bailados, ficou muito penhorada ao ser destinguida com
os camarins do palco, graca que a empresa exploradora do D. Afonso, só costuma conceder
às grandes celebridades, que raro veem a Guimarães.

A formosíssima hespanhola, das

---

Ultima notícia

As 3 horas

Foi hoje notificado o chefe da Repartição Municipal de Saúde do encerramento do Posto
Médico, cessando as gratificações e salários ao pessoal:
```

On the last article, OCRticle was able to maintain 100% of the original text and structure. However, the other article has many problems on the first half. This is mostly due to the fact that this article's title, on the original image, is below some of the text, which causes Tesseract to become confused.

Figure 10 shows how the articles were selected in OCRticle. Moreover, the image was converted to black-and-white and its brightness and contrast were increased using OCRticle's sliders, which also improved text detection.

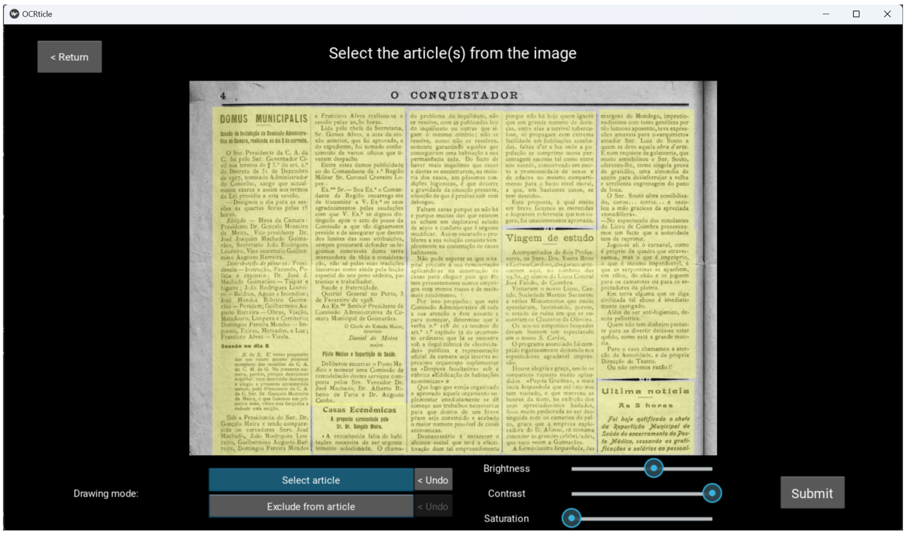


Figure 10 Article selection screen for the 1928 newspaper image.

The issue with the penultimate article can be alleviated by selecting the two columns as separate articles (i.e., as non-intersecting rectangles) in OCRticle. In Figure 10, since the two rectangles over the article are intersecting, OCRticle considers them to be part of the same article. When using two different rectangles, the following output is obtained:

```
# Viagem de estudo



Acompanhados de dois Professores, os Snrs. Drs. Vieira Brito e Correia Cardoso, chegaram
ante- -ontem aqui, no comboio das 19,30, 47 alunos do Liceu Central | José Falcão, de
Coimbra. -

Visitaram o nosso Liceu, Cas- | telo, Sociedade Martins Sarmento e vários Monumentos que
 muito apreciaram, lastimando, porem, o estado de ruina em que se encontram os Claustros
 da Oliveira.

Os nossos simpaticos hospedes deram hontem um espectaculo em o nosso S. Carlos,

O programa anunciado foi cumprido rigorosamente deixando nos espectadores agradavel
impressão.

Houve alegria e graça, sendo os simpaticos rapazes muito aplaudidos. «Pepita Graiêra», a
mais linda hespanhola que até hoje nos tem visitado, e que mereceu as honras da noite, na
exibição dos seus apreciadissimos  bailados, ficou muito penhorada ao ser destinguida com
os camarins do palco, graca que a empresa exploradora do D. Afonso, só costuma conceder
às grandes celebridades, que raro veem a Guimarães.

A formosíssima hespanhola, das
```

```
---

margens do Mondego, imprecionadíssima com tanta gentileza por tão luxuoso aposento, teve
expressões amaveis para o «arquitecto» amador Snr. Luiz do Souto a quem se deve aquela
obra d'arte. E num requinte de galanteria, que muito sensibilisou o Snr. Souto,
ofereceu-lhe, como singela prova de gratidão, uma almotolia de azeite para desinferrujar
a velha e arrelienta engrenagem do pano de boca.

O Snr. Souto ultra sensibilisado, corou... sorriu... e osculou a mão graciosa da
apreciada «tonadillera». -No espectaculo dos estudantes do Liceu de Coimbra presenceamos
um facto que a autoridade tem de reprimir.

Jogou-se ali o carnaval, como é proprio da quadra que atravessamos, mas o que é impróprio,
o que é mesmo imperdoavel, é que as serpentinas se apanhem, em rôlos, do chão e se joguem
para os camarotes ou para os espectadores da plateia.

Em terra alguma que se diga civilizada tal abuso é imediatamente castigado.

Além de ser anti-higienico, denota pelintrice.

Quem não tem dinheiro pastante para se divertir deixa-se estar quêdo, como está a grande
maioria.

Paro o caso chamamos a atencão da Autoridade, e da propria Direcção do Teatro.

Ou não teremos razão?!

---
```

While not a perfect solution, it substantially improves text detection. One would just need to delete the three dashes between both text blocks (they are introduced by OCRticle to separate distinct articles within the same file) in order to manually merge both articles.

The only remaining issue with OCRticle's output is the occasional appearance of vertical bars, colons or hyphens which don't appear in the original text. This happens because the paper on the original image has some imperfections, like folds or signs of age, which confuse Tesseract. While converting the image to black-and-white fixes some of these issues, it's impossible to fully remove these imperfections without manually editing the image. Another method to fix these errors would be to develop an algorithm to detect these "intruders" and remove them. However, this introduces a new problem, i.e., how do we detect if a vertical bar or other erroneous text element is supposed to be in the text or not. With OCRticle, since the main focus was text detection and not necessarily text correction, we followed a more conservative approach, and decided to let each individual user deal with these imperfections, instead of trying to automatically fix them. However, a future version of OCRticle could possess such a feature, even if optional or togglable.

7 Conclusion

This paper showcased OCRticle, a prototype software capable of performing Optical Character Recognition on articles while maintaining the article's original structure. OCRticle is able to successfully receive an image with its articles highlighted and extract them in textual format. It's able to automatically detect some types of text blocks, like titles, marking them as such in the generated Markdown file. The case studies shown in the previous section demonstrated that, although not perfect, OCRticle is a much more valuable solution than just using Tesseract when performing OCR on an image with one or more articles, even more so if one wishes to keep the text's original structure.

7.1 Release

OCRticle is publicly available for download and installation on PyPI, at `https://pypi.org/project/ocrticle/`, and on GitHub, at `https://github.com/RisingFisan/OCRticle`.

On Unix systems, it can be installed by first installing Tesseract and then running `pip install ocrticle`. On Windows systems, due to external dependencies, installation using `pip` may fail. Therefore, an executable file is available for download on the project's GitHub page. Tesseract installation is still required for Windows users.

References

1 What is ocr (optical character recognition)? - aws. URL: `https://aws.amazon.com/what-is/ocr/`.

2 Matt Cone. Markdown guide. URL: `https://www.markdownguide.org/`.

3 Freeocr. URL: `http://www.paperfile.net/`.

4 Search what you see. URL: `https://lens.google/`.

5 Trey Harris. Converting a scanned document into a compressed, searchable pdf with redactions, September 2022. URL: `https://medium.com/@treyharris/converting-a-scanned-document-into-a-compressed-searchable-pdf-with-redactions-63f61c34fe4c`.

6 Google answers whether it's better to ocr text in pdfs or not, August 2022. URL: `https://iloveseo.com/seo/google-answers-whether-its-better-to-ocr-text-in-pdfs-or-not/`.

7 An introduction to markup. URL: `https://port.sas.ac.uk/mod/book/view.php?id=568&chapterid=336`.

8 Kivy: Cross-platform python framework for gui apps development. URL: `https://kivy.org/`.

9 Kaan Kuguoglu. How to use image preprocessing to improve the accuracy of tesseract, July 2021. URL: `https://towardsdatascience.com/getting-started-with-tesseract-part-ii-f7f9a0899b3f`.

10 12+ best free ocr software for windows [2022 updated list], September 2022. URL: `https://www.softwaretestinghelp.com/ocr-software-for-pc/`.

11 Ocrspace. URL: `https://ocr.space/`.

12 Image to text converter using ocr online. URL: `https://www.onlineocr.net/`.

13 Pytesseract. URL: `https://pypi.org/project/pytesseract/`.

14 Online ocr - free ocr pdf document scanner & converter. URL: `https://www.sodapdf.com/ocr-pdf/`.

15 Tesseract user manual. URL: `https://tesseract-ocr.github.io/tessdoc/`.

16 Improving the quality of the output - tesseract documentation. URL: `https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html`.

17 Tesseract.js: Pure javascript ocr for 100 languages! URL: `https://tesseract.projectnaptha.com/`.

18 Languages supported in different versions of tesseract. URL: `https://tesseract-ocr.github.io/tessdoc/Data-Files-in-different-versions.html`.

19 TheJoeFin. Thejoefin/text-grab: Use ocr in windows 10 quickly and easily with text grab. with optional background process and popups. URL: `https://github.com/TheJoeFin/Text-Grab`.

20 James Vincent. Google lens can now copy and paste handwritten notes to your computer, May 2020. URL: `https://www.theverge.com/2020/5/7/21250556/google-lens-copy-paste-handwritten-notes-computer-phone-ios-android`.

Narrative Extraction from Semantic Graphs

Daniil Lystopadskyi
Faculty of Sciences, University of Porto, Portugal

André Santos
CRACS & INESC TEC, Porto, Portugal
Faculty of Sciences, University of Porto, Portugal

José Paulo Leal
CRACS & INESC TEC, Porto, Portugal
Faculty of Sciences, University of Porto, Portugal

Abstract

This paper proposes an interactive approach for narrative extraction from semantic graphs. The proposed approach extracts events from RDF triples, maps them to their corresponding attributes, and assembles them into a chronological sequence to form narrative graphs. The approach is evaluated on the Wikidata graph and achieves promising results in terms of narrative quality and coherence. The paper also discusses several avenues for future work, including the integration of machine learning, graph embedding methods and the exploration of advanced techniques for attention-based narrative labeling and semantic role labeling. Overall, the proposed method offers a promising approach to narrative extraction from semantic graphs and has the potential to be useful in various applications, including chatbots, conversational agents, and content creation tools.

2012 ACM Subject Classification Information systems → Environment-specific retrieval; Information systems → Information extraction

Keywords and phrases Narratives, Narrative Extraction, Information Retrieval, Knowledge Graphs, Semantic Graphs, Resource Description Framework, Web Ontology

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.9

Category Short Paper

Funding This work is financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020.
André Santos: Ph. D. Grant SFRH/BD/129225/2017 from Fundação para a Ciência e Tecnologia (FCT), Portugal.

1 Introduction

Narratives are ubiquitous in human communication and understanding them is essential for various applications[8], including information retrieval, summarization, storytelling, question answering and content generation. However, defining narratives and formalizing them for computational processing is a challenging task.

Narrative extraction is the task of automatically identifying, analyzing, and representing narratives from textual or multimedia data[10]. Semantic graphs, which represent entities and their relationships in a structured and rich manner, offer a promising framework for narrative extraction that can capture both local and global coherence in a text[2].

The study of narratives has attracted significant attention in different research fields, including computer science[8]. Most of the work in this field involves extracting narratives from plain text using Natural Language Processing (NLP) techniques[10] or predefined event-centric graphs[12]. However, extracting narratives from semantic graphs is a relatively unexplored area, which is the focus of our work.

© Daniil Lystopadskyi, André Santos, and José Paulo Leal;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 9; pp. 9:1–9:8
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Semantic graphs provide a structured source of information that is easier for computers to understand. By defining narratives as graphs, we provide a universal representation that simplifies accessing and transferring information between systems[9]. Despite the potential benefits of semantic graphs for narrative extraction, existing methods face several challenges, such as the complexity of the graph structure, the sparsity of data and the lack of sufficient domain-specific knowledge[7].

The main contributions of this paper are twofold. First, we propose an interactive approach for narrative graph extraction from semantic graphs, which combines string matching and rule-based methods to capture the semantic and structural information of a narrative. Second, we demonstrate the effectiveness and adaptability of our approach across different domains and languages. In the following sections, we give background for common concepts in this field, provide an overview of related work, describe our approach in detail, present experimental results, and discuss future work and conclusions.

2 Background

This section provides an overview of general concepts that are relevant to fully comprehend the context of this field of study.

Semantic graphs, sometimes also referred to as knowledge graphs, store domain/context specific information about concepts and relations between them. The information stored in semantic graphs can also be viewed as a set of triples, where each triple corresponds to (Subject, Predicate, Object). In terms of graphs, these triples are represented as (Node, Edge, Node). Those triples are also called RDF triples.

Resource Description Framework (RDF) is a standard description format used for describing and exchanging metadata and other resources on the web. It forms an important part of the semantic web stack and plays a key role in enabling the interoperability and integration of data across different systems and domains.

The information within semantic graphs is structured according to a web ontology. An ontology, in web semantics context, is a standardized way of defining the hierarchy of concepts and relationships that exist within a particular domain, using a set of classes, properties and constraints.

SPARQL is a query language used to retrieve and manipulate data stored in RDF format. SPARQL allows users to query RDF data by specifying patterns and conditions that the data must match. These patterns can include information about the structure of the data, the types of entities and relationships involved, and constraints on the values of properties.

3 Related Work

Narrative extraction is a challenging task that has attracted significant research interest in recent years. Existing methods can be broadly classified into three categories: NLP methods, rule-based methods and hybrid methods.

NLP-based approaches that have been prominent in the field of narrative extraction can be summarized into five stages: Pre-Processing and Parsing; Identification and Extraction of Narrative Components; Linking Components; Representation of Narratives and Evaluation[10]. An example of this approach is narrative extraction from administrative records[4].

Rule-based methods for narrative extraction rely on manually crafted rules or heuristics to identify the narrative structure or content. These methods often require domain-specific knowledge and are limited in their adaptability to new domains or languages. Those types of approaches are sparse, the most recent one being [2], which uses pre-defined mappings between event components and properties in Wikidata to extract events.

Hybrid methods for narrative extraction combine NLP-based and rule-based approaches to leverage their respective strengths. For example, [5] proposed a method that uses both NLP-based and rule-based techniques to extract narrative events and their temporal relations from Wikipedia biographies.

4 Approach

In relation to the existing work in this specific field, our contribution consists of expanding the concept of events by removing constraints such as narrative genres and event classes, allowing for more flexible narratives, as well as providing a web interface for deeper levels of interactability and, consequently, higher generalizability. This approach can be described by the pipeline in Figure 1. The next subsections describe, in detail, each step of the pipeline.

4.1 Narrative Definition and Ontology Specification

In this paper, narratives will be defined as ordered sequences of events. Although there is a significant body of research focused on conceptualizing and defining narratives, this simplified definition is sufficient for the purposes of this study. For the purposes of this study, an event is considered an occurrence linked to a specific point in time. The 5W1H method, suggested by [2], which involves answering questions related to “What”, “Where”, “Who”, “When”, “Why” and “How”, will be used to describe events, with the “How” and “Why” questions excluded due to their complexity, leaving only 4W’s: “What”, “Where”, “Who” and “When”.

To ensure consistency in the description of narratives, a web ontology was created. The ontology has two primary classes: one to define events and the other to define narratives. The event class includes four properties, one for each of the 4W categories mentioned above, as well as a property to establish relationships between events and other entities that do not fit into any of the attributes, although still fundamental for narrative flow. The narrative class, as of time of writing this paper, has one property that links the narrative to its events. However, the properties for the narrative class are still under development and will be determined in due course. An example of an event in this ontology depicting the “Word War I Ottoman Southern Front” can be seen in Figure 2.

4.2 Narrative and Graph Configuration

One of the primary challenges that we face in applying the rule-based approaches universally is the inconsistency in graph structures across different domains[5]. This inconsistency affects how the graph is traversed and how information is retrieved, making it difficult to find a common ground between different graphs. Other inconsistencies, such as variations in schema and uri prefixes for entities and predicates, add further complexity to the problem. Furthermore, it is challenging to infer the meaning of ontology classes and predicates without a clear understanding of the ontology itself.

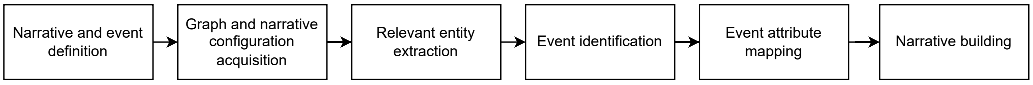

Narrative and event definition
Graph and narrative configuration acquisition
Relevant entity extraction
Event identification
Event attribute mapping
Narrative building

Figure 1 Approach Pipeline.

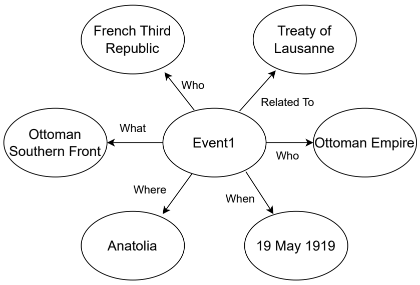


Figure 2 Ontology Event Representation.

Designing an algorithm that can work with any semantic graph, without prior knowledge of the specific graph, is a formidable task [6]. Our approach achieves adaptability by prompting users to provide specific configuration parameters for their graphs, such as the property that could replace “rdf:type”.

Graphs are not the only aspect that requires configuration. Narratives, being a subjective topic, require user input on how they should look and behave. Primarily, there needs to be a parameter that defines how “deep” a narrative should be, seeing as narratives can span out infinitely due to event inter-connectivity. This is also commonly known as the “butterfly effect”, which the parameter “depth” attempts to control.

Some graphs are too large for the algorithm to handle, such as Wikidata. Thus, it is necessary to specify a parameter that controls how much information is retrieved from the graph in order to avoid long execution times or even timeouts. We will refer to this parameter as the “size”.

4.3 Relevant Entity Extraction

The first step of this algorithm involves identifying the entity that represents the input topic. The topic is given by the user as a string. The algorithm queries the graph for this string and returns all entities that are labeled as such, of which there can be more than one. The user then needs to specify which entity they want the narrative for. This entity is then marked as the main entity, and event extraction can begin using it as the central point.

Once the main entity is acquired, the identification and extraction of all entities relevant to the narrative can commence. Relevant entities are the ones linked to the main entity through some property and constitute the narrative space.

The entities are collected through graph traversing using Breadth-first search. The depth of the search is set by the equivalently called parameter which defines how far should we search from the main entity. The higher the depth, the more specific or irrelevant events get in relation to the main entity.

4.4 Event Identification

To identify events within a semantic graph, our approach looks for timestamps associated with graph resources. Two types of events can be derived by manually analyzing semantic graphs and their contents: entity events and property events. Entity events are represented by entities that depict real-world events, such as “World War 2” or “1952 Swiss Mount Everest expedition”, while property events are represented by properties whose range is a literal with datatype “date” and/or “time”, such as “date of birth”. While it is true that there could be entities that define time instances, for the sake of simplicity, those are ignored.

Property events are extracted from all relevant entities that are not classified as events. The final set of events corresponds to extracted property events, as well as all relevant entities classified as events. This set is then used for attribute mapping for each event.

4.5 Attribute Mapping

The process of event attribute mapping requires a deep understanding of entity and predicate types. The assignment involves mapping all relevant data to each W of the 4W's (Who, What, Where, When) for every identified event, which is a challenging task. For this purpose, we ask user to, manually, assign one of the 4 classes ("Person/Group", "Location", "Event" or "Other") to all extracted entities and properties for further processing. As a result, this assignment can substantially influence the resulting narrative structure. The main advantage of this approach is high adaptability to different graphs, since it does not rely on hardcoded assignments.

For entity events, the attributes are mapped according to the following rules:

- **Who**: All entities linked to the event entity that are classified as "Person/Group".
- **What**: Label of the event entity, concatenated with the property label that contains the timestamp of the event, e.g. "start time".
- **Where**: All entities linked to the event entity that are classified as "Location".
- **When**: Value assigned to the property that contains the timestamp of the event.
- **Related to**: All entities linked to the event entity that are classified as "Event".

On the other hand, for event properties, the key difference is that not all objects of type "Person" and "Location" that belong to the same entity are relevant to the event. Also, we assume that all the necessary data is contained within the RDF triples of the same entity, meaning that all incoming links to this entity are ignored. The first step is to find all triples related to the triple containing the event property. This is done by clustering RDF triples of the same entity based on Ratcliff-Obershelp similarity between each triple's property labels and only match properties that show similarity scores greater or equal than a certain threshold, which, at this point of the development, has to be configured manually, through trial and error.

Once we have the set of all related triples, the attributes are mapped according to the following rules:

- **Who**: All objects from clustered triples that are classified as "Person/Group".
- **What**: Label of the event property.
- **Where**: All objects from clustered triples that are classified as "Location".
- **When**: Value assigned to the event property.
- **Related to**: All objects from clustered triples that are classified as "Event".

The summary of this process can be visualized in Figure 3 for entity events, and in Figure 4 for property events.

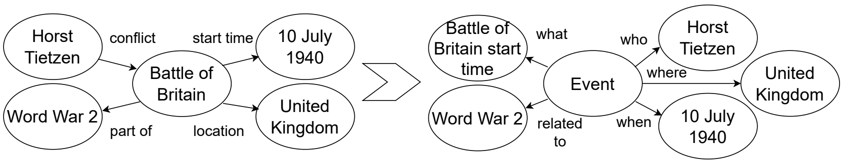


Figure 3 Entity Event Example.

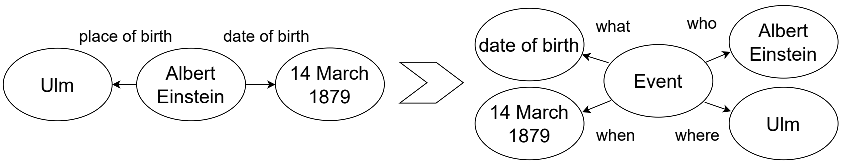


Figure 4 Property Event Example.

4.6 Narrative Building

After gathering all of the relevant data, the next step in the event extraction process involves assembling the narrative, which is, perhaps, both the simplest and the hardest step of the process. For the purpose of this paper and for the sake of clarity, we will be using a simple approach.

The narrative is essentially an ordered sequence of events that occurred within a certain period of time. As the extracted events all contain timestamps, building the narrative becomes a straightforward task of arranging the events in chronological order. This chronological order will serve as the backbone of the narrative, providing a clear timeline for the sequence of events. Since the goal is to build semantic graphs out of extracted narratives, each extracted event becomes a node in the graph. These events are linked to their attribute nodes, which can be either literals or other entities. A narrative is a node itself, linked to every event node that constitutes it. The final narrative graph can be embedded into original semantic graph.

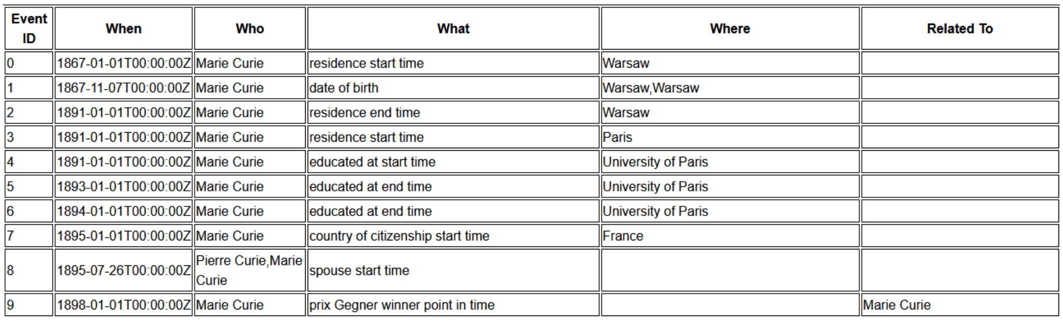
A more complex approach would involve incorporating other important concepts such as the perspective of the narrator, interconnected narratives and character roles. However, these more complex narrative-building options will be explored only after the primary method has been fully developed and refined.

5 Preliminary Results

To evaluate the quality of our approach, we conducted experiments on the Wikidata graph, a benchmark knowledge graph for information retrieval. The results were extracted in form of a table, with each row corresponding to an event and each column representing an event attribute. All results were obtained for parameters “depth” equals one and “size” equals thirty.

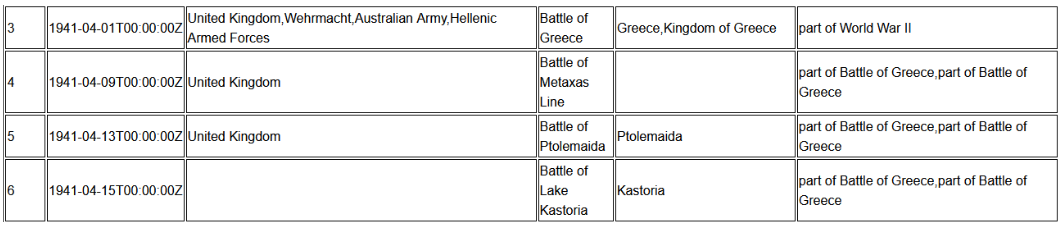
For the first example, “Marie Curie” was used as the topic for the algorithm. Once the correct type was selected for the topic, in this case a person, and all entity and property classes were assigned, the program returned results seen in Figure 5. In total, we managed to obtain forty-eight events, which had to be truncated to the first ten in order to fit in this paper. In those ten example events, we can deduct the general flow of the narrative, starting from birth and progressing towards marriage and winning an award. As can be seen in the table, some attributes are optional, since those were not found. Overall, the “What” attribute requires more inference from the part of the reader, since it is not always obvious what it is referring to. This is one of the possible quality-of-life adjustments that could be introduced.

Another topic used was “Battle of Greece”, a historical event, for which seven events were extracted and can be found in Figure 6. Once again, events that were too large were excluded due to space limitations. Just as in the first example, some items appear duplicated due to errors in the implementation of the algorithm, which can be resolved easily. In this short example, we can see the main entity, Battle of Greece, which is then linked to other



Event ID	When	Who	What	Where	Related To
0	1867-01-01T00:00:00Z	Marie Curie	residence start time	Warsaw	
1	1867-11-07T00:00:00Z	Marie Curie	date of birth	Warsaw,Warsaw	
2	1891-01-01T00:00:00Z	Marie Curie	residence end time	Warsaw	
3	1891-01-01T00:00:00Z	Marie Curie	residence start time	Paris	
4	1891-01-01T00:00:00Z	Marie Curie	educated at start time	University of Paris	
5	1893-01-01T00:00:00Z	Marie Curie	educated at end time	University of Paris	
6	1894-01-01T00:00:00Z	Marie Curie	educated at end time	University of Paris	
7	1895-01-01T00:00:00Z	Marie Curie	country of citizenship start time	France	
8	1895-07-26T00:00:00Z	Pierre Curie,Marie Curie	spouse start time		
9	1898-01-01T00:00:00Z	Marie Curie	prix Gegner winner point in time		Marie Curie

Figure 5 Section of Extracted Narrative for Marie Curie.



3	1941-04-01T00:00:00Z	United Kingdom,Wehrmacht,Australian Army,Hellenic Armed Forces	Battle of Greece	Greece,Kingdom of Greece	part of World War II
4	1941-04-09T00:00:00Z	United Kingdom	Battle of Metaxas Line		part of Battle of Greece,part of Battle of Greece
5	1941-04-13T00:00:00Z	United Kingdom	Battle of Ptolemaida	Ptolemaida	part of Battle of Greece,part of Battle of Greece
6	1941-04-15T00:00:00Z		Battle of Lake Kastoria	Kastoria	part of Battle of Greece,part of Battle of Greece

Figure 6 Section of Extracted Narrative for Battle of Greece.

events, such as World War 2, Battle of Metaxas Line, Battle of Ptolemaida, and so on... Once again, some attributes are omitted. The “Who” attributes in this case refers to participants of respective battles.

Overall, despite there being a lot of room for improvement, our preliminary results demonstrate the potential of our proposed approach for narrative extraction from semantic graphs. Seeing as this is a work in progress, a better validation method is required once the algorithm is finished to further evaluate the quality of produced narratives.

6 Conclusions

In this paper, we presented an interactive approach for extracting narratives from semantic graphs. Our approach utilizes matching methods, rule-based techniques and string comparison algorithms to analyse semantic graphs and extract narrative structures by converting them into their own semantic graphs. Our approach was evaluated on the Wikidata graph and showed promising results for a small sample size.

With that being said, there are several avenues for future work to further improve and extend our method. One direction is the integration of graph-based algorithms, such as graph embedding and graph neural networks[11], to enhance the model’s understanding of the narrative. Another direction is the development of machine learning models for property and entity classification, considering the vast amount of data provided by knowledge graphs[3]. Additionally, exploring more advanced techniques for attention-based narrative labeling and semantic role labeling can further improve the quality of the results[1]. Finally, it is important to evaluate our approach on other benchmark graphs and real-world datasets to assess its adaptability and practical usefulness.

In conclusion, our proposed approach for narrative extraction from semantic graphs represents a step forward in the field of event extraction and narrative understanding. We believe that our approach has the potential to be applied to a wide range of real-world applications, including chatbots, conversational agents, and automated story generation. We hope that our work will inspire future research in this exciting field and lead to further advancements in narrative extraction and semantic graph analysis.

References

1 Nandini Anantharama, Simon D. Angus, and Lachlan O'Neill. Canarex: Contextually aware narrative extraction for semantically rich text-as-data applications. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 3551–3564. Association for Computational Linguistics, 2022. URL: `https://aclanthology.org/2022.findings-emnlp.260`.

2 Inès Blin. Building narrative structures from knowledge graphs. In Paul Groth, Anisa Rula, Jodi Schneider, Ilaria Tiddi, Elena Simperl, Panos Alexopoulos, Rinke Hoekstra, Mehwish Alam, Anastasia Dimou, and Minna Tamper, editors, *The Semantic Web: ESWC 2022 Satellite Events - Hersonissos, Crete, Greece, May 29 - June 2, 2022, Proceedings*, volume 13384 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2022. `doi:10.1007/978-3-031-11609-4_38`.

3 Victor de Boer. Knowledge graphs for impactful data science (keynote). In Umutcan Simsek, David Chaves-Fraga, Tassilo Pellegrini, and Sahar Vahdat, editors, *Proceedings of Poster and Demo Track and Workshop Track of the 18th International Conference on Semantic Systems co-located with 18th International Conference on Semantic Systems (SEMANTiCS 2022), Vienna, Austria, September 13th to 15th, 2022*, volume 3235 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022. URL: `https://ceur-ws.org/Vol-3235/keynote1.pdf`.

4 Karine Megerdoomian, Karl Branting, Charles Horowitz, Amy Marsh, Stacy Petersen, and Eric Scott. Automated narrative extraction from administrative records. In Luther Karl Branting, editor, *Proceedings of the Workshop on Artificial Intelligence and the Administrative State co-located with 17th International Conference on AI and Law (ICAIL 2019), Montreal, QC, Canada, June 17, 2019*, volume 2471 of *CEUR Workshop Proceedings*, pages 38–48. CEUR-WS.org, 2019. URL: `https://ceur-ws.org/Vol-2471/paper7.pdf`.

5 Daniele Metilli. *Enhancing the Computational Representation of Narrative and Its Extraction from Text*. PhD thesis, University of Pisa, Italy, 2021. URL: `https://etd.adm.unipi.it/theses/available/etd-10222021-095519/`.

6 Thiloshon Nagarajah, Filip Ilievski, and Jay Pujara. Understanding narratives through dimensions of analogy. *CoRR*, abs/2206.07167, 2022. `doi:10.48550/arXiv.2206.07167`.

7 Emetis Niazmand, Gezim Sejdiu, Damien Graux, and Maria-Esther Vidal. Efficient semantic summary graphs for querying large knowledge graphs. *Int. J. Inf. Manag. Data Insights*, 2(1):100082, 2022. `doi:10.1016/j.jjimei.2022.100082`.

8 Priyanka Ranade, Sanorita Dey, Anupam Joshi, and Tim Finin. Computational understanding of narratives: A survey. *IEEE Access*, 10:101575–101594, 2022. `doi:10.1109/ACCESS.2022.3205314`.

9 Vetle Ryen, Ahmet Soylu, and Dumitru Roman. Building semantic knowledge graphs from (semi-)structured data: A review. *Future Internet*, 14(5):129, 2022. `doi:10.3390/fi14050129`.

10 Brenda Santana, Ricardo Campos, Evelin Amorim, Alípio Jorge, Purificação Silvano, and Sérgio Nunes. A survey on narrative extraction from textual data. *Artificial Intelligence Review*, January 2023. `doi:10.1007/s10462-022-10338-7`.

11 Daniil Sorokin. *Knowledge Graphs and Graph Neural Networks for Semantic Parsing*. PhD thesis, Technical University of Darmstadt, Germany, 2021. URL: `http://tuprints.ulb.tu-darmstadt.de/19187/`.

12 Zhihua Yan and Xijin Tang. Narrative graph: Telling evolving stories based on event-centric temporal knowledge graph. *Journal of Systems Science and Systems Engineering*, 32(2):206–221, April 2023. `doi:10.1007/s11518-023-5561-0`.

Large Language Models: Compilers for the 4th Generation of Programming Languages?

Francisco S. Marcondes
ALGORITMI Research Centre/LASI, University of Minho, Braga, Portugal
José João Almeida
ALGORITMI Research Centre/LASI, University of Minho, Braga, Portugal
Paulo Novais
ALGORITMI Research Centre/LASI, University of Minho, Braga, Portugal

Abstract

This paper explores the possibility of large language models as a fourth generation programming language compiler. This is based on the idea that large language models are able to translate a natural language specification into a program written in a particular programming language. In other words, just as high-level languages provided an additional language abstraction to assembly code, large language models can provide an additional language abstraction to high-level languages. This interpretation allows large language models to be thought of through the lens of compiler theory, leading to insightful conclusions.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Computing methodologies → Natural language processing; Software and its engineering → Compilers

Keywords and phrases programming language, compiler, large language model

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.10

Category Short Paper

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

As the title suggests, this is a speculative paper discussing whether large language models could be considered a higher level of programming language in relation to current high-level languages. In short, assembly language (2nd generation) replaced punch-card programming (1st generation) by introducing mnemonics. These allowed larger and more complex programs to be created in less time. High level languages (3rd generation) in turn replaced assembly language by introducing structured English constraints.

The hypothesis explored in this paper is that large language models could be a 4th generation language, replacing high-level languages by allowing natural language specifications. The aim is then to discuss the strengths and weaknesses of large language models running as *natural to high-level language* compilers. Other natural language processing tasks that would be associated with an interpreter are therefore beyond the scope of this paper. The question that arises is whether large language models can provide an additional level of abstraction for programming [10] similar to the high-level languages provided for assembly language.

As a disclaimer, this paper uses ChatGPT as a basis for discussion, but does not consider ChatGPT to be a compiler. A 4th generation compiler, as well as a 3rd generation compiler, is expected to produce executable code as a result, not intermediate code as ChatGPT does. Note that there are fine-tuned solutions for programming, such as the *GitHub co-pilot*, but due to its current paywall *ChatGPT 3.5* is used in this paper. Although ChatGPT is used to support this discussion, a proper setup should be designed for proper use of the large

© Francisco S. Marcondes, José João Almeida, and Paulo Novais;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 10; pp. 10:1–10:8
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

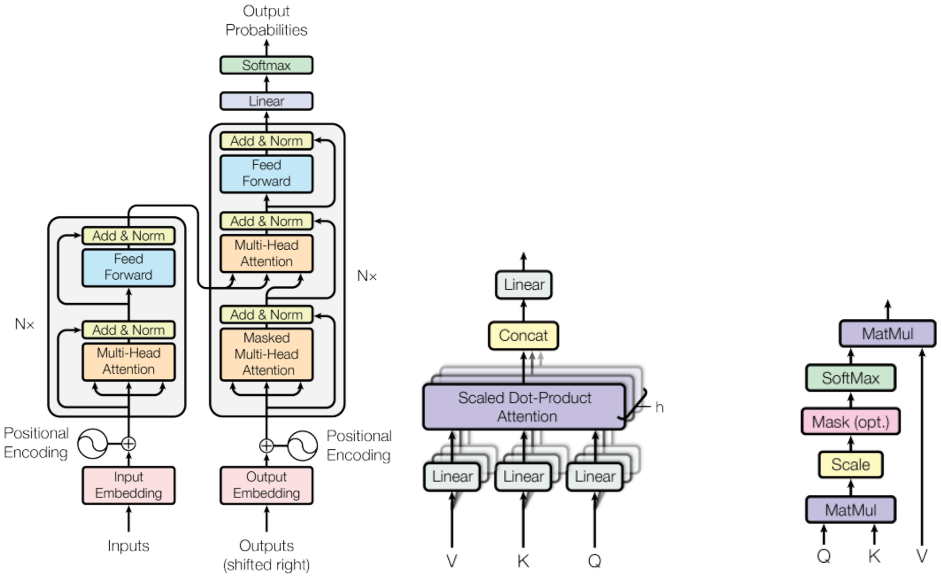
language model as a compiler. In this setup, a programmer is not expected to interfere with the lower-level code any more than a programmer is expected to modify the assembly code generated by a high-level language compiler.

Furthermore, a single prompt is not expected to be sufficient to produce industrial-scale software, nor is a single specification sufficient to achieve the same end. A typical requirement analysis takes several months and results in thousands of mutable requirements whose complexity is managed by interactive and incrementally throughout the project [14]. There is no reason to believe that this will change with the replacement of compiler technology. Therefore, industrial-scale software would require several mutable prompts in order to properly specify an industry level software product.

It is also worth noting that this proposal is not related to no-code or low-code initiatives. In short, these initiatives aim to provide a different way of doing traditional programming, but without providing the higher level language abstraction that large language models do. Despite several papers are addressing large language models for code synthesis, it was not possible to find any discussing it from the compiling perspective [16]. It was also not possible to find papers referring to 4th or next generation compilers linked with large language models; most are linked with model-driven [12] or domain-specific [17] approaches.

2 Theoretical Foundations

Current large language models are based on Transformers (see figure 1). Perhaps the core element in Transformers is the *attention mechanism* [5]. In brief, given an input embedding, the attention mechanism reweights the embeddings for each token in the input to match the context of the input sentence. For example, let `bank` be a token in the input sentence whose sense is equidistant from `river` and `money` in the embedding model. Multiplying the embedding values of `bank` by the embeddings of the other words in the input sentence is expected to bias the `bank` token towards the appropriate sense.



(a) Overall Architecture. **(b)** Multi-head attention. **(c)** Scaled dot-product attention.

Figure 1 Transformers *cf.* [13]. Note that (c) is in (b), which is in (a).

The attention mechanism is the underlying principle of *prompt engineering*. In a nutshell, prompt engineering is concerned with designing a prompt that, when queried by a large language model, returns the best possible answer. As a rule of thumb, a prompt *cf.* [9] consists of: a) instruction; b) context; c) input data; and d) output indicator. Prompt engineering is therefore less about acting out a natural language conversation, and more about describing specific instructions aimed at an output.

This leads to the almost straightforward conclusion that large language models can be understood as a natural *language processor* (see [1]). A compiler, or translator, is a type of language processor that converts sentences from a usually high-level source language to an often low-level target language. The compiled sentences are interpretable by the target machine, which behaves accordingly. Considering that Transformers was built with the goal of natural language translation in mind [13], a relationship between Transformers and compilers can be suggested, with the source language being a natural language and the target language being a high-level language.

3 Proof of Concept

For reference, consider the introductory programming class problem:

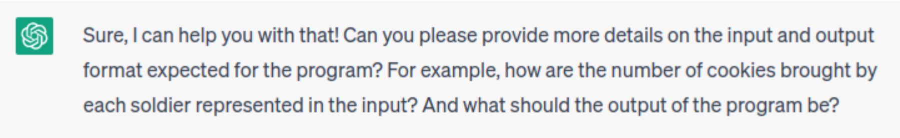
> *The soldiers of the queen of hearts have a problem: once again the queen has sent them to fetch cookies for tea. Five of the soldiers went to get the cookies and returned. Since only one of the soldiers can enter the tea room with the cookies, they need to choose one of them. The problem is that the queen is greedy and has very little patience. Either they quickly figure out which soldier brought the most cookies or they will lose their heads. Your task is to write a program to find the answer.* (1)

The above problem has been prompted in ChatGPT as is. The result is shown in the figure 2a. Recall that a prompt *cf.* [9] is composed of: a) instruction; b) context; c) input data; and d) output indicator. These are the elements being required by the model. ChatGPT does not always give the same output for the same prompt, this is due to the Q, K, V (stands for query, key and value) weight matrices used by the attention mechanism (see figure 1). On a second run of this prompt in a different ChatGPT instance, the language model adopted the previously requested parameters and returned the source code shown in 2b. Therefore, in order to reduce variation, it is necessary to be as specific as possible on prompt building.

Note that the code in figure 2b is well-structured Python source code, capable of running on a Python machine. An immediate assumption is that by providing a formal specification, the resulting code would be enhanced. Prompt (1) is then rewritten as prompt (2) using simple set notation and submitted to ChatGPT, resulting in the code shown in figure 3. Note that the code in figure 3 is not necessarily better or more readable than the code in figure 2b. Such an assumption is therefore not necessarily true in the domain of large language models.

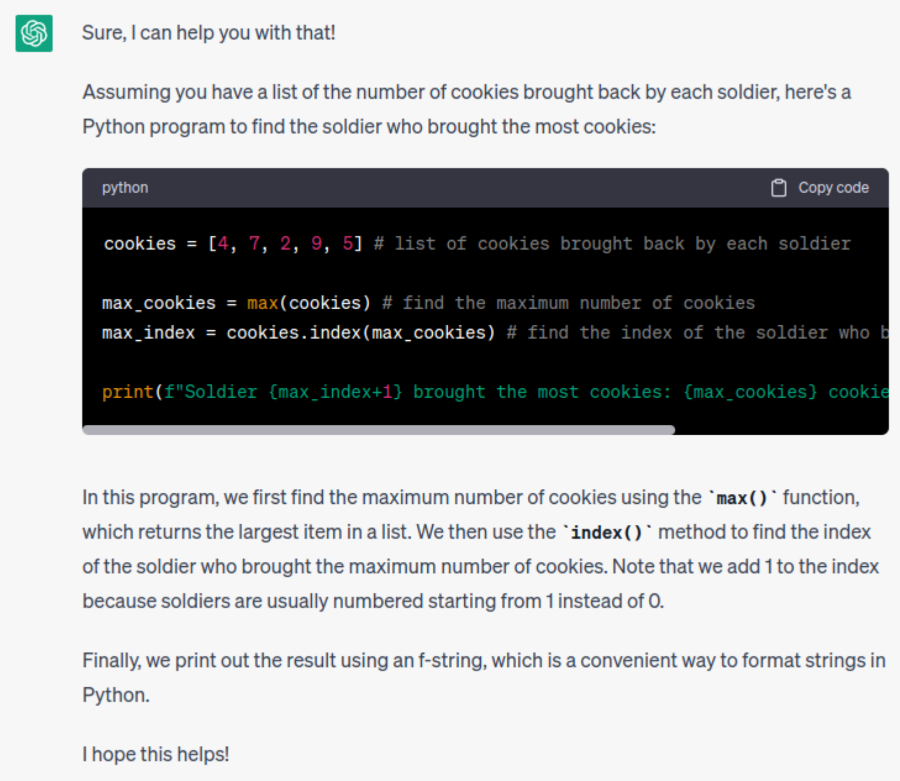
> *Let* $S = \{s_1, ..., s_n | n = 5\}$ *be a set of soldiers,* $C = \{c_1, ..., c_n | n = 5\}$ *be the number of cookies brought by each soldier, and* $f : S \to C$ *is a function that returns the number of cookies of each soldier. The soldier with the maximum number of cookies is given by* $max(\{f(s) | \forall s \in S\}) \vdash min(s)$. *Write a program to implement this algorithm.* (2)

The formal assumption is probably based on the idea that since natural language is ambiguous, some mathematical notation is necessary. This is true in so far as it is not possible to go through a process of clarification by asking questions, which is not the case in ChatGPT, as shown in figure 2a. This does not mean that formal expressions should be avoided, but, as suggested after figure 3, that it is necessary to understand how to use them in this context. Also, that (2) may not be the best way to provide formal specification on ChatGPT.



Sure, I can help you with that! Can you please provide more details on the input and output format expected for the program? For example, how are the number of cookies brought by each soldier represented in the input? And what should the output of the program be?

(a) First response.



Sure, I can help you with that!

Assuming you have a list of the number of cookies brought back by each soldier, here's a Python program to find the soldier who brought the most cookies:

```python
cookies = [4, 7, 2, 9, 5] # list of cookies brought back by each soldier

max_cookies = max(cookies) # find the maximum number of cookies
max_index = cookies.index(max_cookies) # find the index of the soldier who b

print(f"Soldier {max_index+1} brought the most cookies: {max_cookies} cookie
```

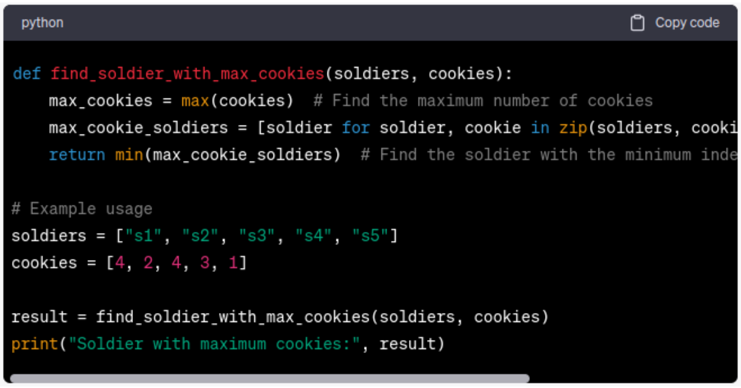
In this program, we first find the maximum number of cookies using the `max()` function, which returns the largest item in a list. We then use the `index()` method to find the index of the soldier who brought the maximum number of cookies. Note that we add 1 to the index because soldiers are usually numbered starting from 1 instead of 0.

Finally, we print out the result using an f-string, which is a convenient way to format strings in Python.

I hope this helps!

(b) Second response.

Figure 2 ChatGPT's responses for the same prompt on different instances.

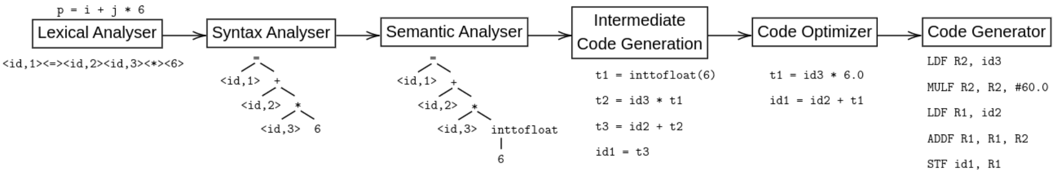


```python
def find_soldier_with_max_cookies(soldiers, cookies):
    max_cookies = max(cookies)  # Find the maximum number of cookies
    max_cookie_soldiers = [soldier for soldier, cookie in zip(soldiers, cooki
    return min(max_cookie_soldiers)  # Find the soldier with the minimum inde

# Example usage
soldiers = ["s1", "s2", "s3", "s4", "s5"]
cookies = [4, 2, 4, 3, 1]

result = find_soldier_with_max_cookies(soldiers, cookies)
print("Soldier with maximum cookies:", result)
```

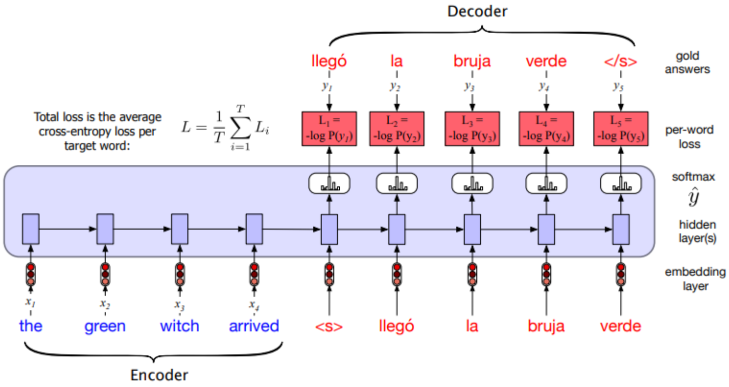
Figure 3 Source code produced by ChatGPT for the formal specification on prompt (2) with circumventing text removed.

4 Insights on a 4th Generation Compiler

At this point it is possible to suggest that ChatGPT (i.e. large language models) can be considered as a translation device. Figure 4 shows a bird's eye view of a compiler as defined by Aho et al. in [1] and a translator as defined by Jurafsky et al. in [5]. The word "translator" is accepted as a synonym for "compiler", so the former can be considered a deterministic translator and the latter a probabilistic one.



(a) A compiler instance as presented in [1].



(b) A translator instance as presented in [5].

Figure 4 Deterministic and probabilistic translator instances.

It can also be argued that the deterministic translator is primarily concerned with syntax (i.e. the structure of sentences based on the Chomsky hierarchy) and the probabilistic translator with semantics (i.e. the relationship between words based on the distributional hypothesis). Therefore, it is not a case of replacement, but of composing these two translation strategies. As a result, if a source is generated with syntactic errors, the generator would produce an improved source by using the error messages as feedback. Note that the use of examples is an important prompt feature due to the few-shot learning property of large language models [2]. This in turn can be manifested as unit tests [11]. In this sense, not only syntactic errors but also behavioural errors eventually introduced by the generator can be automatically corrected. Note also that, not all information may be provided in a prompt. There are two situations to consider. One is the prompt within a project, from which the additional information can be retrieved (e.g. elements of a class, UML blueprints, etc.). If the required information is ambiguous or missing, the compiler is expected to prompt the programmer as shown in figure 2a with each interaction improving either the context or the specification. Note that the correct definition of the prompt is the cornerstone of improved generation; a context with more information than necessary is just as harmful as a context with no information at all. A structured as such is depicted on figure 5.

It is worth noting that if the complexity of the specification becomes unmanageable, it can be split or simply deleted to start again. Splitting opens up the possibility of tackling increasingly large software. Regardless of the development process used (prescriptive or

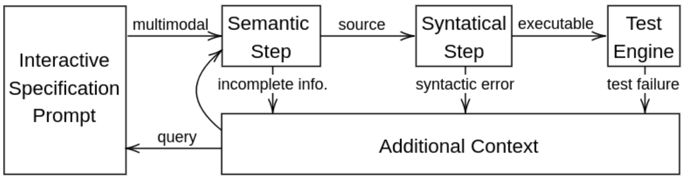


Figure 5 Suggested structure for a 4th generation compiler based on large language models.

agile), the overall organization of the project elements ends up being arranged in a tree-like structure. The same structure can be applied to a series of prompts that make up the software, helping to establish the correct context for each prompt. Consider the following Use Case 2.0 [4] partial scenario:

- Use-case: Registering to SLATE
  1. Authentication Flow
     a. The enrollee provides its e-mail, the SuD sends an e-mail with a confirmation link
     b. The SuD receives a confirmation and set the status to “authenticated enrolee”
  2. [authenticated enrolee] Basic Flow for Author Registration
     a. ...

For this discussion, consider step 1a. Note that this step is a self-contained specification and is sufficient to understand the desired behaviour. However, it is not a prompt in the sense that there is much information missing. As a reference, the 4+1 view model [7] suggests the existence of five views: 1) scenario, 2) logic, 3) development, 4) process and 5) physical. This step description only satisfies the scenario view. For example, on which server is this web service expected to run? In addition, a scenario is expected to satisfy the FURPS+ [3]; and this step only satisfies the “F”. For example, what would the content of the confirmation email be? Therefore, prompting is not equivalent to a requirement analysis yet, as any development task, derives from it.

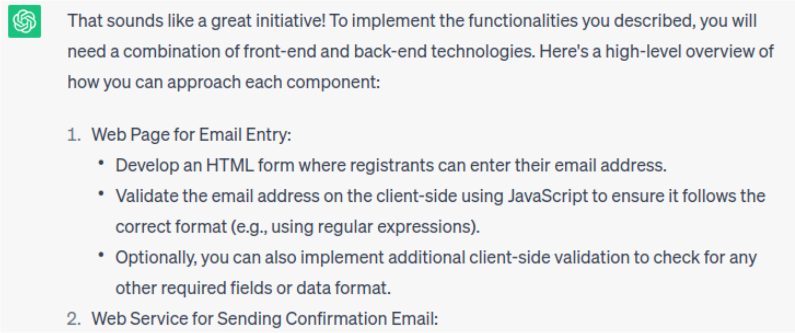
Note that steps 1a and 1b form a slice (a coherent part of a use case that can be elaborated into a deployment [4]). Therefore, to produce a deliverable for this slice, a prompt must be written from these two steps. Considering the constraints presented, the 4+1 view model and FURPS+, it becomes clear that only a chat-based structure is not sufficient to express a whole, industrial-scale software (even a slice of it). One possibility of structure to explore is that provided by literate programming, see [6].

In this sense, from a human perspective, it is not a matter of producing a single prompt, but of producing a structured document composed of several prompts addressing different concerns. Given such a document, the compiler would be expected to make two moves. One towards the refinement of each prompt, i.e. a chat-based interaction aimed at producing an improved prompt. Another towards the unfolding of additional prompts, i.e. the creation of additional slots for prompts to address specific problems. For large software, there could be several specification files, each for a slice. In this perspective, the compiler would act as a specification co-pilot. Note that the context window used by ChatGPT is about 2048 tokens, so managing the context of each prompt is also a feature to be considered.

Another support expected from the compiler is the generation of test cases. Following a behaviour-driven development rationale [11], this leads the programmer to consider several scenarios that improve the resulting program. From the perspective of this paper, this means that the programmer would either refine previous prompts or introduce new prompts.

A possible literate programming early paragraph for the authentication slice could be the one presented in (3). Note that this is not a proper programming prompt yet, submitting it to ChatGPT, an excerpt can be seen in 6, it retrieves several suggestions that illustrate what the refinement and unfolding moves would look like. For example, validating the email with regular expressions would be a refinement on (3); describing the appearance of the HTML form (e.g. colour, logo, etc.) would be an unfolding prompt.

This programme is part of a conference registration platform and is designed to verify that the email provided by a registrant is a valid one. This requires: 1) a web page that allows the registrant to enter their email address; 2) a web service that receives this address and sends the confirmation email; and 3) another web service that generates the confirmation page and waits for the registrant to retrieve it. (3)



That sounds like a great initiative! To implement the functionalities you described, you will need a combination of front-end and back-end technologies. Here's a high-level overview of how you can approach each component:

1. Web Page for Email Entry:
   - Develop an HTML form where registrants can enter their email address.
   - Validate the email address on the client-side using JavaScript to ensure it follows the correct format (e.g., using regular expressions).
   - Optionally, you can also implement additional client-side validation to check for any other required fields or data format.
2. Web Service for Sending Confirmation Email:

Figure 6 Excerpt of ChatGPT response when prompting (3).

The presented example is based on plain literate programming. Since large language models are currently turning into multimodal, it is also possible to enrich the specification with diagrams and other images [15].

5 Conclusion

This paper introduced the possibility of interpreting large language models as a fourth generation programming language, based on the notion that large language models are capable of translating a natural language specification into well-formed source code. This notion opens up and strengthens a wide range of research areas, including further developments from the compiler perspective to specific prompt engineering techniques for producing programs. The key discussion is that the current fears and suspicions raised by this technology are analogous to those that arose during the transition from the second to the third generation of programming languages. It is therefore a natural phenomenon that should be resolved on its own terms, as far as the proposal presented in this paper is concerned.

Comparing this approach with a current compiler raises the question of the probabilistic nature of large language models (the same prompt produces two outputs). In short, this can be addressed by a fixed, perhaps optimal, internal state with respect to the compilation task. However, the strengths and weaknesses of such an approach are a subject for future work. This is somehow related to explainability, a flourishing area of research that is expected to provide answers to questions about why a model has generated a particular piece of code.

Note that ChatGPT may produce incorrect code, but also that it is not tuned for programming, and it will eventually become outdated. However, it is expected that a 4th generation compiler will be able to deal with such problems, just as 3rd generation compilers will be able to deal with code with syntactic problems (perhaps one way, inspired by genetic algorithms, would be to produce a few generations and select the best by a set of parameters).

This assertion assumes that the specification has been written properly and correctly, but it is also necessary to consider that intentional and induction errors will still exist, but at a higher level of abstraction. In this sense, if the executable doesn't run as expected, the programmer should concentrate on fixing the prompt as he currently does with the source.

This leads to issues related to the dataset, which will also need to be addressed in future work. Then, considering the compilation task, it would be necessary to understand the composition of the dataset. Also, which setup would perform better: a general large language model or a fine-tuned one? What would be the fine-tuning parameters? Also, when considering prompting, which software engineering tools, methods and principles are appropriate and which are not. Note that multimodal prompting requires a refresh on model-driven development, see [8]. Finally, this seems to be a promising area of research to be explored further.

References

1 Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. *Compilers: principles, techniques and tools.* Pearson, 2020.

2 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33, 2020.

3 Robert B Grady and Deborah L Caswell. *Software metrics: establishing a company-wide program.* Prentice-Hall, Inc., 1987.

4 Ivar Jacobson, Ian Spence, and Brian Kerr. Use-case 2.0. *Queue*, 14(1):94–123, 2016.

5 Dan Jurafsky and James H. Martin. *Speech and Language Processing.* draft (https://web.stanford.edu/j̃urafsky/slp3/), third edition, 2023.

6 Donald Ervin Knuth. Literate programming. *The computer journal*, 27(2):97–111, 1984.

7 Philippe B Kruchten. The 4+ 1 view model of architecture. *IEEE software*, 12(6):42–50, 1995.

8 Chris Raistrick, Paul Francis, John Wright, Colin Carter, and Ian Wilkie. *Model driven architecture with executable UML*, volume 1. Cambridge University Press, 2004.

9 Elvis Saravia. Prompt engineering guide, 2023. URL: `https://www.promptingguide.ai/`.

10 Robert W Sebesta. *Concepts of programming languages.* Pearson Education, 2019.

11 J.F. Smart and J. Molak. *BDD in Action, Second Edition: Behavior-Driven Development for the Whole Software Lifecycle.* Manning, 2023.

12 Bernhard Thalheim and Hannu Jaakkola. Model-based fifth generation programming. *Information Modelling and Knowledge Bases*, 31:381–400, 2020.

13 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

14 Karl Wiegers. *More about software requirements.* Microsoft Press, 2005.

15 Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *arXiv:2302.00923*, 2023.

16 Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

17 Majd Zohri Yafi. *A Syntactical Reverse Engineering Approach to Fourth Generation Programming Languages Using Formal Methods.* PhD thesis, University of Essex, 2022.

Hierarchical Data-Flow Graphs

José Pereira
Checkmarx, Braga, Portugal

Vitor Vieira
Checkmarx, Braga, Portugal

Alberto Simões
Checkmarx, Braga, Portugal
2Ai, School of Technology, IPCA, Barcelos, Portugal

Abstract

Data-Flows are crucial to detect the dependency of statements and expressions in a programming language program. In the context of Static Application Security Testing (SAST), they are heavily used in different aspects, from detecting tainted data to understanding code dependency.

In Checkmarx, these data flows are currently computed on the fly, but their efficiency is not the desired, especially when dealing with large projects. With this in mind, a new caching mechanism is being developed, based on hierarchical graphs.

In this document, we discuss the basic idea behind this approach, the challenges found and the decisions put in place for the implementation. We will also share the first insights on speed improvements for a proof of concept implementation.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages; Software and its engineering → Compilers; Theory of computation → Graph algorithms analysis

Keywords and phrases Data Flow, Static Application Security Testing, Hierarchical Graphs

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.11

Funding *Alberto Simões*: This paper was partially funded by national funds, through the FCT/MCTES of the projects UIDB/05549/2020 and UIDP/05549/2020.

1 Introduction

SAST (Static Application Security Testing) [8] is one of the different techniques employed by Checkmarx for analyzing source code and scanning it for security vulnerabilities. As the name implies, SAST tools scan the source code without executing it. The identification of potential security weaknesses is performed after constructing an abstract syntax tree (AST) for the code being analyzed and using a query system to find specific code patterns. SAST can be used to detect vulnerabilities such as SQL injection, cross-site scripting, or buffer overflow situations.

One important feature of SAST tools is the possibility to compute data flows. Data flows allow the understanding of which expressions have their values affected by the values of variable declarations or other expressions. This is useful, as an example, to understand if a query to a database might be influenced directly by the user input, or if, during the data flow, there is any kind of sanitization[1] preventing SQL Injection. One of the first works using data flows to analyze software reliability was conducted by Fosdick and Osterweil (1974) [4]. Their work includes a comprehensive explanation of what are data flows and how to represent them as a graph. Data flows are an important part of SAST implementations [7].

---

[1] Sanitization is the term used to any code that prevents the vulnerability to occur, remediating it.

© José Pereira, Vitor Vieira, and Alberto Simões;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 11; pp. 11:1–11:9
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Checkmarx SAST engine does data flow analysis on demand, meaning that, whenever a calculation is requested, rather than traversing a graph looking for paths between sources and sinks[2], it computes which are the next AST nodes to visit checking what immediate data is affected by the first. This process iterates until a destination is eventually reached, or there are no more adjacent nodes.

This might seem inefficient but, in fact, the traditional strategy of path-finding on an actual graph was used before and it produced results 30% slower on average than the current solution. The cause for this is the unnecessary graph expansion for every node on the abstract syntax tree, even when no flow calculation is requested from them, allied with the fact that the computational weight of finding a path in a large graph is heavier than computing adjacency between nodes. Still, the process consumes a considerable portion of scan time, averaging 30% scan time for most source projects.

The proposal is to cache a graph in which there's enough context to avoid searching impossible paths when specific sources and sinks are used. For that, clusters of vertices are created, each one representing an entry or exit point of a certain context. The idea is to be able to match any node on the AST to a cluster of vertices and therefore understand if it is worth exploring for a specific path. There can be several kinds of grouping strategies for the clusters as we will discuss later. For the Proof-of-Concept (POC) described here, nodes were grouped functions/methods in which they are encapsulated.

In the next Section, a small literature review is presented focusing on the main concepts used in the implementation of this solution. Follows Section 3 where the current data flow engine algorithm is explained, allowing a better understanding of the proposed approach. Section 4 describes the POC, including a first evaluation of the obtained results. Section 5 concludes with some final remarks and describes future work.

2 Literature Review

As a first note, Data Flows and Static Application Security Testing are not new concepts. As shown in this section, most of the concepts have more than 50 years. Nevertheless, these concepts are the base that support the development of the Proof-of-Concept here described.

In 1976, Allen and Cocke [1] present multiple situations in which a program data flow can be of use. One of the most relevant for this work is to know what data use might be affected by a particular variable definition, and the inverse, for a given use of a variable, the definitions which can potentially supply values to it. They provide a formal definition of what a data flow is, which might be summarized as a connected, directed graph with a single entry point (usually a variable declaration), and where each vertex[3] represents a statement or expression whose value depends on the graph entry point or an expression or part of an expression which modifies that data item.

As referred to in the introduction, Fosdick and Osterweil [4] present in their paper the definition of a data flow, including a couple of examples of how to represent them as directed graphs, as well as how these flows can be used to detect software implementation problems. They present a system to analyze Fortran code, called DAVE, that detects some of the most common data flow anomalies. DAVE performs this analysis by computing a flow graph search for each variable in a given unit and analyzing subprograms. This is, probably, one of the first SAST implementations.

---

[2] The term *source* or *input* are traditionally used as the first node in the flow search, while the *sink* is the target node (or target nodes).

[3] For clearness, the term *node* will be used to refer to an item in the Abstract Syntax Tree, and the term *vertex* will be used to refer to graph vertices. In most cases, a vertex represents a node, and therefore, some confusion may arise.

In their book [6], Khedker, *et al.* describe many different approaches for data flow analysis, and how they can be used to find different situations. The book finishes with a chapter on implementing data flow analysis in C programs using the GCC C compiler.

There are other examples of data flow analysis, that are not listed here, as they are focused on a single programming language, as the examples shown above.

3 Checkmarx Lazy Flow

Checkmarx SAST solution supports data flow computation. It is executed on demand, every time a specific flow is required. The flow can be computed in the flow direction (execution flow) or backwards. This allows the engine to choose the direction that promises a relatively smaller number of paths. This feature is known as Lazy Flow and is used by most queries used to find vulnerabilities.

The Lazy Flow process receives a set of input nodes from the Domain Object Model[4] (DOM), and a set of sink nodes. The algorithm is also able to deal with a set of sanitizer nodes. These nodes are user-defined and consist of a set of specific instructions that should be considered a flow barrier. As an example, consider the storage of personal information as a password in a database. The flow could be discarded if the password gets encrypted (thus making the flow not vulnerable). The encryption functions act as barriers and are considered sanitizers.

The Lazy Flow algorithm considers each expression or statement as a potential hop in the flow. Each hop has two visitors that decide the possible next (or previous) nodes to look up, according to the flow direction. The search ends whenever a sanitiser node is found or when a maximum number of hops was visited. The algorithm only considers the shorter path between two specific nodes for efficiency.

This flow computation is performed on-the-fly, every time a flow is requested as shortly described in the Section 1. This means that highly reused code blocks are scanned over and over again for different vulnerability detection, producing a time overhead. This is the main problem the proposed POC tries to tackle.

Note that the fact that the graph is not persistent (there is not a proper graph representation for the possible flows) does not have a real impact in terms of performance, as the Domain Object Model acts as the graph, and only in very specific situations the edge computation is not immediate.

4 Data Flow Hypergraph

A common solution for reducing the complexity of a graph is creating an abstraction that clusters vertices together. Each cluster becomes a new vertex, and edges between these clusters are aggregations of the original edges. Inside each cluster, a vertex is a graph.

These data structures are usually referred to as hierarchical graphs and are well-studied. The path-finding algorithms perform at the top level and, when a concrete path is required, look inside the relevant clusters to compute the real path.

These structures are heavily used in navigation, being in artificial or real worlds, and therefore applications are found in the areas of video games, robotics and geographic information systems. Examples of applications of Hierarchical Graphs (HG) include Pelechano

---

[4] The Domain Object Model can be perceived as an Abstract Syntax Tree whose structure is shared among different languages, and that does not mimic exactly the parsed code, but its semantics.

and Fuentes (2016) [9] work for path-finding in Meshes, or the work by Antikainen (2013) [2] use of HG for non-uniform traversal costs. Other examples can be found in the literature [5, 11, 3]. We will use the term Hypergraph to refer to the concrete HG implementation.

For the use case under consideration, one of the first discussions is about the clustering approach. How to consider two nodes from the DOM to be part of the same cluster? This discussion will be presented in Section 4.1. Follows Section 4.2 with an in-depth explanation of the adopted clustering approach. Section 4.3 explains how the HG is being used and presents some analysis of the obtained results.

4.1 Clustering DOM Nodes

One first decision to take is how to cluster the DOM nodes, to produce the hierarchical graph. The approach to cluster the nodes can be defined accordingly with different semantic approaches. The two main ideas discussed were:

- Cluster nodes by the file in which they appear. This was the first idea given the parallel work on an incremental parsing mechanism that deals with the change of a single file in a project repository. This clustering would make the process of updating the HyperGraph easy. Nevertheless, there is no clear definition of what a flow inside a file is. While in some languages or some projects that might exist, a simple file that is just a library would be hard to be properly grouped as a cluster, as it would have an extremely large number of inbound and outbound edges.
- Cluster nodes by the function in which they appear. This would mean that a vertex in the Hypergraph would be a function and connections will represent the flows that enter or exits that function. While the number of vertexes will rise, compared with the previous approach, there is a clear semantic meaning of edges: they are method invocations or stack frames. Curiously, this is quite similar to the second approach proposed by Sharir and Pnueli [10] in 1981.

While the chosen approach was to cluster based on functions, and as it will be seen in the next sections, some changes on the original idea were performed to encompass different entry and exit points from methods. This will be described in the next section.

4.2 Method-based Clustering

For this POC, the chosen strategy to cluster vertices was by method/function invocation. The term method will be used to refer to both functions and methods because the DOM was designed for object-oriented programming languages and despite being able to also support other paradigms, functions are wrapped in default classes for the sake of compatibility, making them static methods.

A DOM node belongs to a specific method cluster if it is under the methods sub-tree. Therefore, computing the first ancestor which is a method declaration is enough to infer the vertex the node belongs to. Vertices on the Hypergraph should be entry points of methods, such as parameters, and exit points, like return statements. Note that data can flow in and out of methods through other kinds of nodes. Arguments can lead us to other clusters/methods and method calls can make data flow into a method.

Edges on the Hypergraph are flows between the entry and exit points. Whenever there are nodes in between this flow, these sequences are stored in the graph, annotating it. This sequence represents the data flow from the method entry point to an exit point.

Follows a simple example. Consider the C#-like code sample in Listing 1, which describes a basic program that would take in an input string and execute one SQL command, that is affected by that same input.

Listing 1 Sample C# code with interprocedural calls.

```
void main() {
    string someInput = readFromStdIn();
    handleInput(someInput);
}

void handleInput(string someInput) {
    if(isBadInput(someInput)) {
        printErrorMessage(someInput);
        abort();
    }
    else {
        string preparedStatement = prepareStatement(someInput);
        executeSqlStatement(preparedStatement);
    }
}

string prepareStatement(someInput) {
    return "SELECT␣*␣FROM␣USERS␣WHERE␣USERNAME␣=␣" + someInput;
}

void executeStatement(string sqlStatement) {
    printResults(database.Execute(sqlStatement));
}
```

Figure 1 shows that same code, where some nodes are highlighted. In green, we have entry data points, and in red we have exit points. Entry points are, mostly, parameter declarations (lines 7, 21 and 26) and method calls (lines 3, 9, 16 and 28). Exit points are return statements (line 23) or parameters inside method calls (lines 4, 9, 11, 16, 17 and 28).

These will be the vertices on the Hypergraph. For the edges, we can compute the data flow between these nodes and aggregate the sequences between entry and exit points of the same method. For simplicity, Figure 2 shows only the relevant flows between the entry and exit nodes. In this image, the purple arrows are edges between an entry point and an exit point of a method, while the yellow arrows are edges between clusters.

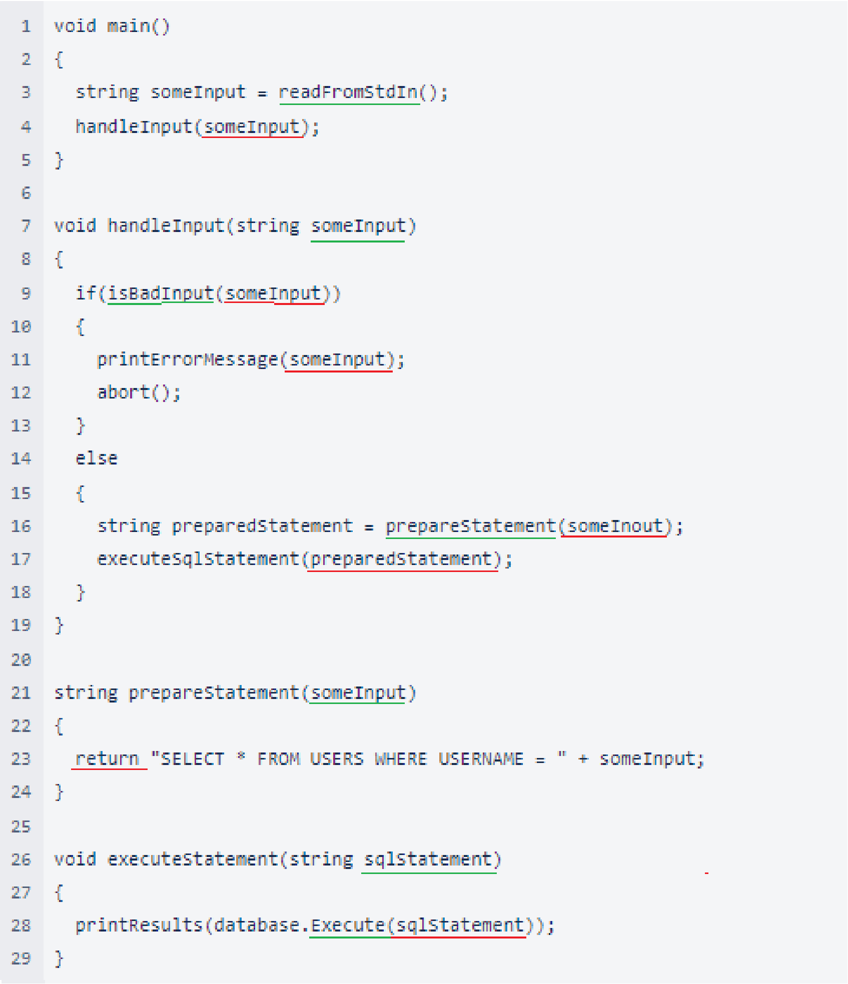
The graph is stored in QuickGraph[5] library using specific vertex and edge definitions. Vertices include the entry or exit node information, and edges include the full path between these nodes. Looking at Figure 2, purple arrows include paths, while yellow arrows are just empty edges. The graph is bidirectional, thus allowing the computation of forward and backward flows.

4.3 Hypergraph Application and metrics

While the final implementation will feature a rewritten flow engine, that will take advantage of the Hypergraph to decide which paths are worth exploring, following the usual implementation of hierarchical graphs, currently the POC uses the Hypergraph information as a cache, fast-forwarding the computation of flows inside methods.

---

[5] `https://kernelith.github.io/QuikGraph/`



```
void main()
{
  string someInput = readFromStdIn();
  handleInput(someInput);
}

void handleInput(string someInput)
{
  if(isBadInput(someInput))
  {
    printErrorMessage(someInput);
    abort();
  }
  else
  {
    string preparedStatement = prepareStatement(someInout);
    executeSqlStatement(preparedStatement);
  }
}

string prepareStatement(someInput)
{
  return "SELECT * FROM USERS WHERE USERNAME = " + someInput;
}

void executeStatement(string sqlStatement)
{
  printResults(database.Execute(sqlStatement));
}
```

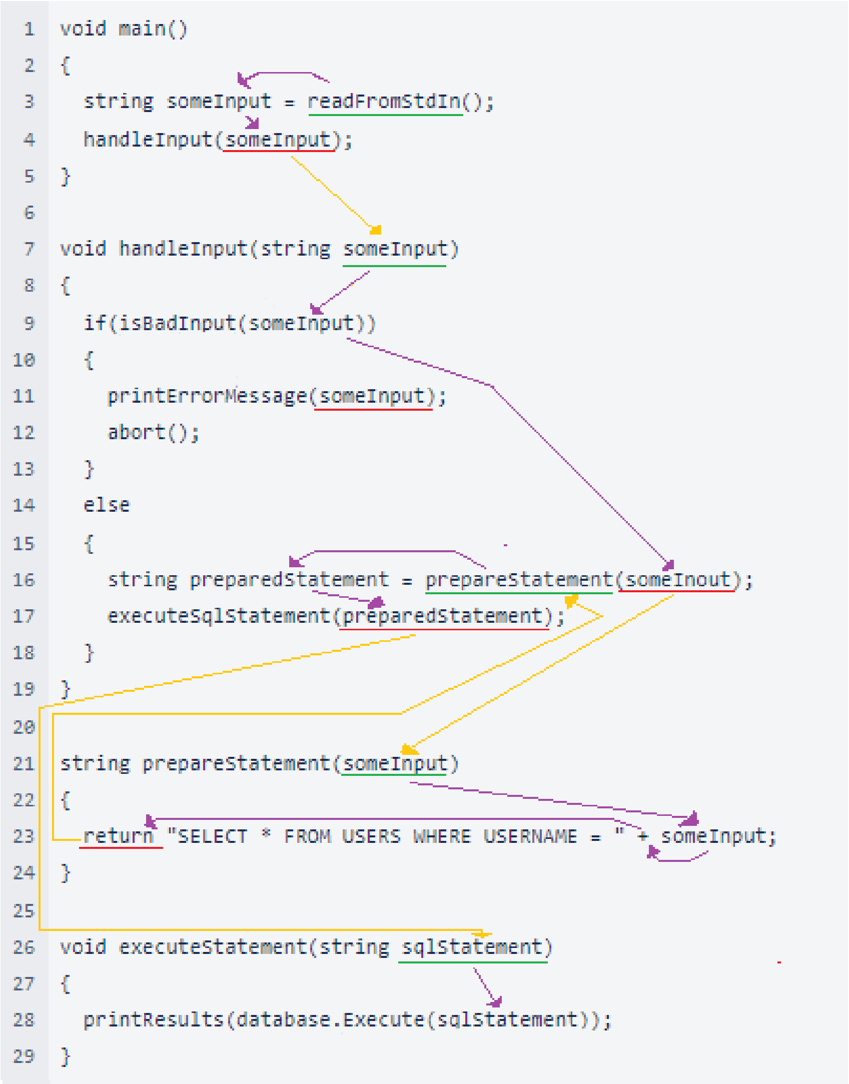
Figure 1 Inbound (green underlines) and outbound flow nodes (red underlines).

Consider an input node and a sink. Given the input node, and considering it is not in the current function scope, the traditional flow algorithm is computed until a relevant node is visited: a method call, a parameter declaration, a return statement, or a parameter in a method call. These are the entry and exit points for the Hypergraph, as stated earlier. At this point, the Hypergraph is queried. If the sink node is inside the Hypergraph cluster of nodes, the traditional flow algorithm keeps in charge. If not, the Hypergraph cluster node is used and the path is fast-forwarded until the end of the flow (next flow jump). While this process does not reduce the amount of visited paths it reduces the amount of calls to the path-finding algorithm. This would result in efficiency improvements for large methods, and little or even an efficiency decrease for small methods.

This prototype implementation was used to measure the performance impact on data flow calculations of some benchmark projects[6], producing the results presented in table 1.

---

[6] These are some open-source projects written in different languages, that are used internally for benchmark purposes.



```
void main()
{
  string someInput = readFromStdIn();
  handleInput(someInput);
}

void handleInput(string someInput)
{
  if(isBadInput(someInput))
  {
    printErrorMessage(someInput);
    abort();
  }
  else
  {
    string preparedStatement = prepareStatement(someInout);
    executeSqlStatement(preparedStatement);
  }
}

string prepareStatement(someInput)
{
  return "SELECT * FROM USERS WHERE USERNAME = " + someInput;
}

void executeStatement(string sqlStatement)
{
  printResults(database.Execute(sqlStatement));
}
```

Figure 2 Flows inside functions: purple arrows are paths along lines of code while yellow arrows are just empty edges.

The first project was the only one with a positive impact from the altered Lazy Flow algorithm, using the Hypergraph. By correlating flow calculation statistics between these projects, several things can be observed. AccorStruts is the project leading in terms of time spent searching for the next references from the total time of flow calculation. Reference finding is the act of mimicking the program's control flow in order to understand which symbol reference is the data flowing into next. This is a costly procedure that has specific logic for each different DOM node.

One other observation, looking into the paths returned by the Lazy Flow algorithm, is that the number of resulting flows is different when using the unaltered Lazy Flow and the version using the Hyperhraph. This is an indicator that the algorithm is probably lacking context and making some wrong assumptions about node sequences when compared to the pure Lazy Flow approach. Finally, the only coherent and significant number regarding the number of path reuses from the Hypergraph is with the AccorStruts project. Every other

Table 1 Times before and after the Fast-Forward implementation (LOC=Lines of Code).

Project	LOC	Query time	Query time with FF
AccorStruts	79.857	02:32.3	01:35.7
WebGoat	117.234	10:39.4	11:09.6
Qmxpp	20.478	00:24.0	00:30.3
Bookstore	17.588	00:25.9	00:43.0

project has a very low number of reuse, meaning that either the methods are used only once per flow, that the clustering approach was not the best, or simply implies bugs in the implementation.

5 Conclusions and Future Work

In this article, we present a first approach to develop a proof of concept for a hierarchical graph to compute data flows for SAST. We are still in the early stage of the process, with a prototype that is already allowing the analysis of the graph reuse and its impact on the flow computation efficiency. Nevertheless, a lot of effort is still required to have a fully operational solution. And more investment should be put into the POC to extract clearer conclusions.

In the making of this article, the generation of the Hypergraph and its reuse was achieved by piggybacking the LazyFlow logic. The next steps would be to match the exact same results as the standard Lazy Flow with the HyperGraph approach, creating a proper base for a benchmark. After, the actual idea that served as motivation for the POC should be put into practice. For that, a custom graph path-finding algorithm needs to be developed, taking into account the same context that Lazy Flow uses. For instance, one cannot leave a method cluster through means of a return statement into a different instance from where it was entered.

Another point that requires further improvement is the use of output or reference parameters, that are available in some languages. At this point, those were deliberately ignored to have a simpler setup for initial analysis of the improvements resulting from this approach.

References

1 F. E. Allen and J. Cocke. A program data flow analysis procedure. *Communications of the ACM*, 19(3):137, March 1976. `doi:10.1145/360018.360025`.

2 Harri Antikainen. Using the hierarchical pathfinding a∗ algorithm in GIS to find paths through rasters with nonuniform traversal cost. *ISPRS International Journal of Geo-Information*, 2(4):996–1014, October 2013. `doi:10.3390/ijgi2040996`.

3 Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):1–22, 2004.

4 Lloyd D. Fosdick and Leon J. Osterweil. Data flow analysis in software reliability. *ACM Computing Surveys*, 8(3):305–330, September 1976. `doi:10.1145/356674.356676`.

5 Matthias Grundmann, Vivek Kwatra, Mei Han, and Irfan Essa. Efficient hierarchical graph-based video segmentation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2141–2148, 2010. `doi:10.1109/CVPR.2010.5539893`.

6 Uday P. Khedker, Amitabha Sanyal, and Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press, March 2009.

7 Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. Data flow analysis. In *Principles of Program Analysis*, pages 35–139. Springer Berlin Heidelberg, 1999. `doi:10.1007/978-3-662-03811-6_2`.

8 Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer Berlin Heidelberg, 1999. `doi:10.1007/978-3-662-03811-6`.

9 Nuria Pelechano and Carlos Fuentes. Hierarchical path-finding for navigation meshes (HNA∗). *Computers & Graphics*, 59:68–78, October 2016. `doi:10.1016/j.cag.2016.05.023`.

10 Micha Sharir and Amir Pnueli. Two approaches to interprocedural data flow analysis. In Steven S Muchnick and Neil D Jones, editors, *Programme Flow Analysis*, pages 189–233. Prentice Hall, April 1981.

11 Edgar-Philipp Stoffel, Korbinian Schoder, and Hans Jürgen Ohlbach. Applying hierarchical graphs to pedestrian indoor navigation. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS ’08, New York, NY, USA, 2008. Association for Computing Machinery. `doi:10.1145/1463434.1463499`.

Type Annotation for SAST

Marco Pereira
Checkmarx, Braga, Portugal
University of Minho, Braga, Portugal

Alberto Simões
Checkmarx, Braga, Portugal
2Ai, School of Technology, IPCA, Portugal

Pedro Rangel Henriques
ALGORITMI Research Centre/ LASI, DI-University of Minho, Braga, Portugal

Abstract

Static Application Security Testing (SAST) is a type of software security testing that analyzes the source code of an application to identify security vulnerabilities and coding errors. It helps detect security vulnerabilities in software code before deployment reducing the risk of exploitation by attackers.

The work presented in this document describes the work performed to upgrade Checkmarx's SAST tool allowing the execution of vulnerability detection taking into account expression types. For this to be possible, every expression in the Document Object Model needs to have a specific type assigned accordingly to the kind of operation and to the different operand types.

At the current stage, this project is already supporting the expression type annotation for three programming languages: C, C++ and C#. This support has been done through the addition of a new Resolver Rule to the Resolver stage, allowing for the generalization of languages. We also compare the complexity of writing vulnerability detection queries with or without access to type information.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages; Software and its engineering → Compilers

Keywords and phrases Static Application Security Testing, Type Annotation, C, C++, C#

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.12

Funding *Alberto Simões*: This paper was funded by national funds (PIDDAC), through the FCT - Fundação para a Ciência e Tecnologia and FCT/MCTES under the scope of the projects UIDB/05549/2020 and UIDP/05549/2020.
Pedro Rangel Henriques: This work has been supported by FCT - Fundação para a Ciência e Tecnologia within R&D Units Projects Scope: UIDB/00319/2020.

1 Introduction

Static Application Security Testing (SAST) is a fundamental tool for the software development life-cycle. Although SAST is limited to detecting vulnerabilities and software issues during compile time, not being able to find run-time bugs, it is able to discover a large variety of problems. The number of results depends on the quality of the developed queries that find vulnerable code structures. But, these queries' effectiveness is constrained by the information the language parsers are able to extract and infer from the source code.

This section will start by discussing what is SAST and type annotation and concludes with a brief description of the main goals of this work. Section 2 discusses some related work, following Section 3 with an introduction to Checkmark's product pipeline. The implementation of the type annotation system is depicted in Section 4 and in Section 5 we

© Marco Pereira, Alberto Simões, and Pedro Rangel Henriques;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 12; pp. 12:1–12:13
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

discuss the changes needed to support C, C++ and C# with the same code base. Finally, Section 6 compare the complexity of vulnerability detection queries before and after the availability of type annotation. Section 7 concludes with some remarks and future work.

1.1 Static Application Security Testing

Static Application Security Testing (SAST) [9] is a kind of security testing that is performed on the source code of an application to identify security vulnerabilities and coding errors before the application is deployed. It analyzes the application's source code for potential security flaws such as input validation errors, buffer overflow vulnerabilities, and SQL injection flaws. SAST tools use a combination of pattern matching, data flow analysis, and other techniques to identify vulnerabilities, providing detailed information about each issue detected and guidance on how to remediate them.

SAST is an essential part of the software development life cycle (SDLC) because it helps detect and address security issues early in the development process, reducing the cost and effort required to fix them later on. By identifying potential vulnerabilities before the code is deployed, SAST reduces the risk of exploitation by attackers and helps ensure the security and integrity of the application.

Usually, SAST scans an application before the code is compiled, and thus is known as white box testing.

1.2 Checkmarx's Engine

Checkmarx's primary product is a SAST Engine responsible for processing projects in various programming languages. To guarantee language independence the engine produces a generic structure, named DOM (Domain Object Model). Thus, the DOM is a language-agnostic structure that contains information on variable declarations, assignments, conditions, expressions and so on. The DOM is then queried by different queries in order to find vulnerabilities in the code like SQL Injections. Each query is created following the flow of the vulnerability it tries to detect, looking either for code constructs or to data-flows.

1.3 Type Annotation

Programming languages can be statically typed or dynamically typed (or even, in some special situations, completely non-typed). For typed languages, each variable or expression yields a type, either specified clearly (for static typing) or inferred by the context (for dynamic typing) [5, 12].

Most programming languages allow declaring variables with a specific type. This type can range from a byte or character, an integer, used to represent whole numbers, to floating point values or even pointers and objects.

Along with the type of a variable, there can be information on its size. The most common example is declaring an integer, where it can be specified to take two, four, eight or sixteen bytes (in C the default integer usually takes four bytes and the remaining sizes can be stated using the keywords `short`, `long` or the repetition `long long`). There can be also information on the signedness of the variable: whether the variable can represent both positive and negative values (the default) or only positive ones.

1.4 SAST Problems solved with Type Annotation

There are different kinds of software vulnerabilities that can be tackled using type information. This section describes one real-world example that falls in the class of Integer Overflow situations.

Integer Overflow, also known as Wraparound, has been ranked number 13 in 2022 CWE Top 25 vulnerabilities [6] and has been consistently ranked top 15 in the past years. The vulnerability consists of trying to store a too-large (or too-small) value in a variable, whose data type is not able to deal with. If we consider an integer variable, with default size and signedness modifiers, in a C program running on a 64-bit machine, it will be able to store up a maximum value of 2 147 483 647. This means that, if a variable of this type is yielding this value and is incremented (for example, using the increment operator), the value will wrap, and the variable will have the value -2 147 483 647.

An example of an integer overflow that results in a buffer overflow was detected in an older version of OpenSSH (v3.3) [1]. For the code presented in Listing 1 the variable `nresp` is obtained through a user-controlled function. This means that if the variable gets any value higher than $\frac{1}{8} \times$ `MAX_UINT`, the expression inside the `xmalloc` function will wrap (here we consider that we are running in a 64-bit machine, where a pointer is eight bytes). Given that `xmalloc` function parameter is an unsigned integer, the wrapped value will be too small to deal with the size of the packed being received, and data will be written in non-reserved memory.

Listing 1 OpenSSH 3.3 excerpt vulnerable to integer overflow

```
nresp = packet_get_int();
if (nresp > 0) {
    response = xmalloc(nresp * sizeof(char*));
    for (i = 0; i < nresp; i++)
       response[i] = packet_get_string(NULL);
}
```

While SAST is unable to detect the value present in the `nrest` variable, it can detect that variables of the same type are being multiplied and that their values are not checked for possible overflows.

1.5 Objectives

The goal of the development described in this document is to create a proof of concept system that adds information about types to each declared variable and each and any expression that uses them. This information should be as complete as possible, including not only the type itself, but also its size or signedness details, or even the target type in case of arrays, pointers or objects.

Given that Checkmarx's tool is language-independent, the developed solution must be extensible to support statically or dynamically typed languages with the same code base.

2 Related work

There is not much work related to type annotation for SAST. Nevertheless, in this section we present some related work at different levels: on how to represent types, on type inference tools and on tools that rely on type information to find code vulnerabilities.

A comparison can be performed with the way Checkmarx deals with languages and the way LLVM (formerly known as Low-Level Virtual Machine) does. LLVM is a collection of modular and reusable compiler and tool-chain technologies. It provides a set of tools for building compilers, code analyzers, debuggers, and other related software. The key idea behind LLVM is to provide a set of low-level, platform-independent instructions and optimization steps that can be used to build high-level language compilers.

In LLVM, type annotations are added to the intermediate representation during the front-end stage of compilation. The front end is responsible for translating the source code of a programming language into the LLVM intermediate representation, much like the way Checkmarx's Engine constructs a Domain Object Model (DOM). To represent types, the LLVM project provides a type system that is designed to be platform-independent and flexible. It includes built-in types such as integers and floating-point numbers and user-defined types such as structures and arrays. Figure 1 presents a diagram that describes all the implementations of the LLVM type system [11].

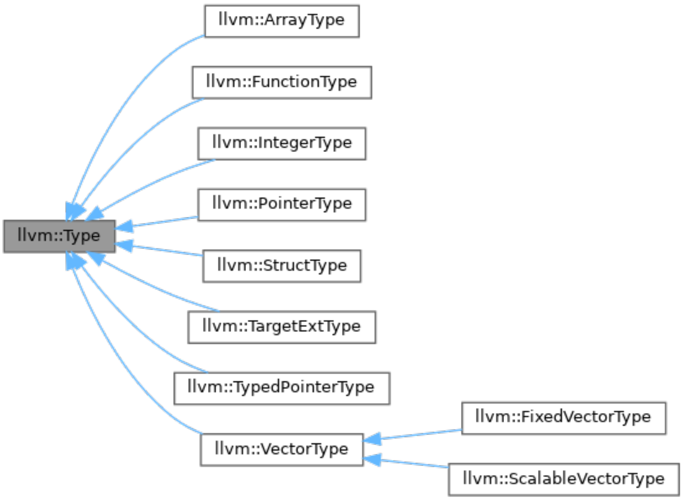


Figure 1 LLVMType diagram.

Type inference, by itself, is common, especially for dynamically typed languages. Note that these language interpreters can benefit from run-time to perform their type analysis. Nevertheless, there are works for static type analysis for these languages. For the Python programming language, such an example is presented by [4]. A more complex example for type inference [3] has the goal to infer types from code executable files, and therefore, not having any hints from the code itself.

A technical report [7] from the University of California at Berkeley focuses on a similar problem as some of the queries being developed at Checkmarx that will take advantage of type inference. Their goal is to detect signedness conversions that can cause application bugs. Their work is developed on top of the Valgrind [8] tool.

3 Checkmarx's Pipeline

In this section, some information on the current state of Checkmarx's engine pipeline will be provided for context.

Initially, the source code gets parsed by ANTLR [10], generating a parse tree. This tree is then used by two sets of visitors. The first set of visitors creates a global symbol table for the variables found, trying to store their type information, accordingly to their declaration type (statically typed languages) or based on the values that are assigned to them. The second set of visitors produces an abstract syntax tree (in fact, it is called Universal Abstract

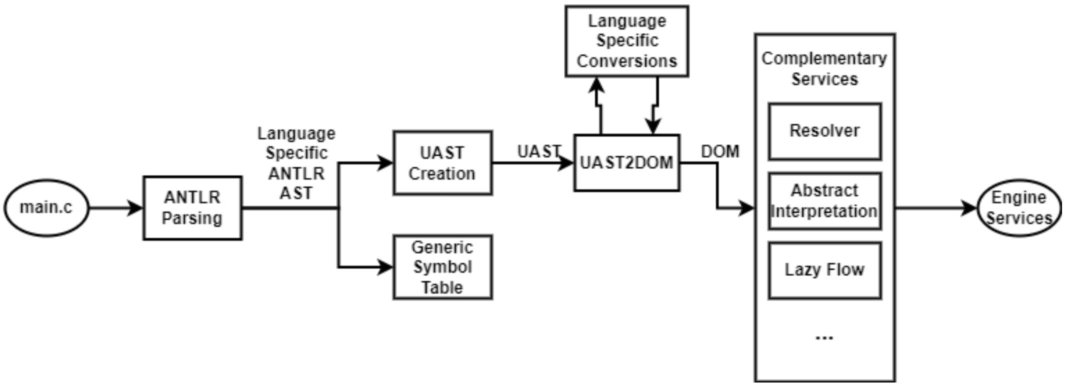


Figure 2 Checkmarx source code parsing pipeline.

Syntax Tree – UAST – as it is almost language independent). This tree is then converted to a Domain Object Model (DOM) tree, that is shared among all programming languages. The vulnerability scans run on top of this tree. A set of other services are then run on top of this model, namely a resolver, that finds internal references for symbols, an abstract interpretation engine, or even a lazy data-flow service. This structure is depicted in Figure 2.

Follows an elaborated explanation of the relevant steps involved in the type annotation process.

Symbol Table

The first stage of the pipeline is responsible for generating the symbol table. A set of visitors traverse the syntax tree constructed by the ANTLR grammar that gathers information about declared variables and literal values. Thus, some information on types is obtained at this point, as well as some type modifiers. Nevertheless, this information is not available for expressions, as no resolution of symbols is performed.

Listing 2 shows the type information collected by this stage that will be useful for the type annotation process. The example shows some information about an `unsigned long` variable. type associated with each variable contains, among other data, a list of modifiers and the name of the type. The list of modifiers will contain all the strings of the type that aren't the base type name, in this case, the two modifiers present are "unsigned", the signedness modifier of the type, and "long", the size modifier of the type. The `Modifiers` array can also contain strings like "static" and "volatile". The name of the type is "int" as that is the type declared implicitly for an `unsigned long` variable.

Listing 2 Information on the type and its modifiers for an `unsigned long` expression after the Type Inference stage

```
Type: {
    ...
    Modifiers: [ unsigned, long ],
    Name: int
    ...
}
```

UAST Creation

The parsing tree is traversed a second time using another set of visitors to create a Universal Abstract Symbol Table (UAST). This tree structure is shared among other languages, with minor language-specific details. Regarding types, the UAST does not add much information, it just compiles the data collected in the symbol table and adds it to the correct tree nodes.

UAST to DOM Conversion

The Domain Object Model (DOM) is a tree structure shared among all programming languages. This step processes the UAST in order to create this DOM structure. In the DOM, type information is much more clear and straightforward as can be seen in Listing 3. This Listing relates to the same data presented in Listing 2. At this stage the type information is saved in a class named "TypeRef", there are class variables which save each of the type-related modifiers instead of having them all on a single list.

Listing 3 DOM information on type modifiers for `unsigned long`

```
TypeRef: {
    TypeName: int,
    TypeSignedness : Unsigned,
    TypeSize : Long,
    ...
}
```

Resolver

The Resolver is one of the complementary services that uses the information available in the DOM to link all references, namely variables and functions with their definitions. This information is crucial for the type annotation process, as the return type of an expression that uses a variable requires the knowledge of that variable's type, as well as the return type of an expression that uses a function call, requires its return type.

4 Development

This section starts by discussing the different points, in Checkmark's engine pipeline, where the type annotation process could be implemented, and the challenges that arise from that approach. Follows the description of the implemented solution.

4.1 Solution Discussion

Three different stages were analyzed as possible implementation points for the type annotation system:

1. The first one was using the first pipeline step when the global symbol table is constructed. While this stage would allow easy implementation, the drawback is how specific is this stage accordingly with the language begin analyzed. This would mean that future improvements and the addition of type annotation on other languages would require the replication of similar code
2. The second approach was using the traversal done when converting the UAST into the DOM. The implementation of the type annotation step at this point would have little overhead, as the traversal through every tree node was already being performed either way.

This would, also, be a much better solution in terms of allowing for the generalization of different languages. The implementation would be a simple visitor for each node type in the DOM structure.
Unfortunately, the type of declared variables is not associated with the calls of said variables at the UAST2DOM stage, this means that no information on variable types is present. Thus, the type annotation implementation had to be moved further to the end of the pipeline.

3. There are complementary services that run on top of the final DOM structure. As already referred, one of those services is the Resolver. The Resolver is composed of rules, each one of them responsible for a different task. For example, one of them is responsible for the attribution of types to variables, functions and method calls. Adding a new rule at the end of all the other Resolver rules would allow access to all the information gathered by the previous steps. Thus, the information on the types would already be present and allow for the expression type annotation with all the information needed.

Taking into account the different approaches, the development was performed considering the addition of a new Resolver rule. It supports the infrastructure for type annotation for all languages, as the DOM structure that supports the type annotation engine is language-independent. The main challenge, which will be discussed in the next section, is to deal with different types from different languages, as well as the different type-inferring mechanisms that each one of these languages implements.

4.2 Type Annotation Implementation

As discussed before, the solution was implemented as a new rule in the Resolver engine. As expected, this approach requires a new traversal of all nodes in the DOM structure. However, given only expressions will have a type (statements do not have an inferred type), the solution is able to select only nodes that inherit from the expression node type, mitigating this problem to some extent.

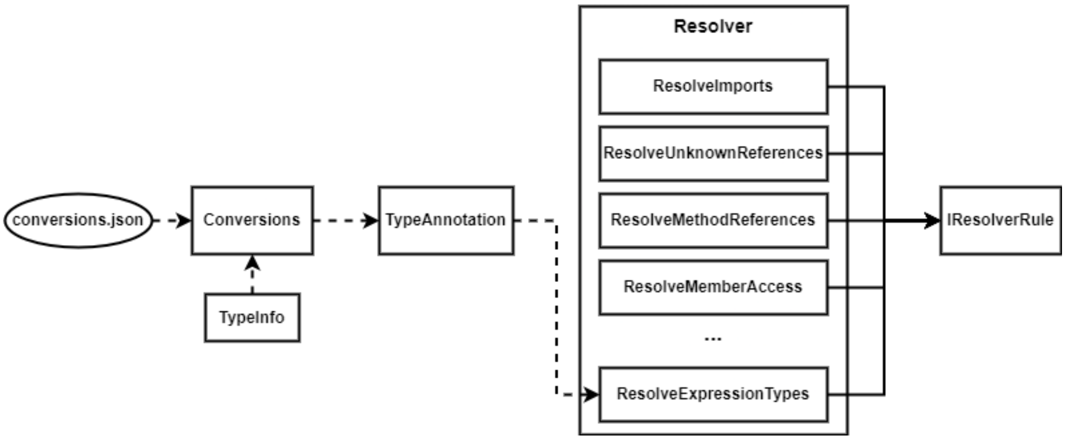


Figure 3 Resolver architecture.

Figure 3 shows that the Resolver is composed of various rules, each of them implementing the `IResolverRule` interface. This interface defines a main method that is called by the Resolver for each one of the rules. Along with this method, for the rule that is implemented, the constructor method is also be very important.

The constructor method is responsible for calling the `TypeAnnotation` class constructor and retrieving, from every node in the final tree, those of type *Expression*. These expressions are then processed by the mentioned `Execute` method, which iterates through the expressions and calls the `AddAnnotatedType` method.

The dotted lines simply denote files/classes that are used. The `TypeAnnotation` class, for example, is used inside the `ResolveExpressionTypes` rule.

4.2.1 The `TypeAnnotation` Class

The `TypeAnnotation` class goes through the different types of expressions (binary and unary operators, method invocations, casts, etc) and calls the `DetermineExpressionType` method.

This method is polymorphic, meaning that there is a different implementation for each expression type, thus making the code maintainable and easily extensible, and also recursive, as some expression types depend on the types of the child expressions. Thus, every time a node is visited, the process looks for the existence of the annotated type and returns if it exists. This is required given that the class constructs a list with all the expressions in the DOM structure.

The type annotation is stored as a custom attribute, meaning that no change was required in the base DOM object structure. To decide which type to assign to each expression another class, named `Conversions`, is used. It is responsible for deciding the type of a node given the child node types. For example, to decide on the type of a binary expression the method looks for the type of the left and right operands, to the operator.

4.2.2 The `Conversions` Class

The `Conversions` class is responsible for providing the `TypeAnnotation` class with the type resultant of different operand and operator combinations. It has two different stages that are equally important: a first stage, implemented in the class constructor, and a second one implemented in the `ResultOf` methods that are called by the `TypeAnnotation` class:

1. The constructor of this class uses JSON file, as shown on Figure 3, that describes, for each node type, operator or method, and operands, the inferred result type. This file content is loaded and a structure is created with the required information for dealing with the expression type annotation.
   This file uses a domain-specific language in order to reduce the number of repetitions. Details on this syntax are described in the next Section. For example, it is possible to specify that, for binary operators, all the combinations of operand types will always have the same result, regardless of the specific operator.
   In order to support different languages a new feature has been added, that will allow this main conversions file to be overridden, for specific node types. For example, considering the C programming language, the result of the division of two integer operands is, also, an integer, but in other languages, like Python, the result would be a floating-point value. So, for these exceptions, a Python-specific conversion file will be created that will just specify the main behaviour changes when compared with the base conversions file.
   The conversion information is stored on an instance of the `TypeInfo` structure. This structure is a simple type representation that contains basic type information: the type name (char, integer, etc.) and its signedness and size properties.
2. The `ResultOf` method uses this information. It receives all the operand types present and the operator (or other information like the name of the method being invoked), and the type of the expression that is being processed.
   It then iterates the map containing the corresponding expression type conversions and returns the `TypeRef` resultant of the received operands and operator combination.

4.2.3 The conversions JSON file

As explained previously, there is a JSON file that keeps all the combinations of operands, operators and nodes, as well as the inferred expression type for that node. The main focus of this file is to keep all this information in a simple and easily processable file, and also readable by the programmer.

Listing 4 Excerpt from `conversions.json`

```
"BinaryOperator": {
  "*": {
   "bool": {
    "default": "itself"
   },
   "char": {
    "Signed Short int": "Signed Short int",
    "Unsigned Short int": "Unsigned Short int",
    [...]
    "float": "float",
    "double": "double",
    "Long double": "Long double",
    "default": "char"
   },
   "Signed Long int": {
    "Unsigned Long int": "Unsigned Long int",
    "Signed LongLong int": "Signed LongLong int",
    [...]
    "Long double": "Long double",
    "default": "Signed Long int"
   },
   "Long double": {
    "default": "Long double"
   }
  }
 }
```

Consider the binary operator and the C programming language as an example. The resulting type for a binary operator expression is only dependent on the operands and not on the operator itself (except for the comparison operators, whose resulting type is a Boolean). To reduce the number of entries in the conversions file, the operator can be replaced by a special symbol: the asterisk (‘`*`’). Note that the operators are not represented by their symbol but by an internal name. Therefore, this asterisk will not conflict with the multiplication operator. Listing 4 shows a partial excerpt of the binary operator rules.

The C programming language always selects the more precise type from a binary operator. If the first operand is a Boolean value, then no matter what the second operand is: the inferred expression type will always be the other operand type. As another example, for long double values, there is no more precise type so the resulting type will always be a long double. The “default” keyword means any other type not specified in that rule, just like the `default` keyword in a `switch` statement.

There is also a “`itself`“ keyword that is used in conjunction with the “`default`” keyword. It means that the resulting type is always the type matching the *default* keyword.

5 C, C++ and C#

While the prototype started for the C and C++ programming languages, there was the need to guarantee that the approach is easily extensible for other programming languages. Before trying a dynamically typed language, where SAST will not be able to perform much

Listing 5 Excerpt from the `R10_03.cxql` query.

```
CxList assignExprs = Find_AssignExpr();
CxList paramDecls = Find_ParamDecl();
CxList decls = All.NewCxList(Find_Declarators(), paramDecls);
CxList unknownReferences = Find_UnknownReference();
CxList enumDecls = Find_Enum_Declarations();
CxList realLiterals = Find_RealLiterals();
CxList integerLiterals = Find_Integer_Literals();
CxList charLiterals = Find_CharLiteral();

CxList boolDecl = decls.FindByType("bool");
CxList charDecl = decls.FindByType("char");
CxList enumDecl = All.FindAllReferences(enumDecls) - enumDecls;
CxList shortDecl = decls.FindByTypeModifiers(TypeSizeModifiers.Short);
CxList intDecl = decls.FindByType("int")
         .FindByTypeModifiers(TypeSizeModifiers.Default);
CxList longDecl = decls.FindByTypeModifiers(TypeSizeModifiers.Long);
CxList floatDecl = decls.FindByTypes("float", "double", "long␣double");

// Get all references from essential types
CxList boolDeclRefs = unknownReferences.FindAllReferences(boolDecl);
[...]
// Find for integer literals with 'L' suffix
CxList literalsWithSuffix =
  integerLiterals.FindByRegex(@"[1-9]?[0-9]+[lL]?", false, false, false);
[...]

// Checks if the type of return statement references is different from
   ↪ the return type of the methodDecl
foreach(CxList retRefs in returnStmtRefs)
{
 CxList methodDeclReturn = retRefs.GetAncOfType<MethodDecl>();
 methodDeclReturn = methodDeclReturn.CxSelectDomProperty<MethodDecl>(
    ↪ method => method.ReturnType);

 CxList types = paramDecls.FindDefinition(retRefs);
 types = types.CxSelectDomProperty<ParamDecl>(parDecl => parDecl.Type);

 string retTypeName = types.CxSelectElementValue<TypeRef, string>(x => x.
    ↪ TypeName).FirstOrDefault();
 string methodDeclRetTypeName = methodDeclReturn.CxSelectElementValue<
    ↪ TypeRef, string>(x => x.TypeName).FirstOrDefault();

 if (methodDeclReturn != null && retTypeName != null &&
     retTypeName != methodDeclRetTypeName)
   result.Add(retRefs.GetFathers().FindByType<ReturnStmt>());
}
[...]

// Add to result non-addition assign expressions that have char
// on the left side and real, integer literals on the right side
addToRefs.Add(integerLiterals);
assignExprs -= assignAddExprs;
result.Add(getRefs(charDeclRefs, addToRefs, assignExprs));
```

inference at compile time, tests were performed for C#. While not too different from C, it has some relevant differences that allowed the understanding of possible limitations.

The implementation of the C# language was made simply by adding support for expressions that do not exist in C, such as the Try/Catch expressions, although this expression exists as a statement, it can never be used as an assignment to a variable in C/C++ contrary to C#.

C# also adds the Throw as an expression. Nevertheless, it works as a statement, as the evaluation of the current expression is interrupted and the exception is raised. Thus, its return type is irrelevant.

Another issue is the type system. C# does not support type modifiers. Instead, there are different types, one for each combination of signedness and size. For example, the type "`long`" in C# is the equivalent of a "`long int`" in C/C++. Thus, these new types need to be taken into account in the conversions file. As these types do not overlap with the C/C++ ones, they can be described in the same file without any issue.

6 CXQL Query exploration

Checkmarx uses queries written in a Domain Specific Language (DSL) based of C#, known as CxQL, to look up vulnerabilities in the DOM. Each query searches for a specific vulnerability in a specific language (although there are some exceptions).

Before the addition of expression type annotation queries that needed type information were hard to write. Consider Rule 10.3 from MISRA [2] (a group of safety guidelines for writing code for the automotive industries). It states that "*The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category.*"

This rule is currently implemented without any annotation of types. This query is quite large and not readable, as can be seen in Listing 5. The code is very repetitive, with more than 200 lines of code. It is partially replicating the type annotation process but in query time. Also, other queries rely on similar information and repeat most of this code to be able to infer expression types.

The code gets simpler by rewriting this query using the information provided by the type annotation system. Listing 6 shows the complete new query. It has less than 50 lines of code and is more maintainable. Testing the query in a project of 177 558 lines of code, we notice that the original query takes 8.17 seconds, while the new version takes 7.86 seconds. Even though it provides a speed improvement, it should be noted that the main gain of this query is to allow for more readable and understandable code.

7 Conclusion

This document presents the implementation of a POC for a type annotation system for Checkmarx's SAST engine. While the tool is still in a proof-of-concept stage, it has proven to be possible to extend to other statically typed languages and its usage in query development makes them more readable and maintainable. In fact, the type of code assessment that can be done with this new information is larger, allowing Checkmarx to develop new directions for vulnerability scans.

Currently, there are some main directions to improve this prototype and integrate it officially:

Listing 6 Rule `new_R10_03.xql`

```
var expressionsWithAnnotatedType = All.FindExpressionWithCustomAttribute(
    ↪ "AnnotatedType");
var assignments = All.FindByType<AssignExpr>();

var typeSizeModifiersRanking = new Dictionary<TypeSizeModifiers,int>{
   { TypeSizeModifiers.LongLong, 1 }, { TypeSizeModifiers.Long, 2  },
   { TypeSizeModifiers.Default,  3 }, { TypeSizeModifiers.Short, 4 }}

var typeNameRanking = new Dictionary<string,int> {
    {"double", 1}, {"float", 2}, {"int", 3}, {"char", 4}, {"bool", 5}};

foreach (CxList assign in assignments) {
 var leftTypeList = assign.CxSelectDomProperty<AssignExpr>(x => x.Left);
 var rightTypeList = assign.CxSelectDomProperty<AssignExpr>(x => x.Right)
     ↪ ;
 var assignExpr = All.NewCxList(leftTypeList, rightTypeList);

 if ((expressionsWithAnnotatedType * assignExpr).Count == 2) {

   TypeRef leftAnnotatedType =
      leftTypeList.CxSelectDomProperty<Expression>(x =>
        (x.CustomAttributes.First(attr => attr.Name == "AnnotatedType")
        .Parameters[0] as TypeOfExpr).Type).GetFirstGraph() as TypeRef;
   TypeRef rightAnnotatedType =
      rightTypeList.CxSelectDomProperty<Expression>(x =>
        (x.CustomAttributes.First(attr => attr.Name == "AnnotatedType")
        .Parameters[0] as TypeOfExpr).Type).GetFirstGraph() as TypeRef;
  try {
   if (
    typeSizeModifiersRanking[leftAnnotatedType.TypeSize] >
        ↪ typeSizeModifiersRanking[rightAnnotatedType.TypeSize]
    || (
    typeSizeModifiersRanking[leftAnnotatedType.TypeSize] ==
        ↪ typeSizeModifiersRanking[rightAnnotatedType.TypeSize]
    && typeNameRanking[leftAnnotatedType.ResolvedTypeName] >
        ↪ typeNameRanking[rightAnnotatedType.ResolvedTypeName]
    )
    || leftAnnotatedType.TypeSignedness != rightAnnotatedType.
        ↪ TypeSignedness)
   {
    result.Add(assign.Clone());
   }
  }
  catch (Exception e) {
   result.Add(assign.Clone());
  }
 }
}
```

- Explore new statically typed languages that are farther from C. As an example, Java or TypeScript are probable candidates.
- Attempt type annotation for dynamically typed languages, like Python, Perl, JavaScript or Ruby. We expect that many expressions will be annotated with a generic *any* type, but that some others might be possible to annotate properly. For some of these languages, like Python, code hints can be used in the source code to specify types. This information can be used for the type annotation process.
- Upgrade the proof-of-concept to handle more complex data types, record-like (classes, struct), arrays, pointers or objects. While some of these will prove to be a challenge, others will be simpler to support.

References

1 Acunetix. What is integer overflow? `https://www.acunetix.com/blog/web-security-zone/what-is-integer-overflow`. Accessed on May 26, 2023.

2 Motor Industry Software Reliability Association. *MISRA-C: 2012: Guidelines for the Use of the C Language in Critical Systems*. HORIBA MIRA, 2019.

3 Juan Caballero and Zhiqiang Lin. Type inference on executables. *ACM Comput. Surv.*, 48(4), May 2016. `doi:10.1145/2896499`.

4 Mingzhe Hu, Yu Zhang, Wenchao Huang, and Yan Xiong. Static type inference for foreign functions of python. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 423–433, 2021. `doi:10.1109/ISSRE52982.2021.00051`.

5 Leandro T. C. Melo, Rodrigo G. Ribeiro, Breno C. F. Guimarães, and Fernando Magno Quintão Pereira. Type inference for c: Applications to the static analysis of incomplete programs. *ACM Trans. Program. Lang. Syst.*, 42(3), November 2020. `doi:10.1145/3421472`.

6 MITRE. Cwe top 25 list (2022). `https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html`. Accessed on May 26, 2023.

7 David Alexander Molnar and David Wagner. Catchconv: Symbolic execution and run-time type inference for integer conversion errors. Technical Report UCB/EECS-2007-23, University of California at Berkeley, February 2007.

8 Nicholas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. *SIGPLAN Not.*, 42(6):89–100, June 2007. `doi:10.1145/1273442.1250746`.

9 Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer Berlin Heidelberg, 1999. `doi:10.1007/978-3-662-03811-6`.

10 Terence Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. The Pragmatic Bookshelf, Raleigh, 2007.

11 The LLVM Project. LLVM compiler infrastructure and tools. `https://llvm.org/`, accessed 2023.

12 Stuart M. Shieber. *Constraint-Based Grammar Formalisms: Parsing and Type Inference for Natural and Computer Languages*. MIT Press, Cambridge, MA, USA, 1992.

Characterization and Identification of Programming Languages

Júlio Alves ✉
ALGORITMI Research Centre/LASI, University of Minho, Braga, Portugal

Alvaro Costa Neto ✉
Federal Institute of Education, Science and Technology of São Paulo, Barretos, Brazil

Maria João Varanda Pereira ✉
Research Centre in Digitalization and Intelligent Robotics, Polythechnic Insitute of Bragança, Portugal

Pedro Rangel Henriques ✉
ALGORITMI Research Centre/LASI, University of Minho, Braga, Portugal

Abstract

This paper presents and discusses a research work whose main goal is to identify which characteristics influence the recognition and identification, by a programmer, of a programming language, specifically analysing a program source code and its linguistic style. In other words, the study that is described aims at answering the following questions: which grammatical elements – including lexical, syntactic, and semantic details – contribute the most for the characterization of a language? How many structural elements of a language may be modified without losing its identity? The long term objective of such research is to acquire new insights on the factors that can lead language engineers to design new programming languages that reduce the cognitive load of both learners and programmers. To elaborate on that subject, the paper starts with a brief explanation of programming languages fundamentals. Then, a list of the main syntactic characteristics of a set of programming languages, chosen for the study, is presented. Those characteristics outcome from the analysis we carried on at first phase of our project. To go deeper on the investigation we decided to collect and analyze the opinion of other programmers. So, the design of a survey to address that task is discussed. The answers obtained from the application of the questionnaire are analysed to present an overall picture of programming languages characteristics and their relative influence to their identification from the programmers' perspective.

2012 ACM Subject Classification Software and its engineering → Language types; Software and its engineering → Formal language definitions

Keywords and phrases Programming Languages, Programming Language Characterization, Programming Language Design, Programming Language Identification

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.13

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

Computers have evolved to be capable of recognizing and translating sentences, written according to formal rules, to machine code, which enables them to execute the tasks they are being asked to do. These sets of sentences are called *programming languages*. Every programming language has its own syntactic and semantic rules that make them unique [8], and that must be strictly followed in order to construct valid programs.

Each programming language has been developed with certain goals in mind, the so called programming paradigm, with some being being applied to specific fields of application, such as artificial intelligence or web development, while many others have been denominated as

© Júlio Alves, Alvaro Costa Neto, Maria João Varanda Pereira, and Pedro Rangel Henriques;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 13; pp. 13:1–13:13
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

general purpose languages. It can be said that behind the design of a programming language lies a philosophy of problem solving, usually suited for its intended area of application [9]. It is also evident that all languages have different characteristics, that can become obstacles not only to the development process, but also in teaching, sometimes creating such a mixture of diversified technologies applied to a system's construction that one could denominate as a *technological cocktail*. Due to each language's nature, it may be harder for programmers and students to understand how to properly apply the language rules to meet their needs [13]. The study of these differences and the identification of languages' main characteristics are not common practices in typical academia courses, making it a very important subject to be tackled.

It is then possible to raise some interesting questions that deserve further research: what indicates that two programming languages are different if what distinguishes them is not clear? What characterizes and identifies a language? Is it a syntax driven feature, such as the opening and closing symbols of code blocks? Could it be how a variable is declared? If a programming language is incrementally modified, at what point does it lose its identity?

This paper starts with a short section regarding fundamentals of programming languages, where it is explained how a programming language is specified and the role of its formal grammar. Section 3 presents the most relevant characteristics of a set of programming languages that have been elected for this study: `C`, `C++`, `C#`, `Java`, `Python` and `Haskell`. Section 4 discusses the design process of a survey, through which programmers were inquired about how they identified a programming language through several source code snippets. Section 5 presents the analysis of results collected using the referred survey. Finally, the paper concludes emphasizing what was discovered from the study so far conduced and proposes the potential improvements to be done through future research.

2 Fundamentals in Programming Languages

Programming languages are defined by sets of rules. These rules define both its *structure* – syntactic rules – and *meaning* – semantic rules. A *context-free grammar* is usually defined to generate a parser, that checks if a source code abides to these syntactic rules. The sentences derived from these rules are used to create programs that perform specific tasks, solve problems, and interact with other systems. A considerable number of programming languages has been developed, each one portraying its own syntax and features that make them more suitable to solve problems in a certain domain [2, 5].

As previously stated, the main way to define the syntax of a programming language is by using a grammar. A formal grammar is composed of a set of rules that must be used to derive correct, or valid, sentences of that language. In order to specify a grammar, four elements are needed:

- **Finite set of tokens** - The elementary symbols of the language defined by the grammar, usually called the Alphabet or Vocabulary;
- **Finite set of non-terminal symbols** - Represent the language concepts that give raise to, or derive, the sentences and sub-sentences;
- **Finite set of grammar rules** - Substitution rules in which a non-terminal symbol (on the left) may be substituted by a sequence of terminals and non-terminals defined on the right side of the derivation operator;
- **Start non-terminal symbol** - The starting point for the derivation process.

Despite this standard way of designing programming languages, the substitution rules and the concepts expressed by each non-terminal symbol provide distinct characteristics to each language, establishing a sense of identity. This sense may also be realised through the use and application of each language, which allows for their characterization beyond formal definitions.

3 Characterization of Programming Languages

As it is known by programmers, every programming language is different. These differences may be on their purpose, the paradigms they realize, features (such as portability or efficiency) or, and most notably, their lexicon and syntax. Programming languages may be divided in two groups that are defined by their programming paradigms: *declarative* and *imperative*. The former includes all the languages that are based on a set of declarations or specifications of several program elements, while the latter is based on a sequence of instructions, or orders. As can be observed in the following sections, out of all the selected languages the only language under study that belongs to the declarative paradigm is `Haskell`. In order to better understand these differences from the programmers' points of view and construct a well founded survey, it was important to collect the main features that define each languages identity, gathering the syntactic and semantic characteristics that are either uniquely or commonly recognizable to each of them. In this way, it became easier to determine which questions to ask and how to ask them, as will be discussed in Section 4.

The main reasons for choosing these languages were:

- General similarity (`C` *versus* `C++`, `Java` *versus* `C#`);
- General dissimilarity (`C` *versus* `Python` and `Haskell`);
- Variation of paradigms;
- Ascendancy (`C` *versus* `C++`, `C#`, and `Java`);
- Possible familiarity of the respondents.

It is important to note that these reasons were meant to compare similarities and dissimilarities, in order to raise or lower the linguistic contrast of the snippets in the questionnaire.

3.1 C

Being one of the most used and recognizable programming languages in history, `C` was an important entry point for later comparison in the survey. Many other languages, such as `C++`, derived their syntactic and semantic rules from those previously found in `C`. Some of the defining characteristics of `C` are [14]:

- `C` is a Procedural Programming language;
- `C` is strong and statically typed;
- A mandatory `main` function that defines the entry point for execution;
- Curly brackets are used to enclose the contents of a code block;
- Semi-colons indicate the end of a statement;
- Variables and constants must be explicitly declared using the `type identifier` pattern;
- Functions are the structural elements of the source code;
- Function signatures must have a return type, followed by its name and a pair of parenthesis grouping all arguments;
- Functions may return nothing (`void`);

- The fixed-length array is the only data collection built into the language and its declaration follows the basic variable declaration pattern. Lengths and dimensions are defined using square brackets after the identifier;
- Pointers are memory addresses and are used to indirectly access memory addresses;
- A pointer is declared like any other variable, with an added asterisk prefix before the identifier;
- An ampersand is used to fetch the address of a variable;
- Characteristic and recognizable functions are present in the standard library, such as `printf` and `scanf`.

3.2 C++

`C++` was designed as a *super set language* of `C`, inheriting a lot of its syntactic and semantic characteristics. The concept of classes is introduced in `C++`, which brings `C`'s structure and form to a new paradigm. Some of its main characteristics are [15]:

- Just like `C`, `C++` is also strong and statically typed;
- `C++` imported all the basic syntactic and semantic elements from `C`. Therefore, characteristics such as the entry point for execution, code block notation, declaration syntax and so on were either unaltered or simply augmented;
- The concept of classes was the most structural aspect that `C++` implemented on top of `C`, defining an Object Oriented Programing (OOP) language;
- Classes are also structural elements of the source code;
- Classes are declared using the `class` keyword, in a similar fashion to composite structures (`struct`) in `C`;
- Data members are declared inside the class definition using the standard variable declaration pattern;
- Methods may be declared either inside the class definition or outside, using a specific notation (`class::method`);
- The standard library added several data structures, such as dynamic vectors, linked lists, and queues, while also provided several new redundant methods and operators that implemented the same functionality as `C`'s standard library – « instead of `printf`, » instead of `scanf`, and so on;
- Some other additions to `C`'s mechanisms are also present, such as operator overloading, `new` and `delete` commands, templates, virtual functions *etc.*;
- In `C++`, the notion of *namespaces* is introduced. A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it.

3.3 C#

Just as `C++` was designed to be an evolution of `C`, `C#` was meant to pursue this path further concerning `C++`, while also competing in the same space of `Java`. `C#` simplifies many of `C++`'s features and solves some of its problems (such as pointers and memory allocation issues) while maintaining its power and paradigm. Some of `C#` characteristics are [1]:

- Like its predecessors, `C#` is also strong and statically typed;
- The need of a `Main` method, instead of a function;
- As a successor of `C++`, it maintains the same syntatic features to delimit code blocks and to end statements, both carried over from `C`;
- When declaring a variable, one can explicitly state what the variable type will be or can use the keyword `var`, and the variable type will be inferred by the compiler;

- As `C#` is an Object Oriented Programming language, classes are the structural elements;
- `C#` offers a variety of data structures, such as arrays, lists, stacks, queues, sets and dictionaries;
- Directives indicate that a specific namespace will be used. It is composed by the keyword `using`, followed by the desired namespace identifier;
- References types supplant pointers and remove the asterisk notation.

3.4 Java

`Java` is an Object Oriented Programming language that, despite not being based on `C` and `C++`, certainly was influenced by them. Some of `Java` characteristics are [6]:

- `Java` is a strong and statically typed language;
- As `Java` was inspired by `C` and `C++`, many syntactic and semantic elements have been imported into it, such as the mandatory entry point (`main` method), curly brackets to delimit code blocks, semi-colons to end statements and the standard variable declaration pattern;
- All variables must be declared with a type;
- There are specific declaration modifiers, such as `final`;
- Just like `C#`, classes are the structural element;
- `Java` also offers a significant variety of data structures, such as arrays, lists, stacks, queues, sets and maps;
- Types can be divided into two semantic categories: primitive types, such as `boolean` and numeric types; and reference types such as classes, interfaces and arrays. Values in reference types relate to objects;
- The special type `null` has no name and can be assigned or cast to any reference type. Its reference is the only possible value of an expression of type `null`;
- Conversions, such as identity conversion, widening and narrowing primitive conversion, unchecked conversion and capture conversion;
- In order to use other packages, one must use the keyword `import` followed by the name of the package, similar to `C++` and `C#` namespaces.

3.5 Python

`Python` is an Interpreted, Scripting and Procedural language but also supports other programming paradigms such as Object Oriented and Functional. Some of `Python` characteristics are [10]:

- Unlike the previously discussed languages, `Python` is dynamically typed;
- A `main` function or method is not required;
- In `Python`, instead of using braces to delimit code blocks and scope, these are defined by indentation;
- Semi-colons are not required to mark the end of a statement, a new line is enough;
- Variables are declared when a value is assigned to them, without needing to declare their type;
- In order to define a function, its name must be preceded by the keyword `def`. After the function name, a pair of parenthesis must be used containing and optional a set of arguments;
- Despite being an OOP language, `Python`'s structural elements are functions;
- There are different data structures like lists, tuples, sets, dictionaries, strings and range available in the standard library, some of which have special syntax associated with their use.

3.6 Haskell

Differently from all other languages previously discussed, `Haskell` is a purely Functional Programming language meaning, instead of telling the computer what to do, the programmer tells the computer what something is. `Haskell` is also lazily executed, so the interpreter will compute values only when they are actually needed, unless told otherwise. Some of `Haskell` characteristics are [11]:

- `Haskell` is a Declarative language since is based on the specification of a set of functions and expressions;
- `Haskell` is a strong and statically typed language;
- Code blocks are delimited by indentation, just like `Python`;
- A new line is used to define the end of a statement;
- In order to declare a variable in `Haskell`, the programmer must use either the keyword `let` or `var`, followed by the variable name. Variables are immutable by default, however the keyword `var` allows the variable to be mutable;
- Due to `Haskell`'s inference mechanism, it is not needed to specify variables types in declarations;
- In order to declare a function type, the programmer declares its name followed by the `::` operator – which can be read as "type of" – and all its parameters. The parameters are separated by the arrow operator (`->`), with the last parameter being the return type;
- Functions are the main structural element of the language;
- The data structures offered by `Haskell` are: lists, tuples, sets, maps, strings and ranges;
- Instead of requiring multiple lines of code to write nested `if...then...else` conditions, `Haskell` implements *guards*. Guards are indicated by a pipe character (`|`) followed by a boolean expression and what will be evaluated in case the expression is true. Should it be false, the next guard is evaluated, and so on;
- `where` - `Haskell` doesn't allow to store variables for a future use. However, with the keyword `where`, users can declare variables to be used inside a function with its biggest limitation being the fact that `where` scope is limited to the function it was declared;
- `Haskell` also has a `switch` like statement, with the use of the keyword `case` in the format `case <expression> of <pattern> -> <result>`;
- Recursion plays a big role on the regular use of the language.

4 Survey Design

A crucial part of this research was the development and implementation of a survey[1] to gather the different perspectives of a programmer regarding their known languages identification. While not directly applied, the general concepts behind Value-Focused Thinking [7] founded the rationale throughout the construction of the survey, as was the case with previous studies [4].

Since the survey contemplated multiple programming languages and the target audience had various degrees of knowledge of these languages, it became critical to weigh one's answers. For this reason, the survey started with a section containing only one question on the respondent's familiarity with the six chosen programming languages (`C`, `C++`, `C#`, `Java`, `Python` and `Haskell`). Respondents could choose his or her level of knowledge in each language from five different options, ranging from complete ignorance to profound knowledge.

---

[1] Available at `https://forms.gle/6kBuHhYD5FPHK8Cp9`.

From this section forward, all questions regarded different code snippets written in the programming languages under study. These code snippets were created aiming for conciseness, while also contrasting or emphasizing specific linguistic traits. As previously stated in section 3, both similarities and dissimilarities were implemented when two snippets were compared. This was made to evaluate not only how minor details influence the differentiation of similar languages, but also to identify which linguistic elements have stronger identification roles, raising similarity between very different languages.

The second section aimed to evaluate if the respondent could identify the language of a snippet of code, justifying his or her answer. There were six pairs of questions (“What is the language present in the following snippet?” and “Justify your rationale for the previous answer.”), one for each language. The purpose of this section was to understand the reasoning behind the identification of a programming language and to detect if multiple people use the same thought process, specifically if the snippet contains a distinctive feature that is commonly reported as important for the identification. Using a similar strategy as applied to the previous section, for each language the respondent could choose one of four possible answers: doesn’t know, it sure isn’t, maybe and absolutely is. Listing 1 presents the snippet used for the question regarding the C language identification.

Listing 1 C language snippet used in the second section of the survey.

```
int main()
{
    if( n <= 1 || (n = atoi(argv[1])) <= 0 ) n = 8;
    int hist[n];
    solve(n, 0, hist);
}
```

For the third section, the identification of programming languages was evaluated through comparisons. In this section, the respondent was presented with two code snippets representing solutions to the same problem, each written in a different language, and asked to identify which programming languages were used in each snippet. Listings 2 and 3 exemplify the comparisons that were made in each of the third section’s questions. The similarities in general form were intentional to pinpoint characteristics that respondents justified as crucial to the identification of each language – or his or her inability to do so.

Listing 2 Java snippet used in the third section of the survey. This snippet was presented in the same question as listing 3 for comparison.

```
class FileIOTest {
    public static void main(String[] args) throws Exception {
        var lines = Files.readAllLines(Paths.get("input.txt"));
        Files.write(Paths.get("output.txt"),lines);
    }
}
```

Listing 3 C# snippet used for comparison with the one presented in listing 2, in the third section of the survey.

```
class FileIOTest
{
    public static void Main(string[] args)
    {
        var lines = File.ReadLines.("input.txt");
        File.WriteAllLines("output.txt",lines);
    }
}
```

In the fourth and last section, a new approach was applied. This new approach consisted in progressively modifying a given a code snippet in one programming language until it became a substantially different. The respondent was then asked at which point of these progressive changes he or she believed that the snippet of code no longer resembled the original language, that is, at which point the language lost its identity. Figure 1 shows one of the original snippets, followed by their progressively altered versions. The changes were small and related to one of the language's main characteristics at each version. This section aimed to establish what is the *breaking-point* to a language's identification, meaning which characteristics are so entrenched into its definition that once it is changed, programmers can no longer associate code with it.

```
int main(int argc, char *argv[])
{
    char t[255]="alphaBETA";
    str_toupper(t);
    printf("uppercase:␣%s\n",t);
    str_tolower(t);
    printf("lowercase:␣%s\n",t);
    return 0;
}
```

```
int main(int argc, string argv)
{
    string t="alphaBETA";
    str_toupper(t);
    printf("uppercase:␣%s\n",t);
    str_tolower(t);
    printf("lowercase:␣%s\n",t);
    return 0;
}
```

```
int main(int argc, string argv)
{
    string t="alphaBETA";
    str_toupper(t);
    write("uppercase:␣",t);
    str_tolower(t);
    write("lowercase:␣",t);
    return 0;
}
```

```
main(argc, argv)
{
    t="alphaBETA";
    str_toupper(t);
    printf("uppercase:␣",t);
    str_tolower(t);
    printf("lowercase:␣",t);
    return 0;
}
```

Figure 1 Four progressively altered snippets of C code used to establish what is the breaking point for the identification of a programming language.

5 Survey Results Analysis

To start the results analysis, it is important to begin with the first question, as it shows how well the respondents knew the six chosen languages. `Java`,`C` and `Python` were the most well known languages, with 95%, 88% and 93%, respectively, of the respondents stating to be, at least, capable of using the languages. These results are indicative that questions regarding `Java`,`C` and `Python` can give a higher understanding on what identifies these languages. On the other hand, `Haskell`, `C++` and `C#` respectively had only 47%, 45% and 31% of respondents stating to be, at least, capable of using the languages.

5.1 Second Section: Language Identification

Regarding `C`, respondents successfully identified it, stating the main reasons to its identification were:

- Variable declaration syntax;
- Use of pointers;
- Syntax features like semicolons, brackets to delimit code blocks and functions signature;

Like `C`, `Java` was also correctly identified, with the main reasons being:

- Methods access modifiers;
- Methods signatures;
- Variables types;
- The usage of `System.out.println`;

On the third snippet shown, the language present was `C#`, however, as the respondents weren't very familiarized with this language, there was a consistent lack of confidence to distinguish whether the language was `Java` or `C#`. Out of the total, nine people said they were sure it was `Java`, and nine said they were sure it was `C#`. On the other hand, twenty four respondents said it could be `Java`, and twenty one said it could be `C#`. There were also fourteen respondents who believed it could be `C++`. It was unanimous, however, that it was neither `Python` nor `C`. The audience revealed a big difficulty in finding characteristics which allowed them to distinguish between `Java` and `C#`. The respondents who could successfully identify it as `C#`, expressed that what made them sure was the use of PascalCase naming convention on identifiers. Since it represents a writing style other than a language characteristic, it is safe to say that the identification wasn't a consequence of the language's formal definition, but its conventions of use instead.

The fourth snippet consisted of a piece of `C++` code. It was very obvious for the audience that this snippet wasn't `C`, `Java` or `Python`. There was some uncertainty if it could be `C#` as the respondents could identify this language as being a part of the `C` language's branch of derived languages. Nonetheless, there was a major consistency as identifying the language as being `C++` due to:

- The usage of the `std` library;
- `Unsigned long long` type;
- The method or member function call notation.

On the fifth snippet shown, the language present was `Python`. It was very evident for the audience that this was `Python` as no other language was given a positive answer and `Python` was picked as the right language by almost one hundred percent of the users. What made them answer `Python` was:

- Block syntax and indentation;
- The use of `def` when defining a function;
- Type inference;

For the last snippet on this section, respondents were finally presented with a `Haskell` code snippet. Once again, the respondents were certain it couldn't be any other language other than `Haskell`. The reasons given for this were:

- Functional programming style;
- Point free syntax, a style of writing Haskell code that avoids explicit mention of the arguments of a function;
- Function signatures;
- The complete overall difference to the other languages;

5.2 Third Section: Language Comparison

5.2.1 `Java` vs `C#`

As `Java` and `C#` were two languages which it was expected some difficulties to arise, two comparisons were made. On the first comparison, the respondents were able to successfully identify the languages correctly, stating – as they did on the previous section – the difference in naming convention as the biggest contributor for the identification. Respondents also pointed the way exceptions are thrown in `Java` as a differentiating factor. On the second comparison however, there was a big uncertainty on the `C#` snippet, with respondents being mainly divided between `C#`, `C++` and `Java`. There were also some answers that were open to the possibility of it being `C` or even another language. This indecision is due to respondents not being able to associate the types `ulong` and `uint` to a language. `Java`'s identification was very straightforward to the respondents, stating that the use of the keyword `final` on an argument as the reason the snippet was written in `Java`.

5.2.2 `C` vs `C++`

In this comparison, the respondents were on the same page, with virtually no problems to the correct identification. However, there were only two reasons on why they believed one language was `C` and the other `C++`: the functions used to output something (`printf` for `C` and `cout` for `C++`) and the use of `namespace` on `C++`.

5.2.3 `C` vs `Java`

Once again, there was no doubt within the respondents on what language was present in each snippet. Respondents stated that the output functions `printf` and `System.out.println` were the main reasons for their correct identification. There were also a few answers pointing that the way arrays are declared was also a contributing factor.

5.3 Fourth Section: Language Identification Breaking-point

For `C`, respondents were adamant that from the second snippet onward the language couldn't be `C`. The modification that triggered this opinion was the replacement of `char *` with a new type `string`, commonly used in other languages.

With `C++`, the answers were also unanimous on saying that, from the second snippet on, the language could not be `C++`. The modification on how libraries are imported, switching from `namespace` and `#include <libraryName>` to `with` was the defining change to arrive at the breaking-point.

On the third question, respondents agreed that the first snippet was `C#` and were also open to the possibility of it on the second snippet. It is possible to infer that the output function is not a very defining characteristic of the language, as that was the modification made from the first to the second snippet, changing `System.Console.WriteLine()` to `write()`. However, on the third snippet it became obvious to the respondents that the language wasn't `C#`. On this snippet, the way code blocks are structured was modified, replacing brackets with indentation, such as in `Python`.

Curiously, the exact same reaction was obtained with `Java`. On the second snippet, the code blocks syntax was modified from curly brackets to indentation, triggering the respondents to not identify the language as being `Java`.

With `Haskell` and `Python`, respondents were open to the possibility of being the original languages until the function declaration syntax were modified. In `Haskell`'s case, the modification was the replacement of the standard syntax `functionName :: argument -> argument -> result` with `functionName(argument, argument)`. `Python`'s modification was even more subtle, with the mere change of the keyword `def` to `function`.

The analysis of the results was obtained through the answers gathered from a class of fourth year software engineering students. A select group of people, that included university professors and software engineers, was also invited to answer, however, the survey has not been yet made publicly available, as these results and the students' feedback will be used to make the necessary improvements for a future version.

The respondents were selected for their background in computer programming languages. The majority of the respondents were Master's Degree students present in a course of Languages Engineering at University of Minho (UMinho). In total, 44 people responded to the questionnaire, 39 of which were students. The other 5 people were professors of the Department of Informatics, also at UMinho.

6 Conclusion

Machine Learning [12, 16] has been extensively used on the subject of computer programming languages automatic identification. These studies applied different types of classifications such as Image Based Classification [3], Algorithmic Classification [9] and Source Code Classification [8, 17], however not many studies have been conducted to consider the programmers' points of view. What are the most relevant linguistic features that programmers observe, consciously or not, to correctly identify a language used in a source code? What and how much can a language's syntax and semantics change and still retain its identity? In order to answer these questions, the most straightforward solution is to actually ask people that deal with programming languages. The challenge then becomes not only what to inquire programmers about their perception of a language, how it is recognized and identified, but also *how to properly ask them.*

This paper presented an analysis of typical linguistic features, commonly found in six established programming languages (`C`, `C++`, `C#`, `Java`, `Python`, and `Haskell`) followed by the design and application of a survey that seeks to better understand the intrinsic relationship between programmers and their main tools of the trade: programming languages.

The survey was designed around three main approaches:

1. Direct identification of a programming language;
2. Comparison between similar languages;
3. Determination of the identity breaking-point.

The three approaches were implemented via corresponding sections in the survey. The questions for approaches 1 and 2 were constructed in pairs: a multiple-choice direct question, asking respondents to identify the language in the respective snippets of code; followed by an open-ended justification. The last section (approach 3) used only multiple-choice direct questions, gradually changing snippets of source code to evaluate at which point the original language (shown at the first snippet of each question) lost its identity.

The common threads of identification in the answers showed that some typically associated syntactic features, such as code block syntax, method signatures and variable declaration patterns are contributing factors to identify a programming language – which was expected. Other answers lead to some unpredicted results, pointing to the use of standard library elements (functions, classes, methods, etc.) and specially the naming convention that is typical – and sometimes, mandatory – in several programming languages. While not a characteristic of the formal definition of the language, it is nonetheless prevalent and linguistically related to it.

The comparison between languages pointed the inevitable mixed results between languages that are very similar and descendants from a common parent – `C#` and `Java` being the most prominent example, both derived from `C` and `C++`. The two main differentiating features, as pointed by the respondents, were naming convention and typical standard library elements, once more.

On the other hand, the results were not very conclusive on the breaking-point analysis, since there is an argument to be made for changing the way the questions were asked. Most respondents correctly identified the first snippet as being the original and unaltered language, and the second one forward as promptly loosing its identity. Since the changes were diminutive and triggered the breaking-point almost immediately, it is inconclusive as to which ones contributed the most to the lost of identity.

The feedback was one of the goals of this first application, as respondents pointed to two main difficulties they faced when answering the survey: they felt questions related to languages they didn't know should not be present and, as pointed in the previous paragraph, the breaking-point analysis must change in form, avoiding the immediate lost of identity. The latter showed us that a different approach to the breaking-point analysis must be tackled for a next version, aiming to gather enough information towards a more conclusive argument.

The results already showed interesting facets about the relationship between programmers and the languages they know and use. In a setting of multiple different technologies being applied to the construction of a modern computer system – a *technological cocktail* – this naturally occurring change of patterns and identities in the development process might be cumbersome. If well understood, this cognitive load may be lightened by diminishing the changes in language identity or even choosing technologies that use languages with similar identity features. Not only that but, at an initial stage of a programmer study, it can also aid a programmer to learn a programming language with more ease, given that it will allow him to choose a programming language more suited to his preferences.

This research is part of a larger group of studies that aim to identify and understand the human relations involved in computer programming. Future works include the review and refactoring of the survey, specially in the last section, followed by the next version to be openly applied for general public response. Finally, a guiding system will be implemented for the detection of a language's identity in snippets of source code, based on the programmers' points of view.

References

1 Sam A Abolrous. *Learn C-Sharp - Includes the C-Sharp 3.0 Features*. Wordware Pub, 2007.

2 Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques & tools*. Addison Wesley, 2007.

3 Francesca Del Bonifro, Maurizio Gabbrielli, Antonio Lategano, and Stefano Zacchiroli. Image-based many-language programming language identification. *PeerJ Computer Science*, 7, 2021.

4 Alvaro Costa Neto, Cristiana Araújo, Maria João Varanda Pereira, and Pedro Rangel Henriques. Value-Focused Investigation into Programming Languages Affinity. In Alberto Simões and João Carlos Silva, editors, *Third International Computer Programming Education Conference (ICPEC 2022)*, volume 102 of *Open Access Series in Informatics (OASIcs)*, pages 1:1–1:12, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ICPEC.2022.1`.

5 Robert W. Floyd. The syntax of programming languages, 1964.

6 James Gosling, William N. Joy, and Guy L. Steele. The java language specification, 1996.

7 Ralph L. Keeney. Value-focused thinking: Identifying decision opportunities and creating alternatives. *European Journal of Operational Research*, 92(3):537–549, 1996. `doi:10.1016/0377-2217(96)00004-5`.

8 Jyotiska Nath Khasnabish, Mitali Sodhi, Jayati Deshmukh, and Gopalakrishnan Srinivasaraghavan. Detecting programming language from source code using bayesian learning techniques. In *MLDM*, 2014.

9 David Klein, Kyle Murray, and Simon Weber. Algorithmic programming language identification. *ArXiv*, abs/1106.4064, 2011.

10 Dave Kuhlman. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. Platypus Global Media, 2015.

11 Miran Lipovaca. *Learn You a Haskell for Great Good! A Beginner's Guide*. No Starch Press, 2011.

12 Tom Mitchell. Machine learning, 1997.

13 Alvaro Costa Neto, Cristiana Araújo, Maria João Varanda Pereira, and Pedro Rangel Henriques. Programmers' affinity to languages. In *ICPEC*, 2021.

14 Easy Programming. *C Programming Language The Ultimate Beginner's Guide*. CreateSpace Independent Publishing Platform, 2016.

15 Bjarne Stroustrup. *A History of C++: 1979–1991*, pages 699–769. Association for Computing Machinery, New York, NY, USA, 1996.

16 Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 3rd edition, 2010.

17 Shaul Zevin and Catherine Holzem. Machine learning based source code classification using syntax oriented features. *ArXiv*, abs/1703.07638, 2017.

Towards a Universal and Interoperable Scientific Data Model

João Oliveira
University of Minho, Guimarães, Portugal

Diogo Gomes
University of Minho, Guimarães, Portugal

Francisca Santana
University of Minho, Guimarães, Portugal

Jorge Oliveira e Sá
Algoritmi Centre, University of Minho, Guimarães, Portugal

Filipe Portela ✉
Algoritmi Centre, University of Minho, Guimarães, Portugal
IOTECH - Innovation on Technology, Trofa, Portugal

Abstract

The growing number of researchers in Portugal has intensified the appearance of several scientific platforms that allow the indexation of publications and the management of scientific profiles. The diversity and high number of platforms brings problems at the level of crossover and integrity of the information, i.e., the researchers' profiles are rarely updated, and their data are not properly grouped and cross-referenced. Hence, the need arises for a more global platform that enables the synchronization of information, free from constraints imposed by existing data.The study and work carried out aims to solve this problem by creating a robust and interoperable platform based on an innovative library merge algorithm. Thus, this platform includes information regarding publications, researchers and scientific indicators, by crossing and grouping data from several platforms.

2012 ACM Subject Classification Software and its engineering → Development frameworks and environments

Keywords and phrases RDProfile, Researchers, Scientific Platforms, Scientific Data

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.14

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

Nowadays, the number of existing scientific data, libraries, metrics, and indicators (KPIs) is enormous and makes it difficult to manage them by researchers and scientific institutions. The number of researchers has also increased, which makes it even more difficult to access correct and updated information on multiple platforms. Therefore it is vital to collect and combine scientific data from several platforms to answer the institution's requests and improve the quality of project applications by enhancing the researcher's data. To this end, it is necessary to create an algorithm for merging libraries, as part of the RDProfile project – a project that aims to create an inclusive, reliable, and scalable solution that can easily include diverse information from the profiles of researchers.

In the scope of this study, the following research question can be identified throughout the article: How can the data from profiles, quartiles, and indexes from various platforms be conciliated?

© João Oliveira, Diogo Gomes, Francisca Santana, Jorge Oliveira e Sá, and Filipe Portela;
licensed under Creative Commons License CC-BY 4.0

12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 14; pp. 14:1–14:16
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The main objective is to develop an artifact that includes a prototype of a web solution capable of grouping and cross-referencing data from researchers from various sources including Scopus, Web Of Science, and ORCID. To accomplish this goal, it is necessary to create secondary objectives such as developing a responsive interface with pervasive features, developing an API that allows communication with some of the platforms, integrating several APIs, and optimizing RDProfile platform with three new features (profiles, quartiles, and index).

This paper is divided into six sections: Introduction, Background, Material and Method, RDProfile Algorithm, Proof of Concept, and Conclusion. In the first chapter, the theme of the paper is contextualized, as well as its objectives and research question are defined. The most important concepts are described in Background. The Scrum and DSR methodologies are covered in the next chapter – Material and Method – as well as the used tools and metrics, and the three scientific platforms involved in the work, namely, ORCID, Scopus and Web Of Science. In the fourth chapter, RDProfile Algorithm, the architecture and structure of the RDProfile platform is described, with focus on the Middleware (API), where the developed library merging algorithm fits. In Proof of Concept concrete cases are presented, using real data from a Portuguese research institution and finally, in the last chapter, Conclusion, final considerations are made and future work is described.

2 Background

In this section the most important concepts are covered, in order to better understand the background of this paper. Thus, the project to which this work belongs is described, as well as the general concept of scientific platforms and the related work.

2.1 RDProfile

RDProfile is a complete and interoperable platform that allows grouping and cross-referencing researchers' data from several sources, such as Scopus, ORCID and Web Of Science. The goal is to provide users with correct and updated data of researchers and institutions, acting as an aggregator of dispersed knowledge, publicly available and easily accessible. Besides being an inclusive, reliable and scalable solution, RDProfile also intends, through gamification, to help identify the best scientific profiles and improve indicators that allow increasing the number of R&D projects funded based on the improvement of the quality of researchers' metrics [10].

The RDProfile Project started in 2020 and has three phases [10]. The first phase – Research and support the project idea – resulted in an understanding of the state of the art and the definition of the requirements for the RDProfile, providing a clear vision of the added value it adds. The second phase – Explore a possible solution to solve the problem – will result in the creation of a library merging algorithm that will solve the problems that currently exist, such as incorrect identification of references. Afterward, a web solution will be implemented, which will have innovative and useful functions for scientific researchers, such bibliometric indicators about researchers and institutions, metrics about projects and reviews, filters that allow easy cross-referencing, monitoring of citations lists with indexations, among others.

In the third phase, the artefact will be optimized and made universally available. Here the focus will be on ensuring scalability and modularity so that the solution can be accessed by any institution by Interface or API.

2.2 Scientific Platforms

The expansion of Internet has also expanded the number of digital platforms with similar objectives, such as Web Of Science and Scopus, which have been the two most widely used databases for bibliometric analyses [13]. We can divide these platforms into two groups. The indexing platforms allow the indexing and cataloging of articles in a scientific database, and the aggregating platforms, which aggregate various data in order to offer the user more information about publications, researchers and institutions. RDProfile, as already mentioned, is an aggregator since its goal includes making it easier and more efficient to search for researchers and institutions, providing the majority of data in one place.

According to previous study conducted as part of the RDProfile project [2], the most relevant scientific platforms are Scopus, Clarivate Web Of Science, ORCID, Authenticus, Google Scholar, Crossref, Scholarpedia, Semantic Scholar and Dimensions. However, the work will focus on the first four due to their similarity to the RDProfile platform.

Overall, these platforms are commonly used and recognized, but they have limitations or errors, e.g., few features/data, non-matching data, missing or wrong metrics, wrong data grouping, missing citations, and so on witch prevents institutions from getting accurate data [10]. Of the four platforms, Authenticus and ORCID are aggregators, like RDProfile. Authenticus combines the data of other platforms like Scopus and Web Of Science, but it is nationally focused, not scalable, and lacks a responsive, user-centred interface. ORCID has become increasingly popular and has recorded a steady growth in ORCID registrations [1]. This platform allows looking at how individual researchers interact and produce their work. Unlike the others platforms, the researcher becomes the center of the analysis, shifting attention from merely counting publications and citations to a much richer perspective related to the scientific workforce and its internal dynamics [5]. Web Of Science and Scopus stand out as the two most traditional sources of scientometric data [5]. Scopus is often considered as one of the largest curated databases [12]. Although, it does not cover all publications, nor does it expose all metrics of either publications or authors. The same happens with Web Of Science that remains relevant for the scientific area [12]. The combination of those three relevant platforms can provide a good solution to the problem identified in this paper. The more platforms a solution covers, the more complete it becomes, however, in this stage, only these three will be considered, to avoid increasing unnecessarily complexity.

2.3 Related Work

Within the range of platforms previously identified as similar to RDProfile, a study “A Benchmarking Study of the Scientific Platforms”[3] was conducted to Authenticus platform for being the most similar platform.

Authenticus is a database of scientific publications authored by researchers from Portuguese institutions. This database aggregates publications from different scientific platforms, such as Scopus, Google Scholar, Clarivate Web Of Science, Crossref, DBLP and ORCID. Its operation is based mainly on an algorithm capable of identifying the name of an author in a publication and associating it correctly with a particular researcher in the database. Subsequently, and if the association is successful, the publication of that same researcher will be added to the database.

This study concluded that Authenticus contains some limitations, especially when it comes to crossing the data and indicators. Some of the identified limitations are the absence of the h-index indicator and quartiles per publication which are not totally correct, it does not cross the platform data with the publications coming from Google Scholar, it contains limiting filters, it only contains a visualization form and it does not contemplate indicators related to projects and reviews.

3 Material and method

Given the complexity of the work, the Design Research Science and Scrum methodologies were used, as they provided a set of principles, practices and techniques that assisted in the development of the project. Next, Design Research Science is described as a scientific methodology and SCRUM as a methodology to assist in efficient project management. Tools, relevant metrics and the three scientific platforms involved in the work, namely, ORCID, Scopus, and Web Of Science are also described in this chapter.

3.1 Design Science Research

The Design Science Research methodology has emerged in the field of information technology supported by results, which offers a set of procedures and practices for the development of research projects. Design Science Research divides the process into six steps, these being as follows:

- **Problem Identification and Motivation:** in this step, the problem and the research question are defined, and the value of the solution is justified. In the case of the present work, the problem is related to the lack of data transversality among the existing platforms, so the research question is "In what way is it possible to reconcile the data of profiles, quartiles and indexes from various platforms?".
- **Define the Objectives for a Solution:** in this step, the objectives of the solution are defined.The main goal of the solution is to group and cross-reference data from researchers from various sources (Scopus, Google Scholar, ORCID) on a global scale.
- **Design and Development:** in this work, this step defines the artifact design of the RDProfile prototype, referred to in chapter 4.
- **Demonstration:** this step presupposes tests on the functioning of the developed product, demonstrated in chapter 5.
- **Evaluation:** in this step, the developed prototype is evaluated according to the defined goals.
- **Communication:** this step includes communicating the problem and the importance of the artifact, as well as its usefulness and effectiveness to researchers and other relevant target audiences. This article is an example of the communication of the artifacts developed.

3.2 SCRUM

The Scrum is an agile methodology that uses an iterative and incremental approach to optimize predictability and to control risk, relying on three pillars of implementation, namely transparency, inspection, and adaptation [11]. The fundamental unit of scrum is a scrum team. Scrum teams, composed by teams and organizations generating value through adaptive solutions to complex problems, are cross-functional, which means that members have all the skills needed to create value every sprint [11]. Thus, the Scrum methodology was used to assist in the efficient management and structuring of the project, speeding up the development of the practical part of the work developed.

3.3 Tools

Table 1 describes the tools and python libraries used to create the algorithm.

Table 1 Tools Description.

Tools	Description
Python	Versatile language for software development, web apps, and data analysis. Provides libraries like NumPy, SciPy, and Pandas for numerical computing, scientific tasks, and data manipulation
os	Python library used to specify the working directory
glob	Python library used to return all csv files (files with data regarding quartiles)
pandas	Python library used to manipulate data, in this case to combine all csv files into one
csv	Python library to load the generated csv (combined csv)
json	Python library used to transform the combined csv into a json file, used later to load the data into MongoDB
MongoDB	Scalable and flexible non-relational database program for storing large amounts of data. Utilized for storing relevant data in the context of scientific publications and researchers

3.4 Metrics

In order to expose the most relevant information about each researcher, metrics and indicators were selected to enrich the profile of each researcher.

- **Number of Total Citations:** Number of times the researcher's scientific publications were cited by other publications;
- **H-index:** This metric is defined as the number of papers with citation number>h, as a useful index to characterize the scientific output of a researcher [8];
- **CiteScore:** It results from dividing the number of citations received in the last four years by the number of publications that have been published in that same time interval;
- **SCImago Journal Ranking (SJR):** Is a size-independent metric aimed at measuring the current "average prestige per paper" of journals for use in research evaluation processes[7]. There are four quartiles: Q1, Q2, Q3, Q4. In quartile 1 (Q1) are the most important and prestigious scientific journals, while in quartile 4 (Q4) are the least prestigious scientific journals;
- **Source normalized impact per paper (SNIP):** Aiming to allow direct comparison of sources in different subject areas, this metric results from the ratio of the scientific journal's citation count per article and the citation potential in its subject field;
- **Journal Impact Factor (JIF):** Corresponds to the average number of times articles published in the last two years have been cited in the Journal Citations Report (JCR).

3.5 ORCID

ORCID is a platform that, through a digital identifier (ORCID iD), allows the distinction among researchers. Each researcher has his ORCID iD and, in addition, also has his own register (ORCID Record) which contains information about himself and his work. Each researcher manages his own ORCID record data and may place different information about himself. This information includes his name, biography, emails, websites and social links, keywords, countries, and activities[9]. All activities from the investigator profile include:

- **Employment:** Employment lists of organizations where the researcher has been professionally affiliated;
- **Education and Qualifications:** Where the researcher studied and the educational educational or professional qualifications that were awarded to him/her;
- **Invited positions and distinctions:** Positions the researcher has held and awards or awards he or she has received in recognition of his or her achievements;
- **Membership and service:** Memberships in society or association and donations of time or other resources in the service of an organization;
- **Funding:** Grants, awards, and other funding that the researcher has received to support his or her project;
- **Works:** The researcher's work and images of his or her research results, such as publications, conference presentations, data sets among others.

3.6 Libraries

For the creation of the library merging algorithm, the Scopus and Web Of Science APIs, among others, were used. For this reason, it is relevant to understand these two indexing libraries.

Scopus is a comprehensive database specializing in abstracts and citations with enriched data and scholarly literature linked across a wide range of disciplines. This scientific platform is owned by Elsevier and covers various academic disciplines such as science, technology, medicine, social sciences and humanities. Scopus has a large number of indexed publishers and is one of the largest databases of abstracts and citations of peer-reviewed literature[6].

The Web Of Science, owned by Clarivate, is a global publisher-independent citation database. This scientific platform provides researchers with information and technology for the global scientific research community. It provides data and analytics, as well as customized tools and professional services to researchers and the entire research community such as universities, research institutions, national and local governments, and private and public research funding organizations around the world. It also supports more than 95 percent of the world's leading research institutions, multiple governments, and national research agencies with about 20 million researchers in more than 7,000 research organizations[4].

4 RDProfile Algorithm

This section is about the architecture and structure used for the development of the RDProfile platform.

4.1 Architecture

As can be seen in Figure 1, the API is communicating regularly with the APIs of ORCID, APIs, collecting and uploading data about researchers and publications to the database. The communication between the various APIs is done through functions that are timed to run weekly, allowing the information to be kept up to date. In addition, two distinct processes are carried out that aim to collect data relative to the scientific indicators (quartiles, h-index, sjr, jif, among others). The first process consists in exporting data through the SCImago platform – the rankings and quartiles from all journals since 1999 until now (2022) – , which is then transformed and uploaded to MongoDB. For the second process, a Web Scraping mechanism is used that allows collecting data from Clarivate, which is also later uploaded to the database.

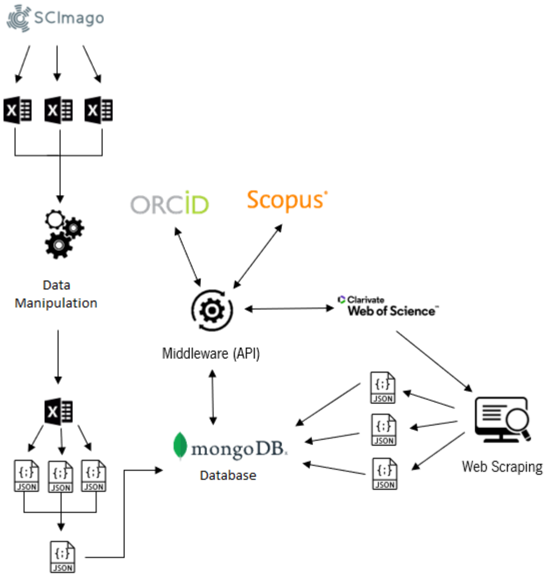


Figure 1 Architecture of the algorithm.

Figure 2 shows how the algorithm works at a high level. First it is checked if the publications are in the database. If they are, it is checked to see if they are unique, otherwise the process ends. If it is unique it is inserted in the "newPublications" array, if not it is alerted that it already exists and verifies the publications again. Next, a series of validations are performed. If the publication is from Web Of Science and Scopus it is inserted into the Web Of Science and Scopus Model, if it is from Scopus it is inserted into the Scopus Model, if it is from Web Of Science it is inserted into the Web Of Science Model. For all these validations the quartiles of the respective publications are added. If all three decisions are false, new publications are checked again and the process is repeated. All the models used are described in subsection 4.2.

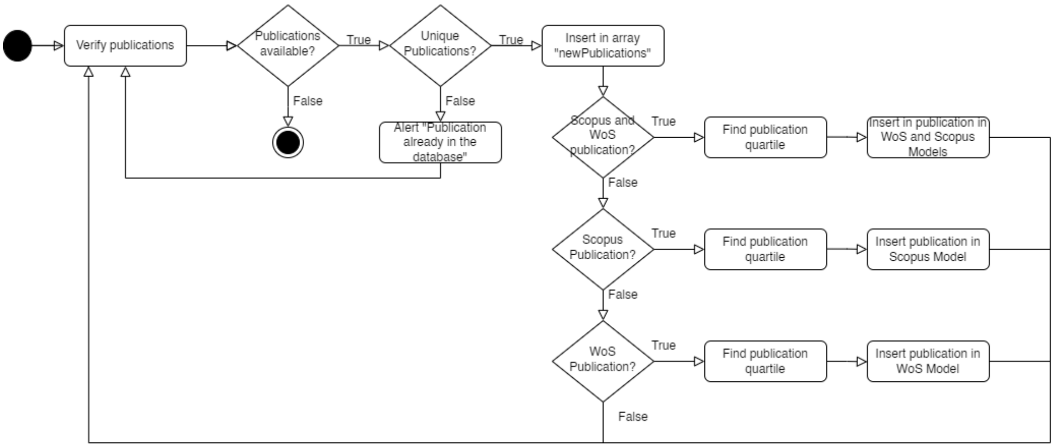

Verify publications
Publications available?
True
False
Unique Publications?
True
False
Alert "Publication already in the database"
Insert in array "newPublications"
Scopus and WoS publication?
True
False
Find publication quartile
Insert in publication in WoS and Scopus Models
Scopus Publication?
True
False
Find publication quartile
Insert publication in Scopus Model
WoS Publication?
True
False
Find publication quartile
Insert publication in WoS Model

Figure 2 High level diagram of the algorithm.

4.2 Modeling, Collecting and Loading Data

The modeling phase consists of creating a data model to define how the data is related to each other and stored in the database. In total six different models were created, namely Researcher Model, Scopus Model, Web Of Science Model, Quartiles Model, Publications Model and Metrics Model. For data collection, some sources were used, namely, API of the Algoritmi Center, ORCID API, Scopus API and the Web Of Science API. After the data collection, the data from each model is loaded into MongoDB. Each created model is described below.

4.2.1 Researcher Model

The Researchers Model contains the data regarding scientific researchers. Table 2 contains a brief individual description of each field and the types of data present in this model.

Table 2 Researcher Model.

Field name	**Description**
id	Unique Identifier of the researcher
academic_degree	Academic degree of the researcher
degois	Unique Identifier CiênciaVitae
gscholar	Google Scholar Unique Identifier
institution	Educational institution of the researcher
name	Name of researcher
orcid	ORCID Unique Identifier
researcher_id	Web Of Science Unique Identifier
scopus	Scopus Unique Identifier
articles	Researcher's articles
name_slug	Researcher's name slug

4.2.2 Scopus Model

The Scopus Model includes data related to Scopus publications, collected using the Scopus API, which returns the scientific publication data according to the eid of the publication. Thus, the Table 3 shows all the field names and its description.

4.2.3 Web Of Science Model

The Web Of Science Model includes data related to Web Of Science publications, collected using the Web Of Science API, which returns the scientific publication data according to the uid of the publication. Thus, the Table 4 shows all the field names and its description.

4.2.4 Quartiles Model

The data from the Quartiles Model were collected through two distinct processes, referring to indicators from Scopus and Web Of Science. For the collection of Scopus indicators the following activities are performed: export the indicator data, transform the data, create the function to upload the data, and upload the data to the database. Regarding the collection of Scopus indicators, it is necessary to process the data from the Journal Citation Reports using a Web Scraping engine. Then the data are loaded into the database. Table 5 contains a brief individual description of each field present in this model.

Table 3 Scopus Model.

Field name	**Description**
dc_identifier	Object identifier of the document
affiliation	Publication's affiliation
article_number	Article number
author	Author of the publication
citedby_count	Number of citations of the publication
dc_creator	Lead author of the publication
eid	Unique Identifier for a publication present in Scopus
fa	Total access
link	Open access status
openaccess	Open Access status
openaccessFlag	Open Access status (boolean)
pii	Publication item identifier
prism_aggregationType	Type of aggregation (i.e. journal)
prism_coverDate	Publication date (YYYY-MMDD)
prism_coverDisplayDate	Date of publication (original text)
prism_doi	Object identifier of the document
prism_eIssn	Electronic serial number (international standard) of a publication
prism_issn	Serial number (international standard) of a publication
prism_pageRange	Interval of publication pages
prism_publicationName	Publication name
prism_url	Publication URL
prism_volume	Volume number of the publication
pubmed_id	Medline Unique Identifier
source_id	Identifier of publication's source
subtype	Subtype of publication (i.e. cp)
subtypeDescription	Description of the subtype of publication (i.e. conference paper)
title	Article title

Table 4 Web Of Science Model.

Field name	**Description**
uid	Unique identifier of a publication in the Web Of Science
citations	Number of citations
identifiers	Publication identifiers (doi, issn, eissn, isbn, eisbn)
keywords	Article keywords
links	Publication links
names	Publication authors
source	Origin of the publication
title	Publication title
types	Publication type (article, editorial material, meeting)

Table 5 Quartiles Model.

Field name	Description
rank	Journal rank
sourceId	Journal Unique Identifier
title	Journal Title
type	Type of journal
issn	Serial number (international standard) of the journal
eIssn	Electronic serial number (international standard) of a publication
sjr	Scimago Journal Rank
sjr_best_quartile	Best quartile of the journal
h_index	Journal h-index
jif	Journal Impact Factor
jci	Journal Citation Indicator
total_docs	Total number of documents
total_docs_3	Total number of documents in the last 3 years
total_refs	Total references
total_cites_3	Total Citations in the last 3 years years
total_citations	Total citations
citable_docs_3	Number of documents citable in the last 3 years
cites_by_doc_2	Number of citations per document in the last 2 years
ref_by_doc	Total references per document
country	Journal Country
region	Journal Region
publisher	Journal Publisher
coverage	Journal Coverage
year	Journal year
categories	Journal categories

4.2.5 Publications Model

This model includes data from Scopus and Web Of Science publications, as well as indicators related to them. The mechanism for data collection regarding the Publications Model results from the interaction between the previously loaded Scopus Model, Web Of Science Model and Quartiles Model. Table 6 contains a brief individual description of each field present in this model.

4.2.6 Metrics Model

The Metric Model includes data related to the indicators of scientific researchers, and just like the Researchers' Model resulted in a model that aggregates data from other models. The mechanism to collect data regarding the Metrics Model results from the interaction between the previously loaded Researchers Model and Publications Model. It also requests the Scopus API to obtain the researcher's h-index. Table 7 contains a brief individual description of each field present in this model.

Table 6 Publication Model.

Field name	**Description**
data	Identifiers (eid, uid, issn, eissn, doi, pii) and year of publication publication
dc_identifier	Object identifier of the document
title	Publication title
citedby_count	Number of citations of the publication
subtypeDescription	Description of the subtype of publication (i.e. conference paper)
link	Link to the publication
types	Publication type (article, editorial material, meeting)
scopus_id	Scopus Unique Identifier
wos_id	Web Of Science's Unique Identifier
author	Publication authors
quartiles	Publication quartiles
citations	Number of citations present in the publication

Table 7 Metrics Model.

Field name	**Description**
id	Unique identifier of the researcher
calgId	Algoritmi Center's Unique Identifier
academic_degree	Researcher's academic degree
orcidId	ORCID's Unique Identifier
scopusId	Scopus's Unique Identifier
gscholar	Google Scholar's Unique Identifier
researcher_id	Web Of Science's Unique Identifier
name	Researcher name
research_lab	Research Laboratory name
research_groups	Research groups name
articles	Investigator articles
citation_count	Researcher's number of citations
editorial_count	Researcher's number of editorials
h_index	H-index of researcher (Scopus)
publication_count	Researcher's number of publications
name_slug	Researcher's name slug
quartiles	Researcher's number of quartiles (total and by years)
final_score	Final score of researcher
ranking	Researcher Rank

5 Proof of Concept

The RDProfile emerges intending to create a robust solution that includes information about publications, researchers and scientific indicators, by crossing and grouping data from various platforms. Prior to the creation of the algorithm described in this paper, Algoritmi Center, a Portuguese scientific institution, developed Algoritmi Center API within the RDProfile project. Thus, the tests performed were done using real data from this institution.

To better understand the algorithm, the actual flow of the general process is shown below:

1. Collect investigator data through the ORCID API and Algoritmi center API;
2. Upload the data to the Researchers Model;
3. Collect data for each publication via Scopus API and Web Of Science API;
4. Upload publications with uid and eid simultaneously to the Publication Model;
5. Upload publications with eid to the Scopus Model and publications with uid to the Web Of Science Model;
6. Collect and load data from Scopus and Web Of Science indicators into the Quartiles Model;
7. Upload data to the Metrics Model, cross referencing data from the Researchers Model and the Publications Model.

Once the flow is complete, it is possible to see the loaded models in MongoDB. Next, real data examples of each model's result is represented in tables. Since the Metrics Model corresponds to the general model, which aggregates the data from the others models, the final result represents the aggregation of relevant data for each researcher. Because of that, it is shown, in image format, real data related with one of the authors of this paper.

In the Table 8 it is shown an example of a Researcher Model, from the researcher referred above, with its attributes.

Table 8 Researchers Model Example.

Field name	**Variable**
id	"730"
academic_degree	"PhD"
dgois	"F311-7C27-F8DA"
gscholar	"HoeD9UgAAAAJ"
institution	"Escola de Engenharia da Universidade do Minho"
name	"Carlos Filipe da Silva Portela"
orcid	"0000-0003-2181-6837"
researcher_id	"G-5324-2012"
scopus	[0: "57194071672"]
articles	Array
name_slug	"carlos-filipe-da-silva-portela"

In the Table 9 it is shown the data of the Scopus publication with eid 2-s2.0-85129325287, present in Scopus Model.

Table 9 Scopus Model Example.

Fields	Variable
dc_identifier	“SCOPUS_ID:85129325287”
affiliation	Array
article_number	null
author	Array
citedby_count	“0”
dc_creator	“Azevedo J.”
eid	“2-s2.0-85129325287”
fa	true
link	Array
openaccess	“0”
openaccessFlag	flase
pii	null
prism_aggregationType	“Book Series”
prism_coverDate	“2022-01-01”
prism_coverDisplayDate	“2022”
prism_doi	“10.1007/978-981-16-7618-5_27”
prism_eIssn	“23673389”
prism_issn	“23673370”
prism_pageRange	“307-318”
prism_publicationName	“Lecture Notes in Networks and Systems”
prism_url	https://api.elsevier.com/content/abstract/scopus_id/85129325287
prism_volume	“350”
pubmed_id	null
source_id	“21100901469”
subtype	“cp”
subtypeDescription	“Conference Paper”
title	“Convolutional Neural Network – A Practical Case Study”

In the Table 10 it is shown the data of the Wos publication with uid 000676684500001.

Table 10 Web Of Science Model Example.

Field name	Variable
uid	“WOS:000676684500001”
citations	{ 0: { bd:“WOS”, count:1, id:6393e85271c0384234a1218a} }
identifiers	{ doi:“10.3390/fi13070178”; issn:null; eissn:“1999-5903”; isbn:null; eisbn:null}
keywords	{authorKeywordsArray}
links	Array
names	{Authors Array, Book Editor Array}
source	{sourceTitle: “FUTURE INTERNET”, publishYear: 2021, ... }
title	“Data Science and Knowledge Discovery”
types	{0:“Editorial material”}

In the Table 11 it is shown an example of Quartiles Model with its attributes.

In the Table 12 it is shown an example of Publiations Model with its attributes.

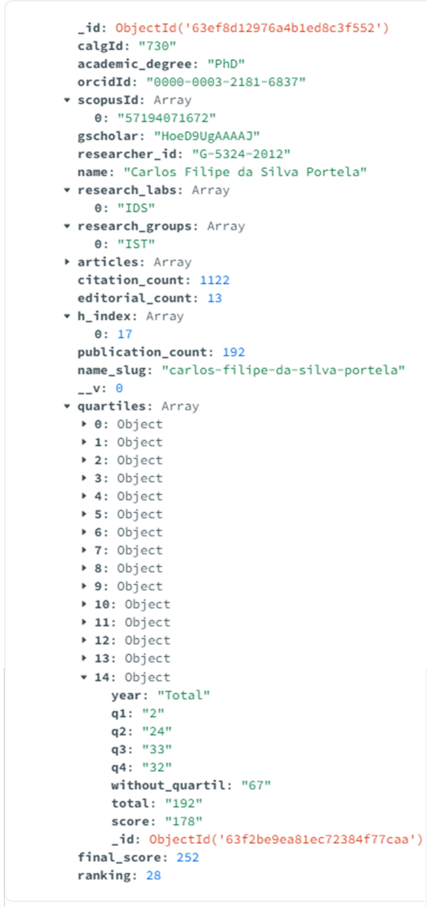
Table 11 Quartiles Model Example.

Field name	**Variable**
rank	“1”
sourceId	“16810”
title	“Annual Review of Biochemistry”
type	[0: journal]
issn	[0: 15454509, 1: 00664154]
eIssn	Array
sjr	“50,518”
sjr_best_quartile	“Q1”
h_index	“293”
jif	null
jci	null
total_docs	“30”
total_docs_3	“80”
total_refs	“5913”
total_cites_3	“3484”
total_citations	null
citable_docs_3	“80”
cites_by_doc_2	“35,78”
ref_by_doc	“197,10”
country	“United States”
region	“Northern America”
publisher	[0:“Annual Reviews Inc.”]
coverage	“1946-1948, 1950-1960, 1962-2020”
year	1999
categories	[{_id: 63596f8c8a7fc80c14936350, area:“Biochemistry”, quartil:“Q1”}]

Table 12 Publications Model Example.

Field name	**Variable**
data	{eid: “2-s2.0-85091404413”, uid: null, issn: null, eissn: null, doi: null, pii: null, year: 2020}
dc_identifier	“SCOPUS_ID:85091404413”
title	“A SWOT analysis of big data in healthcare”
citedby_count	“0”
subtypeDescription	“Conference Paper”
link	https://www.scopus.com/record/display.uri?eid=2-s2.0-85091404413&origin=resultslist&sort=plf-f
types	Array
scopus_id	“635018a59e663c3c83f5931a”
author	Array
quartiles	{jif: null, jcr: Array, sci: Array}
citations	Array

In the figure 3 it is shown an example of Metrics Model loaded in MongoBD. This figure shows all the relevant real data from the researcher Carlos Filipe Portela. Here we can see, among others, the researcher’s unique identifiers of Algoritmi Center, ORCID, Scopus, Google Scholar and Web Of Science. In addition to the number of publications and citations, a list of these publications, containing all relevant data, is also included. As for the quartiles parameter, it contains, for each year, the number of publications per quartile – from quartile 1 to 4, and without quartile. Additionally, this list contains a last global element, that contains the total sum of publications by quartiles, as can be seen in the figure.



```
  _id: ObjectId('63ef8d12976a4b1ed8c3f552')
  calgId: "730"
  academic_degree: "PhD"
  orcidId: "0000-0003-2181-6837"
▾ scopusId: Array
    0: "57194071672"
  gscholar: "HoeD9UgAAAAJ"
  researcher_id: "G-5324-2012"
  name: "Carlos Filipe da Silva Portela"
▾ research_labs: Array
    0: "IDS"
▾ research_groups: Array
    0: "IST"
▸ articles: Array
  citation_count: 1122
  editorial_count: 13
▾ h_index: Array
    0: 17
  publication_count: 192
  name_slug: "carlos-filipe-da-silva-portela"
  __v: 0
▾ quartiles: Array
  ▸ 0: Object
  ▸ 1: Object
  ▸ 2: Object
  ▸ 3: Object
  ▸ 4: Object
  ▸ 5: Object
  ▸ 6: Object
  ▸ 7: Object
  ▸ 8: Object
  ▸ 9: Object
  ▸ 10: Object
  ▸ 11: Object
  ▸ 12: Object
  ▸ 13: Object
  ▾ 14: Object
      year: "Total"
      q1: "2"
      q2: "24"
      q3: "33"
      q4: "32"
      without_quartil: "67"
      total: "192"
      score: "178"
      _id: ObjectId('63f2be9ea81ec72384f77caa')
  final_score: 252
  ranking: 28
```

Figure 3 Metric Model example.

The algorithm developed for RDProfile fills some of the identified gaps of the existing algorithms, namely in Authenticus. This algorithm is more accurate, scalable and modular than the existing and mentioned in section 2. It was created so that any researcher can easily access their updated data. In other words, the RDProfile algorithm, in addition to the total and per-year citations and quartiles indicators, also provides the h-index indicator and indicators about projects and reviews. It also cross-references and groups all researchers' data and indicators on a global scale. Based on this, the platform will have unique features that provide a better experience for researchers and other users.

6 Conclusion

Based on the work, the feasibility of creating the library merging algorithm is verified, as well as the added value it offers. To answer the research question, a literature review was conducted and a prototype of a web solution capable of grouping and cross-referencing data from ORCID, Scopus and Web Of Science was developed. The most in-depth study of ORCID and Authentics allowed us to understand how RDProfile can be an added value. Both the limitations of the missing indicators and the functionalities of each platform were overcome. In other words, the RDProfile platform answers the research question by integrating:

- 4 APIs (ORCID, Scopus, Web Of Science and Algoritmi Center);
- 3 endpoints of the quartiles;
- 2 endpoints for the list of researchers with and without filter applied;
- 3 endpoints for the list of publications with and without filter applied;
- 1 endpoint for the researcher profile with the associated metrics;
- a user interface, for the most recent publications;
- 13 functionalities at the API and user interface level.

The developed algorithm proves that it is possible to integrate the main sources of scientific information in a single platform with highly interoperable and cross-reference data capabilities. Therefore, it becomes possible to create a scalable solution that any researcher can use to do their own analyses, representing a contribution to science.

Further improvements in functionalities and in the user interface are needed to make RDProfile a more complete and useful platform. It is important to optimize the solution so that it includes more platforms, such as Google Scholar, Crossref, Dimensions, among others. In addition, errors related to number of citations should be corrected, since the existing systems are limited to check articles that are correctly identified. Thus, this improvement is important to ensure maximum data integrity and reliability.

References

1 Miriam Baglioni, Paolo Manghi, Andrea Mannocci, and Alessia Bardi. We can make a better use of orcid: Five observed misapplications. *Data Science Journal*, December 2021. `doi:10.5334/dsj-2021-038`.

2 Joanna G. Carvalho. O cv científico dos investigadores. Master's thesis, U. Minho, 2020.

3 Joanna G Carvalho, João Oliveira, Luciana Machado, and Filipe Portela. A benchmarking study of the scientific platforms. submmited to publication, 2023.

4 Clarivate Analytics. Web of science. `https://clarivate.com/products/webofscience/`.

5 Rodrigo Costas, Carmen Corona, and Nicolas Robinson-Garcia. Could orcid play a key role in meta-research? discussing new analytical possibilities to study the dynamics of science and scientists, May 2022. `doi:10.31235/osf.io/sjck6`.

6 Elsevier. Scopus. URL: `https://www.elsevier.com/solutions/scopus`.

7 Borja González-Pereira, Guerrero-Bote Vicente, and Felix Moya-Anegon. A new approach to the metric of journals' scientific prestige: the sjr indicator. *J. Informetrics*, 4:379–391, July 2010. `doi:10.1016/j.joi.2010.03.002`.

8 J. E. Hirsch. An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences*, 102(46):16569–16572, 2005. `doi:10.1073/pnas.0507655102`.

9 ORCID. What is orcid? `https://info.orcid.org/what-is-orcid/`.

10 F. Portela. Rdprofile - ficha tecnica, 2022.

11 Ken Schwaber and Jeff Sutherland. Scrum: A framework for managing agile projects. Online, 2013. URL: `https://www.scrum.org/resources/what-is-scrum`.

12 Vivek Singh, Prashasti Singh, Mousumi Karmakar, Jacqueline Leta, and Philipp Mayr. The journal coverage of web of science, scopus and dimensions: A comparative analysis. *Scientometrics*, 126, March 2021.

13 Vivek Kumar Singh, Prashasti Singh, Mousumi Karmakar, Jacqueline Leta, and Philipp Mayr. The journal coverage of web of science, scopus and dimensions: A comparative analysis. *Scientometrics*, 126(6):5113–5142, 2021.

Integrating Gamified Educational Escape Rooms in Learning Management Systems

Ricardo Queirós
School of Media Arts and Design & CRACS - INESC TEC, Polytechnic Institute of Porto, Portugal

Carla Pinto
School of Engineering & CMUP, Polytechnic Institute of Porto, Portugal

Mário Cruz
School of Education & inED, Polytechnic Institute of Porto, Portugal

Daniela Mascarenhas
School of Education & inED, Polytechnic Institute of Porto, Portugal

Abstract

Escape rooms offer an immersive and engaging learning experience that encourages critical thinking, problem solving and teamwork. Although they have shown promising results in promoting student engagement in the teaching-learning process, they continue to operate as independent systems that are not fully integrated into educational environments. This work aims to detail the integration of educational escape rooms, based on international standards, with the typical central component of an educational setting - the learning management system (LMS). In order to proof this concept, we present the integration of a math escape room with the Moodle LMS using the Learning Tools Interoperability (LTI) specification. Currently, this specification comprises a set of Web services that enable seamless integration between learning platforms and external tools and is not limited to any specific LMS which fosters learning interoperability. With this implementation, a single sign-on ecosystem is created, where teachers and students can interact in a simple and immersive way. The major contribution of this work is to serve as an integration guide for other applications and in different domains.

2012 ACM Subject Classification Social and professional topics → Computer science education

Keywords and phrases Escape Rooms, Interoperability, Learning Management Systems, Standardization

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.15

Category Short Paper

Funding This work of M. Cruz and D. Mascarenhas is funded by National Funds through the FCT - Fundação para a Ciência e a Tecnologia, I.P., under the scope of the project UIDB/05198/2020 (Centre for Research and Innovation in Education, inED).
The author C. Pinto was partially supported by CMUP, which is financed by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project with reference UIDB/00144/2020.
All authors would like to thank Erasmus + MATH-DIGGER project, ref 2021-1-PT01-KA220-HED-000032234.

1 Introduction

Escape rooms are interactive and immersive entertainment experiences in which participants are locked in a room. The goal is to solve a variety of puzzles, riddles, and challenges within a set time limit to ultimately find a way to escape from the room. Escape rooms often have a specific theme or storyline which can be integrated into many academic disciplines such as mathematics, computer science, STEM subjects, physics, biology and many others [4].

© Ricardo Queirós, Carla Pinto, Mário Cruz, and Daniela Mascarenhas;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 15; pp. 15:1–15:8
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

These escape rooms, typically used for entertainment games, have been adapted for educational purposes as a way to engage students in learning in an interactive and playful way. This approach has received attention from educational researchers, who have investigated how Educational escape rooms (EER) can be used to promote learning and develop skills such as teamwork, problem-solving, decision-making, and critical thinking [3].

While EER have proven highly effective in terms of student engagement and active learning [6], it is important to recognise that interoperability with learning environments, such as Learning Management Systems (LMS), is crucial to create a unified ecosystem where teachers and students can easily access and exchange information with each other through two-way communication [5].

This work aims to address the integration of educational escape rooms into learning management systems. Although escape rooms have shown promising results in promoting student engagement and learning, they continue to operate as independent systems that are not integrated into educational environments. The objective of this study is to demonstrate the integration of an educational escape room – for learning Maths – with Moodle LMS using an international specification called IMS LTI [1]. This standard includes a set of services that enable seamless integration and is not limited to any specific LMS. By implementing this approach, a single sign-on ecosystem is created, where teachers and students can interact seamlessly. Teachers can expose links to specific escape rooms within the LMS, and students can access and solve the challenges. Additionally, teachers can monitor students' performance in these rooms through the LMS's gradebook without the burden of manually collecting data from the escape room, reducing the potential for errors and saving valuable time.

The rest of the article is structured in three sections: the second section presents related work on EER and integration frameworks. The following section present the integration architecture, followed by a detailed explanation of its components and the steps needed to integrate the two components using the IMS LTI specification, more precisely, the integration between Moodle LMS and a Math escape room. Finally, the contributions of this article to the scientific community are presented as well as the future work.

2 Related work

EER are a type of learning activity where students are placed in a simulated environment where they must solve a series of puzzles and challenges in order to escape within a given time limit. While solving challenges, students learn a variety of subjects and skills, from critical thinking and problem-solving to content-specific concepts and knowledge. At the same time, they promote collaboration, communication, and creativity. However, in order to maximize the benefits of EER, it is important to integrate them with other learning tools and platforms, such as learning management systems.

2.1 Integration of LMS with EER

Integration of EER with LMS allows for a seamless transfer of data between the two systems, making it easier for teachers to manage student progress and for students to access and engage with the content with a single sign-on environment. The following are some examples of the integration between these two types of systems:

- **Schoology** - Schoology is a popular LMS that offers integration with EER, such as Breakout EDU [2]. This integration allows teachers to assign escape room challenges to students directly through the Schoology platform, and students can complete the challenges and submit their results back to Schoology for grading and assessment.

- **Canvas** - Canvas is another popular LMS that offers integration with EER, such as Escape Classroom and Breakout EDU. With this integration, teachers can easily embed Escape Classroom challenges into their Canvas courses, and students can complete the challenges and submit their results directly through the Canvas platform.
- **Moodle** - Moodle is an open-source LMS that offers integration with EER through several plugins and modules. For example, the "Escape Room" module allows teachers to create and manage escape room challenges within their Moodle courses, and the "Breakout EDU" plugin allows for integration with Breakout EDU escape rooms.
- **Blackboard** - Blackboard is a widely used LMS that offers integration with educational escape rooms through the "Breakout Room" tool. This tool allows teachers to create and assign escape room challenges to students within their Blackboard courses, and students can complete the challenges and submit their results back to Blackboard for grading and assessment.
- **Google Classroom** - Google Classroom is a free LMS that offers integration with EER through various third-party tools and platforms. For example, the "Escape Room Extension" for Google Chrome allows teachers to create and manage escape room challenges within Google Classroom, and the "Escape Room Digital Locks" tool allows for integration with various types of digital escape rooms.

2.2 Integration specifications

The IMS LTI specification [1] defines a set of standard messages and protocols for communication between the LMS and external tools or applications. This ensures that different tools and applications can work seamlessly with different LMS platforms, making it easier for educators to use a variety of tools to enhance the learning experience.

In its 1.3 version, the specification integrates the LTI Advantage as an extension to provide additional services that aim to enhance the interoperability, security, and user experience of the LTI platform. The main services provided by LTI Advantage are:

- **Deep Linking** - allows educators to link directly to specific content items in an external tool, such as a specific quiz or assignment. This enables users to access specific content items without having to navigate through the external tool interface.
- **Assignment and Grade Services** - allows the exchange of assignment and grade data between the LMS and external tools. This enables educators to create and grade assignments directly within the LMS, using external tools to provide additional functionality or content.
- **Names and Role Provisioning Services** - allows the exchange of user and role data between the LMS and external tools. This enables external tools to customize the user experience based on the user's role or other user data.

There are several alternatives to IMS LTI that provide similar functionality for integrating external learning tools with LMS, such as:

- SCORM (Sharable Content Object Reference Model) - a set of standards and specifications that define how e-learning content can be packaged and delivered to learners through an LMS. SCORM includes a runtime environment (the SCORM player) that enables the communication between the content and the LMS.
- xAPI (Experience API) - a newer specification that allows for more flexible tracking and reporting of learning activities, beyond what is possible with SCORM. xAPI enables the collection and analysis of data about a wide range of learning experiences, including informal and social learning.

- Common Cartridge - a standard for packaging and sharing e-learning content, including multimedia, assessments, and other interactive learning objects. Common Cartridge aims to provide a more portable and flexible way of sharing content across different LMS platforms.
- AICC (Aviation Industry Computer-Based Training Committee) - A specification for integrating external learning tools with LMSs, primarily used in the aviation industry. AICC provides a set of communication protocols and data structures for exchanging data between the LMS and external tools.

From all these alternatives presented, the xAPI (Experience API) specification is one of most promising specification. xAPI, also known as Tin Can API, is a specification for tracking and recording learning experiences in a standardized format. It is designed to capture data from a wide range of sources, including mobile devices, simulations, and virtual environments, among others. The xAPI specification enables the collection of a wide range of data, including learning experiences that occur outside traditional LMS, such as informal learning experiences or on-the-job training. This data can include a variety of information, such as learning objectives, assessment scores, and even learner emotions.

The xAPI specification is closely related to a Learning Record Store (LRS). An LRS is a database that stores learning experience data in the format specified by xAPI. It is the system responsible for collecting, storing, and retrieving xAPI statements, which are the standardized format for recording learning experiences. The LRS can also provide analytics and reporting capabilities that enable organizations to gain insights into learning experiences and improve learning outcomes. By recording learning experiences, organizations can gain a more complete understanding of how their learners are interacting with learning content, and can use this information to improve training programs and support individual learning needs.

It's worth noting that each of these alternatives has its own strengths and weaknesses, and the choice of which to use may depend on specific use cases and requirements. However, IMS LTI is currently the most widely adopted and supported standard for integrating external learning tools with LMSs.

3 Integration framework

This section outlines the process of integrating a learning platform (Moodle LMS) with an external tool (Math Escape Room). Firstly, the integration architecture is presented, followed by a brief explanation of one of its components (the escape room). Finally, the steps to integrate the two components, using the IMS LTI specification are listed.

3.1 Integration architecture

An integration architecture refers to the overall structure and design of the framework that enables different software applications or systems to communicate with each other and exchange data seamlessly. Figure 1 illustrates the integration architecture for the communication between the learning platform and the external tool.

The LMS serves as the entry point for both teachers and students. Teachers can add links to access the external tool (either as a whole or for a specific room), while students can access these links and enter the escape room to solve challenges. All performance results are then sent back to the learning platform, specifically to the LMS gradebook of the corresponding course. All these interactions are done implementing the IMS LTI 1.3 (with Advantage services). Meanwhile, all student activity, such as academic performance, behavior patterns,

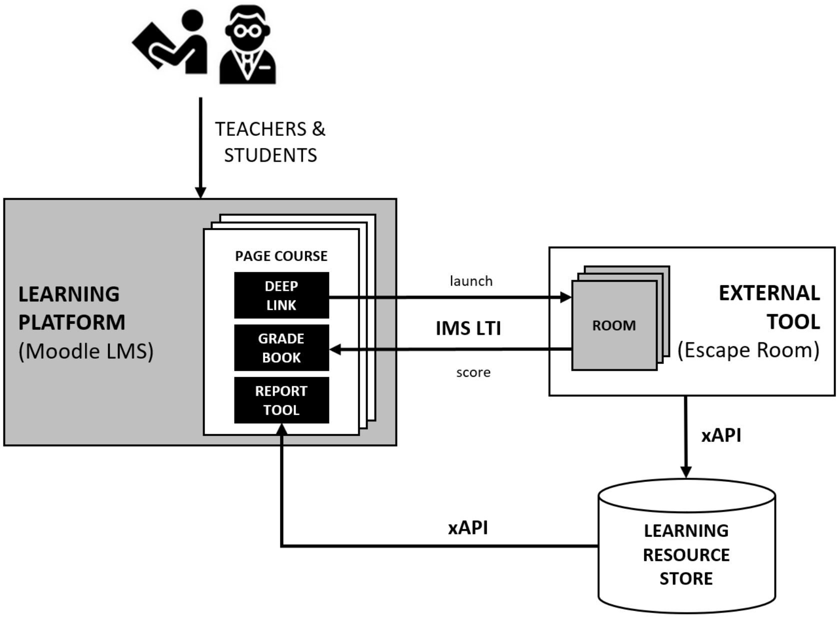


Figure 1 Integration of the learning platform and the external tool.

and social interactions, among others, is sent to special databases called LRS and then forwarded to reporting tools for analysis. All this communication is guaranteed by the xAPI specification. By analyzing student data, educators can gain insights into how students learn, identify challenges they face, and determine the most effective strategies for promoting learning. They can use this information to customize instruction to the needs of individual learners, identify students who may be at risk, and improve overall course design and delivery.

3.2 Math Escape Room

A math escape room is a type of interactive game where players are presented with math problems and puzzles that they must solve to progress through a series of challenges or rooms. The ultimate goal of the game is to “escape” from the room by solving all of the puzzles before time runs out. Math escape rooms can be designed for a variety of skill levels, from basic arithmetic to advanced calculus. They can be used as a fun way to reinforce math concepts in a classroom setting, or as a team-building activity for a group of friends or colleagues.

For this proof of concept, we choose a math escape room from the Math-Digger European project[1] – MATH-DIGGER project - MATHematics Digital Escape Rooms platform – which aims to provide a free tool to maximize students’ enjoyment, engagement and motivation in their math learning process. Students will face exciting math exercises and problems as micro-games in Geogebra, developed for Linear Algebra and Analytical Geometry, Differential and Integral Calculus courses, mostly based in real world problems, and solve them in a virtual reality framework.

Despite its usefulness, it is implemented as a separate web platform to which a student can log in to solve math challenges served in a gamified form (e.g., as challenges and quests rewarded with points and badges, etc.). This means that the student must leave the LMS

---

[1] `https://www.ipp.pt/mathdigger/`

platform to solve the challenge(s), and his/her achievements are not transferred back to the LMS platform. Therefore, the instructor also must log into the Escape Room platform and, somehow, to see the students' individual progress and, finally, to combine this data with the LMS-based student activity data to produce the final grade. The goal of this integration work is therefore to fill the gap between interactive learning environments and popular LMS platforms using an LTI-based approach, in specific, to open the Escape Room ecosystem to educational environments. This is much more flexible and sustainable compared to writing plugins for popular LMS platforms, i.e., by providing an LTI-compliant interface to Escape Room platform, any LTI-compliant LMS can be connected to it. Using the plugin option, every LMS would require a dedicated plugin written only for it and any updates to LMS code could render older plugins incompatible. Furthermore, the LTI, as an established standard, it is expected to stay for a considerable time.

4 Integration steps

This section presents a guide with all the steps to allow the integration of a math escape room (the external tool) with Moodle LMS (the learning platform) using the IMS LTI specification.

It is expected that this section can be useful in other domains by replicating all the steps contained herein to link other environments (new LMS platforms and new external tools) into an educational setup. The main steps are:

1. Adding the external tool into the learning platform;
2. Registering the learning platform in the external tool;
3. Adding course activities in the learning platform.

4.1 Adding the external tool into the learning platform

In this section we detail the steps needed to add the math escape room as an external tool in Moodle LMS. To this end, sign into Moodle as an administrator and select manage external tools to configure a tool manually. This opens the external tool configuration form where the administrator should fill the following fields:

1. Tool name, URL (the URL of the escape room) and description;
2. Specification: LTI 1.3
3. Public Key type: RSA key
4. Public key: leave this field empty. This will be filled out later when we register the LMS (Moodle) as the platform in the escape room.

After saving changes, the Client ID will be filled in automatically which will be necessary for the next step.

4.2 Registering the learning platform in the external tool

The LMS platform needs to be registered in the external tool (the math escape room) to get a Public Key, which allows signatures of incoming messages and service requests to be verified. At the moment, there isn't a User Interface (UI) form to facilitate this process. Therefore, the register code is stored in an index.js file of the web app deployed in a server platform (Heroku).

Listing 1 Main code.

```
// Register platform
const platform = await lti.registerPlatform({
  url: MOODLE_URL,
  name: "MD",
  clientId: process.env.LTI_KEY,
  authenticationEndpoint: "{{MOODLE_URL}}/mod/lti/auth.php",
  accesstokenEndpoint: "{{MOODLE_URL}}/mod/lti/token.php",
  authConfig: {
    method: "JWK_SET",
    key: "{{MOODLE_URL}}/mod/lti/certs.php",
  },
});
```

This JavaScript object contains several properties. Each property contains a key and the respective value. The most important keys are: url - the URL for the platform; name - the name of the external tool configuration in the platform; clientid - the id generated after the creation of the external tool instance in the platform. Please add LTI_KEY as an environment variable in the server platform where the Web app is deployed (in this case in Heroku); endpoint - a set of URLs that defined platform endpoints. When we launch for the first time the escape room it will log the public key. The obtained public key should be added to the external tool in Moodle.

4.3 Adding course activities in the learning platform

After fully configuring the external tool, teachers may add an external tool activity by:

1. Select the added tool (in this case, the escape room) in the preconfigured tool;
2. Add a name and a description to the activity;
3. Select new window as Launch container
4. Enable all options, in the privacy section, namely Share launcher's name with the tool, Share launcher's email with the tool, and Accept grades from the tool;
5. Set Completion tracking, in the activity completion section, to show activity as complete when conditions are met and enable the Require grade option.

5 Conclusion

This paper presents a guide with all the steps needed to allow the integration of a learning platform (Moodle LMS) with the external tool (math escape room) using the IMS LTI specification. The expected impact is significant, as it removes the limitation in the adoption of external tools in a typical educational setting where the LMS plays a central role. Apart from satisfying the needs described above, it also addresses the psychological barrier of instructors being reluctant to add yet another platform to the portfolio of their educational IT tools. The major contribution of this work is to serve as an integration guide for other applications and in different domains.

References

1 IMS Global Learning Consortium. Ims learning tools interoperability (lti). `https://www.imsglobal.org/activity/learning-tools-interoperability`, 2021. Accessed: May 12, 2023.

2 Breakout EDU. Schoology integration. `https://www.breakoutedu.com/schoology`, 2021. Accessed: May 14, 2023.

3 Fernando Gamboa-Rodríguez, Juan Nicolás García-Sánchez, Eloy García-Sánchez, and Arturo Córdoba-Rangel. Educational Escape Rooms in Higher Education: A Systematic Literature Review. *Education Sciences*, 11(3):114, 2021. `doi:10.3390/educsci11030114`.

4 Agoritsa Makri, Dimitrios Vlachopoulos, and Richard Martina. Digital escape rooms as innovative pedagogical tools in education: A systematic literature review. *Sustainability*, 13:4587, April 2021. `doi:10.3390/su13084587`.

5 Nidhi Niyati, Pranav Awati, Neelam V Alai, and Ritu Jain. Escape rooms as an interactive learning tool: A systematic review. *International Journal of Emerging Technologies in Learning (iJET)*, 16(5):223–240, 2021.

6 Alice Veldkamp, Liesbeth Grint, Marie-Christine Knippels, and Wouter van Joolingen. Escape education: A systematic review on escape rooms in education. *Educational Research Review*, 31:100364, November 2020. `doi:10.1016/j.edurev.2020.100364`.

Romaria De Nª Srª D'Agonia: Building a Digital Repository and a Virtual Museum

Sara Cristina Freitas Queirós ✉
ALGORITMI Research Centre / LASI, DI-University of Minho, Braga, Portugal

Cristiana Araújo ✉ 🏠
ALGORITMI Research Centre / LASI, DI-University of Minho, Braga, Portugal

Pedro Rangel Henriques ✉ 🏠
ALGORITMI Research Centre / LASI, DI-University of Minho, Braga, Portugal

Abstract

Romarias are Christian pilgrimages that occur in order to celebrate a specific saint. *Romaria de Nª Srª d'Agonia* (RNS Agonia, for short) occurs aimed at celebrating *Nossa Senhora da Agonia*, patron of all Fishermen, at Viana do Castelo, Portugal. RNS Agonia is a very old event that surely belongs to the Minho's Cultural Heritage. There are many written documents, of various types, that describe the event; so, their digital preservation is mandatory. However, digital preservation is not restricted to a database of digital images obtained by scanning the documents. In this paper we are concerned with digital repositories of XML-based annotated documents from which we can extract automatically data to build a virtual museum that helps on disseminating information about RNS Agonia. Such a Web resource is crucial to support people wishing to know more about that pilgrimage, and also as a booster for tourism. The paper describes the different stages of this project, including the documents annotation process, data extraction mechanisms, the creation of a triple storage to archive the knowledge built from the sources analyzed, and the virtual museum implementation. The methodological approach devised for the project under discussion is based on the creation of an ontology that describes the RNS Agonia domain completely. The idea is to define the XML dialect, to be used in the annotation, from the ontology. Moreover the ontology will also lead the definition of the triple store used to set up the knowledge base that feeds the museum.

2012 ACM Subject Classification Applied computing → Document preparation; Information systems → Ontologies

Keywords and phrases Ontology, XML, Romaria, Pilgrimage, Digital Knowledge Repository, Triple Storage Database, Virtual Museum

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.16

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

With the increasing transition from the analog to the digital world, paper documents have become obsolete and, in consequence, all the past knowledge about the evolution ends up in oblivion. Portuguese *Romarias* are one of the main folk events that have been affected by this change. Alongside, with time, customs and traditions have been changing and adapting to this new reality.

Romaria de Nª Srª d'Agonia occurs every year in Viana do Castelo, being known as the biggest *romaria* in the country. With faith and tradition, between the devotion and the feast of a people, this is how, year after year, for more than two centuries, the RNS Agonia has been made. Like all the others, it has a strong historical past, built up from a set of events that have been expanding, particularly in the last decade.

© Sara Cristina Freitas Queirós, Cristiana Araújo, and Pedro Rangel Henriques;
licensed under Creative Commons License CC-BY 4.0
12th Symposium on Languages, Applications and Technologies (SLATE 2023).
Editors: Alberto Simões, Mario Marcelo Berón, and Filipe Portela; Article No. 16; pp. 16:1–16:16
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

All these events have been documented in articles, books, news, and even transmitted orally through generations, so that all the information is spread over different sources. Therefore, this project aims to collect these diverse materials such as newspaper news, posters, flyers, postcards, photographs and build a documentary repository. From this repository, it will be possible to gather the historical and cultural information, which will allow the recreation of those events, since the beginning of this pilgrimage up to the present day.

To guide the collection and data extraction tasks, we built an ontology – OntoRomaria – to describe the domain of festivities of RNS Agonia. This ontology is composed of the concepts that describe the domain of Pilgrimage and the relationships that link the concepts, building the triples. OntoRomaria aims to structure for archival purposes, in a triple-storage format, all the information collected about the Pilgrimage. In order to publicly display the collected information, we built a Virtual Museum. In this Virtual Museum the users can conceptually browse the knowledge repository about RNS Agonia, a pilgrimage that belongs to Minho's cultural heritage illustrating their strong roots and traditions.

This paper is organized into 9 sections. In Section 2, the RNS Agonia and the main concepts that characterize it are presented, as well as the events that take place throughout the pilgrimage. Section 3 defines two crucial concepts in this project: Virtual Museums and Ontologies. Section 4 presents the ontology built to describe the domain of RNS Agonia. Section 5 shows the construction of the knowledge repository. In Section 6 the architecture of the Virtual Museum is discussed. Section 7 describes the development of the Virtual Museum. Section 8 shows the interface of the Virtual Museum of *Romaria de Nª Srª d'Agonia*. Finally, Section 9 presents conclusions and future work.

2 Romaria de Nossa Senhora da Agonia

Romaria de Nª Srª d'Agonia occurs every year in Viana do Castelo, being known as the biggest *romaria* in the country. Pride in essence, faith, tradition and devotion are the main reasons that drive people, for more than two centuries, to plan, prepare and celebrate this christian pilgrimage [11]. This pilgrimage is devoted to *Nossa Senhora da Agonia*. *Agonia*, which translates to "agony", was chosen as a surname for Our Lady (in Portuguese, "Nossa Senhora") due to the suffering that the constant sea hazards, such as shipwrecks, caused to fishermen's families. The sea in the region was very rough, causing frequently shipwrecks, and the families of the brave fishermen used to wait for them on docks, distressed by the struggle that these men were fighting for their lives. Kneeling, they prayed fervently to Our Lady of Agony. As a way to relieve their pain, they would kneel and pray vigorously to Nossa Senhora, hoping to put an end to the agony of the uncertainty about their loved ones [9]. Nowadays, northern portuguese fishermen and their families, carry on the tradition of visiting *Nossa Senhora da Agonia* sanctuary to pray for protection for the times to come and thank for all the givens of that year.

Nevertheless, this *romaria* is not only about praying, but also about proudly exhibit old costumes and other traditions. All festivities lasts four to five days full of events. As mentioned before, over the last decade this event has become increasingly more publicized, attracting more than 1 million visitors per year, improving the region's tourism.

2.1 Main Concepts

To be able to understand the concept of this Christian Pilgrimage, it is necessary to explore a little bit all the concepts that are associated with it:

1. ***Mordoma*** is a core term when RNS Agonia is mentioned. It describes usually young and single **women** from Viana do Castelo that wear the traditional costumes ("trajes") as a way to pay tribute to the time that gave origin to the beginning of the celebration of these festivities. They are responsible for being the "messengers" by introducing and inviting everyone to gather and celebrate.
2. **Gold** is the most relevant ornament used by *Mordomas* in their *Trajes*, without it, it's incomplete. At the beggining of this traditions, gold jewelry was passed down from generation to generation in the form of inheritance. It had its greatest highlight in christian festivities, trips to the city of Viana do Castelo, where young woman would proudly exhibit all that gold on their chest, representing their families social status.
3. **Chieira** it describes pride and vanity (on the positive side of this adjectives) for wearing the costumes and gold.
4. **Traje** is a portuguese word used to reference old, traditional and historical costumes. Within the context of this pilgrimage, the costumes in question were worn by women from the old times, in the various occasions of their lives such as work, mourning, celebration, engagement, etc.

2.2 Main Events

During the RNS Agonia, five relevant events take place:

- **Parade of *Mordomas***: parade through the city where women from Viana do Castelo proudly exhibit their costumes.
- **Historical/Ethnographic parade** is an highly theatrical moment where traditional celebrations and old habits are represented by each parish of Viana do Castelo.
- **Procession to the sea**: The image of *Nossa Senhora da Agonia* is carried through the streets decorated with salt tapestries to the sea, where it is loaded in one of the flowery boats and sails part of the river Lima.
- **Folk festival**: all the vast variety and richness of *trajes* for all moments, traditional music and folk related traditions are exhibited.
- **Fireworks**: On the end of the festivities of each pilgrimage day fireworks are set off. The last day is marked by a special set of fireworks, that are let off on Eiffel's bridge and boats on the river, leading to a spectacular show of lights that also reflects on the river. This is usually called as "*Serenata*" (Serenade) because it's accompanied by soulful songs.

3 Virtual Museums and Ontologies

To develop the solution it was necessary to understand two concepts: Virtual Museum and Ontology.

On the one hand, **virtual museums** are a digital extension on the Internet, basically, a museum without walls, capable of extending ideas and concepts into the digital space, trying to reveal the essential nature of the museum. At the same time it allows people, who might never be available of visiting a certain museum in person, to reach it out [7].

On the other hand, **ontology** is an artifact used for representation, where these are intended to designate some combination of universal concepts, defined classes, and certain relationships between them [2].

Virtual museums developed using ontologies have gained increasing popularity in recent years as a way to preserve and present cultural heritage in a digital format, since an ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. One example of a virtual museum developed using ontologies is the Virtual Museum of Canada (VMC) (it was decommissioned in order to evolve out to Digital Museums Canada in 2020[1]). The VMC used the Cultural Heritage Ontology (CHO) to represent the information about the museum's collection. At a national level is possible to mention *Museu da Emigração e Comunidades* (Imigration and Comunities Museum)[2]. However, to the best of our knowledge, all of this examples developed the ontology based on annotations made on the documents. The approach presented in this paper aims to reverse the direction of this workflow and systematize the annotation of documents from an ontology already formulated for this purpose.

One of the key advantages of using ontologies in virtual museums is that they provide a standardized way of representing and organizing the information, which makes it easier to search, browse, and access the museum's content. Ontologies also enable interoperability between different virtual museums, allowing information to be shared and reused across different platforms around the world [8].

4 OntoRomaria

The first step towards designing the virtual museum was to define an ontology to describe the domain under work. In order to do it formally, it was used OntoDL+, a Domain Specific Language that was created to allow the rigorous description of ontologies by non-programmer users. OntoDL+ is not a markup language like other ontology description languages (for instance the well known standards OWL or RDF-Schema). It was designed to be easy to write and clear to read, aiming at being used with simplicity by anyone from Science to Humanities. OntoDL+ Compiler not only checks the specification to assure well-formedness and its semantic correctness, but also can convert it to other ontology representation languages [4]. That language just requires the identification of the relevant *concepts* present in the domain, the *attributes* that characterize them, the *relations* that will link concepts to define the domain meaning, and the *triples* (two concepts, the Subject and the Object, and a relation, the Predicate, that connects them) [1]. Optionally a fourth block can be added to declare the *individuals* that will instantiate the concepts, allowing for the ontology population. Listing 1 illustrates that DSL exhibiting a fragment of a OntoDL+ template.

In order to build the ontology, the first step was to read and analyze multiple documents such as newspaper news, posters, flyers, postcards, photographs, among others, highlight the relevant data present in them, and understand which concepts are more important and what are the relevant relations among them. Complementary to that bibliography research process, and in order to enrich the acquisition of knowledge concerning RNS Agonia, many conversations were hold with experts to recover the oral tradition.

After that long manual process of information retrieval, the next step was to the design and specify in OntoDL+ language the envisaged OntoRomaria to describe RNS Agonia. Listing 2 shows a short extract of the ontology developed.

---

[1] Available at `https://www.digitalmuseums.ca`

[2] Available at `https://epl.di.uminho.pt/~ritafaria/MEC/index.php`.

Listing 1 A template exhibiting OntoDL+ Syntax.

```
Ontology ontologyName
attributes { %attribs declaration block (optional)
    attribA: string,
    attribB: enum( va, vb, vc )
}
concepts { %concepts  declaration block
    ConcA[attribA, attribC:int],
    ConcB[attribB],
    ConcC, ConcD, ConcE
}
individuals { %individuals declaration block
    indA, indB
}
relations { %relations  declaration block
    % is-a, iof, pof are predefined
    relA[domain: ConcA, codomain: ConcB],
    rel1, rel2
}
triples { %triples  declaration block
    ConcA =relA=> ConcB;
    ConcB =[ rel1 => ConcC, ConcD;
             rel2 => ConcE ];
    indA =iof=> ConcA[attribA="val1", attribC=3];
    indB =iof=> ConcB[attribb=vb]
}
```

5 Virtual Museum, building the Repository

As previously said, one of the objectives of this project is to create a Virtual Museum extracting the information from a digital repository, which should be populated automatically from documents using a data extraction process. However to enable that process, the documents to be used as information sources have to be annotated in XML.

XML is a markup language used worldwide due to the fact that it is a software and hardware independent tool, easy to be read by humans and computers that allows to emphasize information in a normal text by just wrapping data between tags [10] (annotation elements that may be complemented by attributes). The XML language has no predefined tags. This meta-language just establishes a small set of rules defining what can be a tag, the way annotations open and close, and the way elements can contain other elements. So, when a new XML dialect is desired, it is up to the user to define the set of tags he needs and the element structure he desires.

Listing 3 is an example of a text (that is concerned with RNS Agonia, our specific working domain) annotated in our XML dialect.

As explained above, to start the process of document annotation, it was necessary to define the appropriate tags. The recurrent problem that information engineers face in that stage, to perform that creative task, is how to deal with that subjective concept: *appropriateness*; how to choose XML elements (tags, attributes and relations among elements) that are the most adequate for a domain. Our contribution to overcome this difficulty is the proposal of an *ontological approach*: XML elements shall be defined taking OntoRomaria (conceptual level, without individuals) into account.

Listing 2 OntoRomaria ontology in OntoDL+ (fragment).

```
Ontology OntoRomaria

concepts {
        Romaria[dataInicio: string, dataFim: string, local: string,
                data: string, sinon: string],
        Evento[local: string, data:string,
            fotografia: string, descricao: string, sinon:  string],
        Desfile_Mordomia[nr_mordomas: int],
        Mordoma[descricao: string, explicacao: string, sinon: string],
        Traje[material: string, adereco: string, cor: string,
            significado: string, tipo: string, sinon: string],
}
individuals {
        Romaria_S_Agonia,
        Traje_Mordoma, Traje_Trabalho, Traje_Feirar
}
relations {
        usa, enverga, ocorre,  desfila
}
triples {
        Romaria_S_Agonia =iof=> Romaria[dataInicio=...];
        Romaria_S_Agonia = ocorre => Evento;

        Desfile_Mordomia =is-a=> Evento;
        Desfile_Mordomia = desfila => Mordoma;
        Mordoma = usa => Traje;

        Traje_Mordoma  =iof=> Traje[material=...],
        Traje_Trabalho =iof=> Traje[material=...];
        Traje_Feirar   =iof=> Traje[material=...]
}.
```

Therefore, based on the ontology defined in the previous phase, a group of XML elements was created to markup documents concerned with the RNS Agonia.

For each concept in the ontology a XML element is created; the tag is the concept name. If the concept has attributes, the respective element will have a similar list of attributes, not required to make the annotation more flexible. For each relation, an empty element is created; once again the tag is the relation name.

Using the DTD created according to the principles above, the annotation process followed the guidelines below:

- A sentence that describes a specific 'Concept' is annotated inside the tags: "<Concept></Concept>".
- When 'Concept' is characterized by specific property 'att', it is annotated inside tags with the attribute 'att' associated to its contextual value 'txt-value': "<Concept att='txt-value'></Concept>".
- When two concepts are annotated according to the rules above, and the former is the subject of a relation to the later, which is the object (target) of such a relation, then a tag "<verb/>" is inserted between the concepts; notice that 'verb' denotes the referred relation. This strategy enables an easy extraction of the ontology triples.
- When a concept has properties whose values are long texts, such as a *description* or

explanation, these attributes are treated as an element '<Descritpion>' instead of an attribute. Care mus be taken to enclose that tag within the concept tag. Here follows an example:

```
<Processao_ao_Mar>
    <Descricao>vénia da Senhora a todos os que a veneram
    </Descricao>
</Processao_ao_Mar>
```

Listing 3 Example of an annotatted document with XML tags based on OntoRomaria.

```
<Romaria_Sra_Agonia>
    <ocorre/>
    Desfile da Mordomia
    <Desfile_Mordomia nr_mordomas='mais␣de␣meio␣milhar'> Preparam-se
        durante dias, vestem-se rigorosamente e
        saem a rua para mostrar a historia da <desfila/><Mordoma
        sinon='mulher␣vianense'> mulher vianense</Mordoma>.
        Mais de meio milhar de mulheres
        desfilam orgulhosamente, espalhando a tradicional chieira
        pela cidade no Desfile da Mordomia. Envergando os mais
        belos <usa/><Traje><Descricao>trajes das freguesias de
        Viana do Castelo</Descricao></Traje> e carregando
        <adornado_com/><Ouro quantidade='quilos'><Descricao>
        quilos de ouro ao peito</Descricao></Ouro>, com varias
        <usa/><Adereco qual='pecas␣seculares␣de␣familia'>pecas
        seculares de familia</Adereco>, estas mordomas
        representam, de forma espontanea, a maior montra de ouro
        ao ar livre do mundo.
        <desfila/><Mordoma> Trajam-se a rigor e com minucia
        na arte de bem vestir e ourar, cumprindo todos os detalhes
        herdados de geracoes anteriores, mantendo a tradicao para
        desfilar com orgulho.</Mordoma>
    </Desfile_Mordomia>
    <Desfile_Mordomia>O Desfile da Mordomia e a historia viva
        dos trajes,desde as lavradeiras as varinas, ate aos mais
        requintados trajes das mordomas, usados pelas raparigas e
        senhoras de Viana do Castelo. Muitas das pecas de ouro sao
        unicas e so sao mostradas em publico neste dia, redobrando a
        chieira das mulheres que as usam. Uma tradicao cada vez mais
        enraizada entre as mulheres de Viana do Castelo, que junta
        variasgeracoes, no mesmo quadro, para mostrarem o esplendor
        e a riqueza dos trajes e do ouro de Viana do Castelo.
   </Desfile_Mordomia>
</Romaria_Sra_Agonia>
```

Once XML elements are defined an an appropriated dialect created for RNS Agonia, it is possible to start the document annotation process using the referred dialect (as illustrated in Listing 3). This process enables the Information Extractor to perform its work based on the predefined set of tags.

6 Virtual Museum, Architecture

The architecture describes the fundamental organization of a system and its components. In our case, these components are the (human or software) modules responsible for all processing, from documents treatment to the presentation of the information extracted in a web interface.

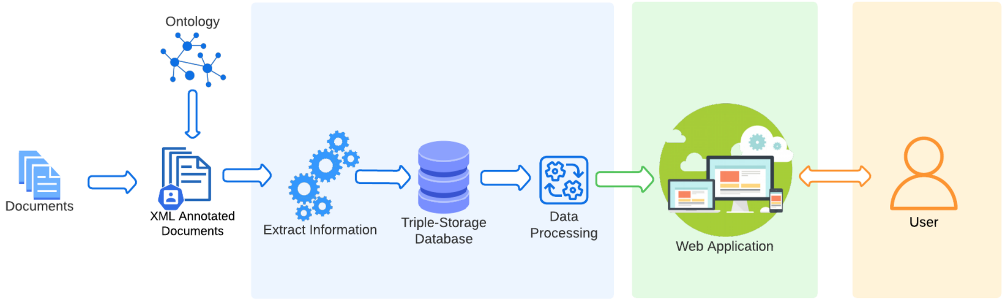


Figure 1 System Architecture.

Figure 1 allows for an easier understanding of the different components and how they interact in order to present the final product to the user. To make the image clearer, a color scheme was used to distinguish them.

The leftmost, non-colored, part refers to an external process that has to be done prior to the automatic processing. The source documents are simple articles, newspapers news, flyers, photos, etc, that were gathered in order to extract valuable information. As already explained, the ontology is used for the annotation process, specifically in tags creation, since that process involves identifying instances of important concepts and their respective relations. At this first stage the documents are manually annotated with the mentioned XML tags.

After this process, the information processing represented by the blue component begins. In this module, the Data Extractor will traverse the XML tree (the internal representation of the annotated documents) and get sets of triples: subject, predicate and object that describe in an ontological style the information obtained. In our case, these components refer respectively to the concepts of the pilgrimage and relationships with other concepts and their attributes. After extracting them they will be stored in a triple-storage database. That database will be processed to answer the requests that will have origin on the web application.

As Figure 1 shows, the green component refers to the Web application, responsible for interacting with the user. It receives the requests and converts them into appropriate queries that are send to the triple database and collects the responses to display to the users. The final user, in the orange component, has the chance to conceptually navigate over the information displayed.

7 Virtual Museum, Development

The development is divided in four phases as described in Figure 2. The first three will be described in this section and the Data presentation is discussed in Section 8.

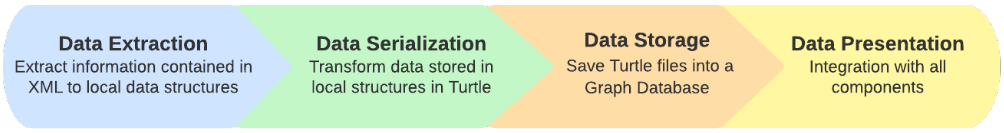

Data Extraction
Extract information contained in XML to local data structures
Data Serialization
Transform data stored in local structures in Turtle
Data Storage
Save Turtle files into a Graph Database
Data Presentation
Integration with all components

Figure 2 Stages of data processing evolution.

As mentioned before, from the designed ontology, a set of XML tags were created and adapted to manually annotate documents. Once those rules for standardizing annotations were in place, the next step was data extraction, taking them into account.

7.1 Data Extraction

To extract information from the annotated documents, using Python and the *xml.etree* library, the following data structures were created:

1. Two classes, one for each entity: concepts and relations. Within each of that class, were defined generic attributes to store the relevant information about that entity. For "concept" class those attributes were id, properties (attributes), description, explanation and relations.
   Once that classes were defined, an array of objects for each class was created to store the data. Each object in the array represents a specific instance of the entity type, and contains the values of its corresponding attributes.
   Storing the data in this way, it is easily accessible and manageable using Python object-oriented programming capabilities. For example, it is possible to search for all instances of a particular entity type, or filter the data based on certain attributes and easily add information while documents are processed.
   Overall, using this approach allows for efficiently and effectively store the extracted data from the documents, while also providing flexibility and ease of use for further analysis and processing.
2. One stack to keep track of the hierarchical relation and to inheritance among concepts. Once the closing tag of the concept in question is found, this one is popped out of that stack and the "parent" of that concept is maintained to keep analysing and storing further relations.

Given as input an annotated document like the example shown in Listing 3, the data structure produced by the extractor is:

```
Romaria_Sra_Agonia
    Relacoes:  [('ocorre', 'Desfile_Mordomia')]
    Atributos:  {}

Desfile_Mordomia
    Relações:  [('desfila', 'Mordoma'), ('destaca', 'Mordoma'), ('enverga', '
        Traje')]
    Atributos:  {'descricao': [ Desfile da Mordomia e a historia ...
            riqueza dos trajes e do ouro de Viana do Castelo.]}
Mordoma
    Relacoes:  [('usa', 'Traje'), ('usa', 'Adereco')]
    Atributos: {'sinonimo': ['mulher vianense'], nr_mordomas: ['mais de
        meio milhar'], 'descricao': [Trajam-se a rigor e com minucia na
        arte de bem vestir e ourar ... desfilar com orgulho.]}
```

```
Traje
    Relacoes: [('adornado_com', 'Ouro')]
    Atributos: {'descricao': ['trajes das freguesias de Viana do Castelo']}
Ouro
    Relacoes: []
    Atributos: {'quantidade': ['quilos'], 'descricao': ['quilos de ouro ao
            peito']}
```

7.2 Data Serialization

The main goal in terms of data extraction and storage was to use an appropriate triple-storage approach. In the context of triple-storage, RDF stands for *Resource Description Framework*. It is a standard model for data interchange on the web, using a graph-based data model to represent information about resources in the form of subject-predicate-object triples. Each triple in RDF represents a statement about a resource, where the subject is the resource being described, the predicate is the property or attribute of the resource being described, and the object is the value of the property or attribute. The triples can be stored in a variety of ways, such as in a triple-store database or as a text file in one of the RDF serialization formats like Turtle or RDF/XML, providing a flexible and inter-operable way to represent and exchange data on the web. To this end, among the many formats that are available, Turtle was chosen to serialize the data for its representation through RDF graphs [3]. In order to perform this serialization, two main steps must be performed:

1. Convert the ontology OntoRomaria into Turtle format.
2. Convert also the internal representation of the extracted data (described in Subsection 7.1) into Turtle.

Convert OntoRomaria to Turtle

Since the first step is to convert OntoRomaria to Turtle, that process was made by using a tool accessible online, OntoCnE-Nave[3] that provides an environment to visualize and explore ontologies [6]. The referred navigation tool has an extra, but actually useful, functionality that converts ontologies written in OntoDL to Turtle format (required, as input format, by that ontology processing environment). OntoRomaria was written in OntoDL+ and that could be a hindrance to the use of the tool. However, as OntoDL+ is a superset of OntoDL, that conversion was straightforward. Given OntoRomaria as input to OntoCnE-Nave, the translator produced a Turtle version proper for its visualization as a graph.

Convert Extracted Data to Turtle

After extracting data from the sources and storing it in internal structures, it was necessary to clean and prepare the extracted data and convert it to Turtle. For that purpose, it was necessary to execute the following tasks:

1. A prefix for the ontology has to be established to make IRIs shorter, allow the reuse, clarity and compatibility between ontologies, tools and software based on RDF. In this particular situation, the predefined prefix is **http://example.org/ontoromaria#**. It is important to note that the mentioned domain[4] is public and can be used in this context.

---

[3] Available at `https://computationalthinking4all.epl.di.uminho.pt/OntoCnENave`.
[4] Available at `http://example.org/`.

2. Along side with the previous prefix, it is important to establish other three that will help to specify types and labels:

   ```
   @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
   @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
   @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
   ```

3. Every concept has an IRI appropriate to the scope of the pilgrimage and two associated relationships: one defining its label according to the RDFS prefix and the other its type, according to the RDF prefix. If it has additional relations with other concepts they will be presented linking directly to those concepts IRIs, as it's possible to see:

   ```
   <http://example.org/ontoromaria#Romaria_Sra_Agonia> rdf:type
   ontoromaria:Romaria_Sra_Agonia;
       rdfs:label "Romaria Sra Agonia";
       ontoromaria:ocorre <http://example.org/ontoromaria#Procissao_ao_Mar>;
       ontoromaria:ocorre <http://example.org/ontoromaria#Desfile_Mordomia>;
       ontoromaria:descricao "Ao meio-dia,o som de ... Revista de Gigantones
       e Cabeçudos";
   ```
4. Specific cases such as dates need to be treated properly so, when the converter detects that the object of a triple is a date, and follows the usual format (yyyy-mm-dd), adds that type information type to the triple, using the XSD prefix.

Following all the guidelines above, the data stored in the internal data structures can be converted to Turtle. Therefore, it is ready to be moved to the storage phase, as described in Subsection 7.3.

Listing 4 exhibits the Turtle triples generated for the input document shown in Listing 3 and converted to the internal representation illustrated in Subsection 7.1.

Listing 4 Example of an annotatted document converted to Turtle.

```
@prefix ontoromaria: <http://example.org/ontoromaria#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.org/ontoromaria#Romaria_Sra_Agonia> rdf:type
ontoromaria:Romaria_Sra_Agonia;
                 rdfs:label "Romaria␣Sra␣Agonia";
                 ontoromaria:descricao "Ao␣meio-dia␣...␣e␣Cabecudos";
                 ontoromaria:ocorre
            <http://example.org/ontoromaria#Desfile_Mordomia>;

<http://example.org/ontoromaria#Desfile_Mordomia> rdf:type
ontoromaria:Evento;
                 rdfs:label "Desfile␣Mordomia";
                 ontoromaria:desfila
                <http://example.org/ontoromaria#Mordoma>;
                 ontoromaria:usa
                <http://example.org/ontoromaria#Traje>.

<http://example.org/ontoromaria#Mordoma> rdf:type
ontoromaria:Mordoma;
                 rdfs:label "Mordoma";
                 ontoromaria:sinonimo "mulher␣vianense";
                 ontoromaria:nr_mordomas "mais␣de␣meio␣milhar";
```

```
          ontoromaria:adereco "pecas␣seculares␣de␣familia";
                  ontoromaria:descricao "Trajam-se␣...␣orgulho.".

<http://example.org/ontoromaria#Traje> rdf:type
ontoromaria:Traje;
                  rdfs:label "Traje";
                  ontoromaria:descricao "trajes␣...␣Viana␣do␣Castelo".
```

7.3 Data Storage

After extracting the data available in the annotated sources and serializing it in Turtle, it was necessary to study the graph databases (which allow triple storage) available systems. For this, three options were considered:

- **GraphDB**[5]
- **Neo4j**[6]
- **StarDog**[7]

GraphDB, Neo4j and Stardog are all popular graph databases with similar functionality but also have some differences. Some key factors to consider when comparing these three graph databases[5] are listed below:

1. **Performance and Scalability**: GraphDB and Neo4j are known for their high performance and scalability, while Stardog has focused more on providing a rich set of semantic reasoning features.
2. **Query Language Support**: GraphDB supports the SPARQL query language, while Neo4j uses the Cypher query language, and Stardog supports both SPARQL and a subset of SQL. This factor is exclusively dependent on the programmer's choice depending on his familiarity with the languages.
3. **Reasoning and Inference**: Stardog is known for its strong support of semantic reasoning and inference, while GraphDB and Neo4j have more limited support for these features. If reasoning and inference are a critical requirement, Stardog may be the better choice.
4. **Integration with Other Tools and Frameworks**: Integration with Other Tools and Frameworks: All three graph databases have strong integration with various tools and frameworks, including popular programming languages, data integration tools, and machine learning frameworks.

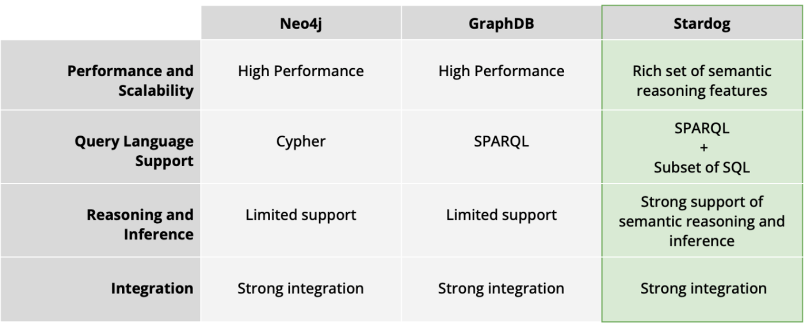
Given this general insight, summarized as a table in Figure 3, **Stardog was considered the best option** since it focus on a rich set of semantic reasoning features and supports SPARQL as query language. This is a plus since it allows to infer relations that are not explicitly declared in the database.

8 Virtual Museum of *Romaria de Nª Srª d'Agonia*

After gathering and processing all the information, the goal is to provide a conceptual navigation regarding the Romaria domain. Thus, Figures 4, 5, 6 and 7 illustrate a navigation flow through the information exposed allowing to explore concepts and their relationships.

---

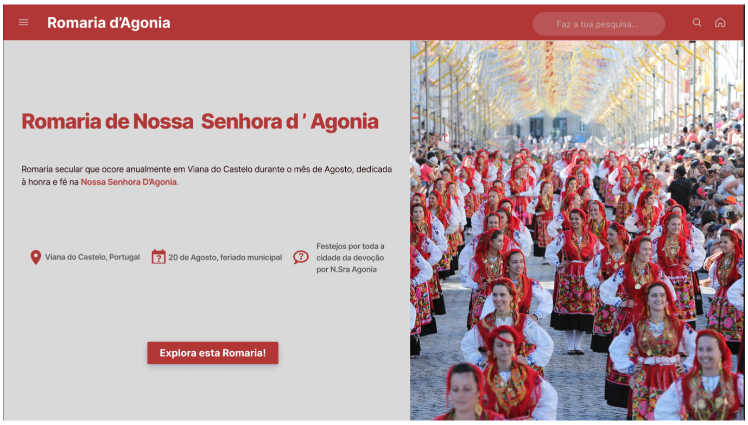
[5] For details, see `https://www.ontotext.com/products/graphdb/`
[6] For details, see `https://neo4j.com/`
[7] For details, see `https://www.stardog.com/`



	Neo4j	GraphDB	Stardog
Performance and Scalability	High Performance	High Performance	Rich set of semantic reasoning features
Query Language Support	Cypher	SPARQL	SPARQL + Subset of SQL
Reasoning and Inference	Limited support	Limited support	Strong support of semantic reasoning and inference
Integration	Strong integration	Strong integration	Strong integration

Figure 3 Summary of DB Features.

As an example, consider the case where the user wants to explore *Desfile da Mordomia*. After landing in the museum's opening page (Figure 4), the user can choose to start the navigation in order to explore the knowledge about RNS Agonia. Clicking on the main button, he will access the museum's Homepage (Figure 5) that displays some general information and the buttons that allow for the navigation to the subpages where the main concepts are detailed. Choosing the button Mordomia, the user is redirected to a page (Figure 6) that allows to explore that concept. In its explanation there are several other concepts that can also be visited. By clicking on one of these concepts, *Mordomas*, for example, another page is displayed with the information necessary for the user to understand the domain (Figure 7).



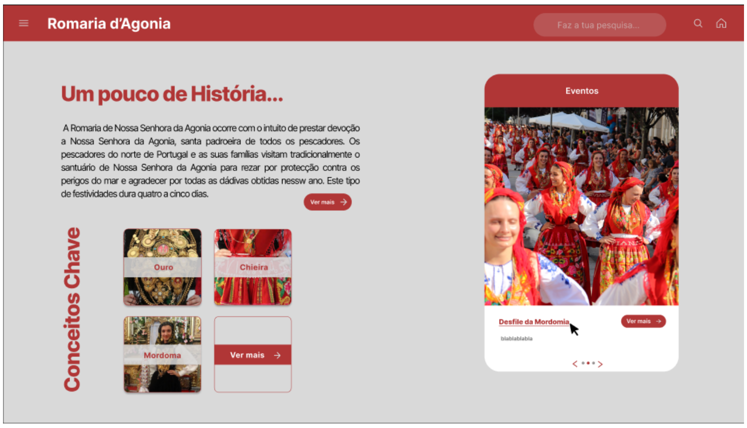
Romaria secular que ocore anualmente em Viana do Castelo durante o mês de Agosto, dedicada à honra e fé na Nossa Senhora D'Agonia.

Viana do Castelo, Portugal

20 de Agosto, feriado municipal

Festejos por toda a cidade da devoção por N.Sra Agonia

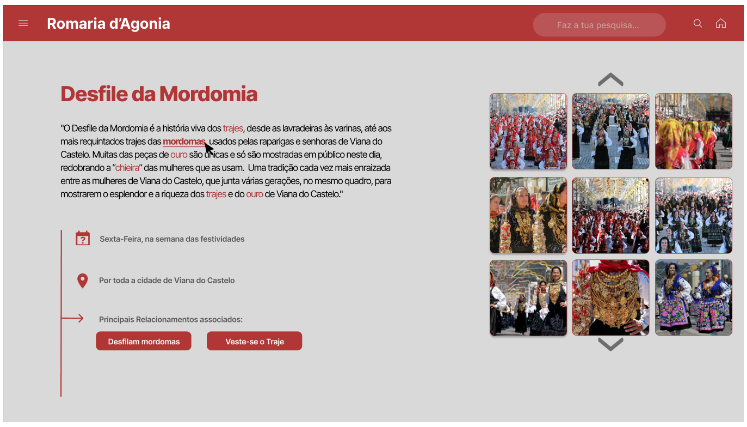
Figure 4 Landing Page on Virtual Museum for RNS Agonia.



Um pouco de História...

A Romaria de Nossa Senhora da Agonia ocorre com o intuito de prestar devoção a Nossa Senhora da Agonia, santa padroeira de todos os pescadores. Os pescadores do norte de Portugal e as suas famílias visitam tradicionalmente o santuário de Nossa Senhora da Agonia para rezar por protecção contra os perigos do mar e agradecer por todas as dádivas obtidas nessw ano. Este tipo de festividades dura quatro a cinco dias.

Figure 5 Home Page on Virtual Museum for RNS Agonia, showing the main concepts.



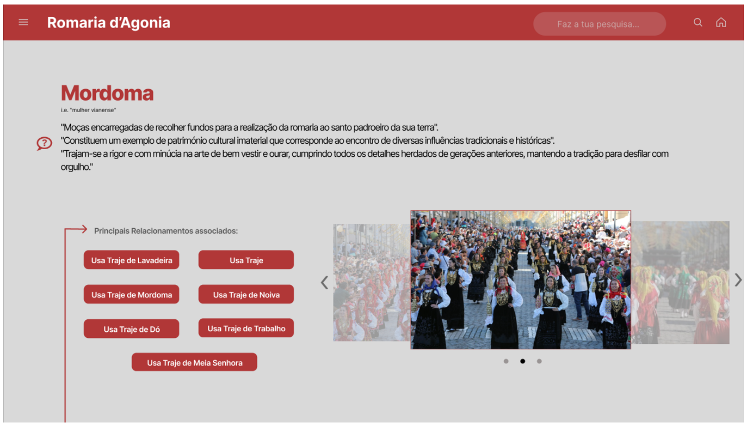
Desfile da Mordomia

"O Desfile da Mordomia é a história viva dos trajes, desde as lavradeiras às varinas, até aos mais requintados trajes das mordomas, usados pelas raparigas e senhoras de Viana do Castelo. Muitas das peças de ouro são únicas e só são mostradas em público neste dia, redobrando a "chieira" das mulheres que as usam. Uma tradição cada vez mais enraizada entre as mulheres de Viana do Castelo, que junta várias gerações, no mesmo quadro, para mostrarem o esplendor e a riqueza dos trajes e do ouro de Viana do Castelo."

Sexta-Feira, na semana das festividades

Por toda a cidade de Viana do Castelo

Figure 6 *Desfile da Mordomia* Page on Virtual Museum for RNS Agonia.



Mordoma

i.e. "mulher vianense"

"Moças encarregadas de recolher fundos para a realização da romaria ao santo padroeiro da sua terra".
"Constituem um exemplo de património cultural imaterial que corresponde ao encontro de diversas influências tradicionais e históricas".
"Trajam-se a rigor e com minúcia na arte de bem vestir e ourar, cumprindo todos os detalhes herdados de gerações anteriores, mantendo a tradição para desfilar com orgulho."

Figure 7 *Mordomas* Page on Virtual Museum for RNS Agonia.

9 Conclusion

The main goal of the project discussed in the paper is the creation of a documents repository and the development of a Virtual Museum for the ancestral Portuguese pilgrimage *Romaria de Nª Srª d'Agonia* (denoted as RNS Agonia, for short).

To achieve the established objectives it was necessary to deepen the knowledge related to the *Romaria* domain (as summarized in Section 2), and study Virtual Museum and Ontology concepts (discussed in Section 3).

RNS Agonia was described in Section 2 presenting some elements of its history, namely its origin and evolution, its festive events, main symbols and remarkable characteristics that distinguish it from all others.

After defining the problem domain using an ontology designed specifically (OntoRomaria), it was discussed how to create a XML dialect to markup documents describing the *Romaria*. To cope with the annotated texts, a data extraction process was implemented to build up a digital repository supported by a triple data store.

It was also explained how the storage, structured according to OntoRomaria, is queried to create the envisage virtual museum.

The hypothesis that we intend to prove with that project is that the web space built from the data related according to the ontology will provide a conceptual navigation over the pilgrimage information that offers to the virtual museum Visitor a chance to easily create knowledge about the *Romaria*.

The engine built to automatize the information extraction and archival, and the Web site implementation, the ontology created and the markup process are the contributions of the work here exposed.

To the best of our knowledge, there is no other system built according to the ontological approach here proposed. We are aware that the process still includes some manual tasks, but in the future we plan to automatize all the phases.

The first prototype of the referred engine and a small repository to be used as a test scenario have been developed. More tests and optimizations must be done in the near future, as well as more documents shall be annotated to increase the knowledge repository.

Of course experiments have to be designed and conducted to assess the users satisfaction and the platform effectiveness.

After that, we intend to find partners and start cooperation with local public or private institutions that can support the project in order to maintain that virtual museum to promote *Romaria de* N^a Sr^a *d'Agonia*.

References

1 Dean Allemang and Jim Hendler. *Semantic Web for the Working Ontologist*. Morgan Kaufmann, 2011.

2 Robert Arp, Barry Smith, and Andrew D. Spear. *Building Ontologies with Basic Formal Ontology*. Massachusetts Institute of Technology, 2015.

3 Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax. `https://www.w3.org/TR/rdf-concepts/`. accessed April 10, 2022.

4 Alexandre Dias. Ontodl+. `https://epl.di.uminho.pt/~gepl/GEPL_DS/OntoDL/index.html`, 2020. accessed November 8, 2022.

5 Bob DuCharme. *Learning SPARQL: Querying and Updating with SPARQL 1.1*. O'Reilly Media, 2013.

6 Daniela Fonte, Alda Lopes Gançarski, Daniela da Cruz, and Pedro Rangel Henriques. Conversor de OntoDL para Turtle. Project report, Dep. of Informatics, University of Minho, June 2022.

7 Werner Schweibenz. Virtual museums. *The Development of Virtual Museums, ICOM News Magazine*, 3(3), 2004.

8 Werner Schweibenz. The virtual museum: an overview of its origins, concepts, and terminology. *The Museum Review*, 4(1):1–29, 2019.

9 VisitPortugal. Santuário da senhora da agonia. `https://www.visitportugal.com/pt-pt/content/santuario-da-senhora-da-agonia`, 2020. accessed September, 2022.

10 W3Schools. Introduction to xml. `https://www.w3schools.com/xml/xml_whatis.asp`. accessed November 9, 2022.

11 Álvaro Campelo. *A Falar de Viana(Volume I, Série 2)*, chapter As festas da Sra. D' Agonia "Virgem do Mar, Mordoma da Terra: um contraste de rostos, uma festa de emoções!". Biblioteca Municipal de Viana do Castelo, 2012.