Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes

Simran Arora
Stanford University
simarora@stanford.edu

Brandon Yang
Stanford University
bcyang@stanford.edu*

Sabri Eyuboglu
Stanford University
eyuboglu@stanford.edu*

Avanika Narayan
Stanford University
avanikan@stanford.edu

Andrew Hojel
Stanford University
ahojel@stanford.edu

Immanuel Trummer
Cornell University
itrummer@cornell.edu

Christopher Ré
Stanford University
chrismre@stanford.edu

ABSTRACT

A long standing goal in the data management community is developing systems that input documents and output queryable tables without user effort. Given the sheer variety of potential documents, state-of-the art systems make simplifying assumptions and use domain specific training. In this work, we ask whether we can maintain generality by using the in-context learning abilities of large language models (LLMs). We propose and evaluate Evaporate, a prototype system powered by LLMs. We identify two strategies for implementing this system: prompt the LLM to directly extract values from documents or prompt the LLM to synthesize code that performs the extraction. Our evaluations show a cost-quality tradeoff between these two approaches. Code synthesis is cheap, but far less accurate than directly processing each document with the LLM. To improve quality while maintaining low cost, we propose an extended implementation, Evaporate-Code+, which achieves better quality than direct extraction. Our insight is to generate many candidate functions and ensemble their extractions using weak supervision. Evaporate-Code+ outperforms the state-of-the art systems using a *sublinear* pass over the documents with the LLM. This equates to a 110× reduction in the number of documents the LLM needs to process across our 16 real-world evaluation settings.

PVLDB Reference Format:
Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. PVLDB, 17(2): 92 - 105, 2023.
doi:10.14778/3626292.3626294

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/HazyResearch/evaporate.

1 INTRODUCTION

Organizations often seek insights trapped in heterogeneous data lakes (*e.g.* the web, corporate data lakes, and electronic health records) [10, 26, 54]. In their raw form, these data sources cannot easily support analytical queries. A long standing goal of the data management community is to develop systems that automatically convert heterogeneous data lakes into queryable, structured tables [12, 15, 47, 66, inter alia.]. In this work, we investigate whether recent large language models can help address this problem.

We study systems that take as **input** heterogeneous documents (*e.g.* HTML webpages, PDFs, text) and **output** a tabular, structured view of the documents. These systems must identify the schema and perform extraction to populate the table.

Example 1. Medical researchers frequently use data spanning electronic health records (EHR), clinical trials, knowledge sources (e.g. PubMed), and FDA reports to understand and monitor patients and treatments [8]. Consider the large collection of **FDA 510(k)** reviews for premarket medical devices, which have been the subject of multiple studies [64, 68]. Our objective is to output a table that automatically structures the attributes that are distributed in the ~20-page **PDFs**, for instance the `device classification`, `predicate device code`, and `indications for use`.

Systems designed to tackle this problem must balance a three-way tradeoff between **cost** (data lakes may hold millions of documents), **quality** (output tables should be able to accurately support an analyst's queries), and **generality** (different data lakes have different document types and structure). See Section 2 for a formal task definition and further discussion of this tradeoff.

Given the range of formats, attributes, and domains across documents, prior systems rely on simplifying assumptions (e.g. handling one document format). The majority of works focus on structuring HTML [12, 15, 25], assuming the attributes and values are at specific positions in the HTML-DOM [22, 43, 44, 67]. For unstructured text, current approaches use linguistic tools (e.g., dependency parsers) to introduce structure [15, 25, 31, 48] and then apply heuristic rules over the resulting structure to extract information. The documents in Example 1 highlight the limitations of the prior approaches: they lack (e.g. HTML) structure and, consistent with recent evaluation efforts [67], we find the SoTA approaches for unstructured text perform poorly on long semi-structured PDFs (See [5]). Some systems assume there is a human-in-the-loop, labeling data and writing heuristic rules for extraction [53, 57], while others assume access to annotated training documents from the domain [22, 43, 44]. Researchers manually annotated the reports in Example 1 [64].

* Equal contribution.

Proceedings of the VLDB Endowment, Vol. 17, No. 2 % ISSN 2150-8097.
doi:10.14778/3626292.3626294

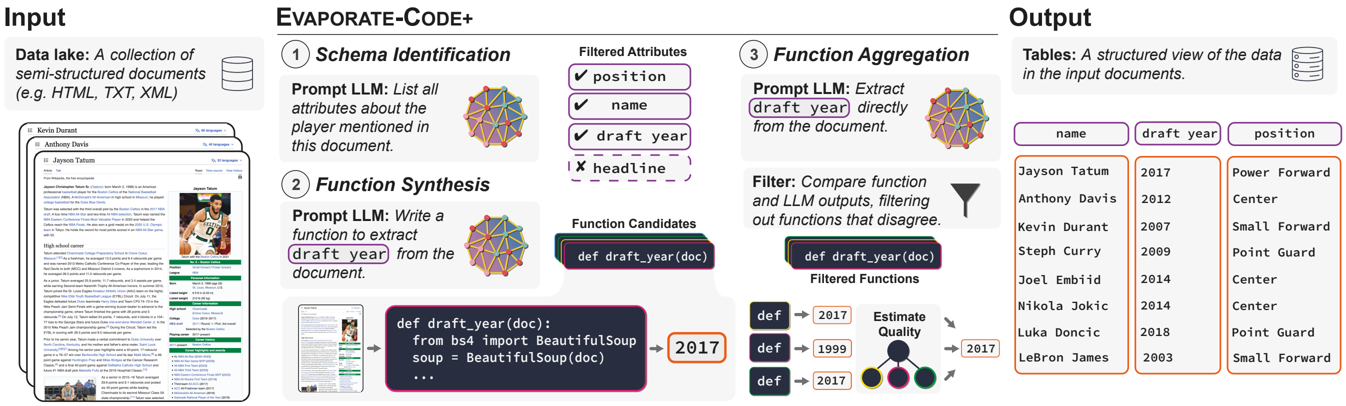

Figure 1: The user provides a collection of documents (e.g. NBA player bios) and EVAPORATE outputs a table by identifying attributes and populating columns. EVAPORATE avoids running expensive LLM inference on all documents by (1) synthesizing the key attributes from a small sample of documents and (2) synthesizing (e.g. Pythonic) functions that then are reused at scale to process documents. Because function quality is variable, EVAPORATE (3) applies an algorithm that generates many candidate functions and ensembles their extractions using weak supervision.

In this work, we explore whether we can improve generality by leveraging *large language models* (LLMs). An LLM is a deep learning model that is pretrained on broad data and can be adapted to diverse tasks, from machine translation to data wrangling [14, 46]. At inference time, the models take as input a natural language task description termed a *prompt* [11, 14] and generate a natural language response. See Section 2.3 for more background on LLMs.

EVAPORATE. (Section 3) We present EVAPORATE, a system that uses LLMs to produce structured views of semi-structured data lakes. Our evaluation spans 16 real-world settings from movie and university websites to *FDA 510(k)* reviews [22, 32, 34, 37, 43, 64, 68].

The user inputs a collection of documents and EVAPORATE automatically identifies the schema and performs extraction to populate the table. Our implementation requires *no customization, training, or human effort* to support the diverse evaluation settings. We propose two fundamental strategies for implementing this interface, identifying a tradeoff between their cost and quality:

(1) EVAPORATE-DIRECT (Figure 2) The LLM directly extracts values from documents.
(2) EVAPORATE-CODE (Figure 4) The LLM *synthesizes code* that is then applied to process documents at scale.

EVAPORATE-CODE is cheap, but underperforms EVAPORATE-DIRECT by 24.9% (13.8 F1 points) averaged across our evaluation settings. We thus seek a new code synthesis approach. We present EVAPORATE-CODE+, which achieves better quality than direct extraction. Our insight is to synthesize many code snippets for extraction and ensemble their outputs using weak supervision.

Direct Extraction (Section 3.1). Our first implementation, EVAPORATE-DIRECT, applies a *single prompt* (included in [5]) to each document in the input. The prompt instructs the LLM to both identify the schema and extract values. Remarkably, we find that in some settings, with a single prompt and no task specific modifications, performance is already competitive with state-of-the-art systems that rely on domain specific assumptions and training.

However, this implementation is very expensive. LLMs are optimized for interactive, human-in-the-loop applications (*e.g.* ChatGPT) [65], not high-throughput data processing tasks [56]. The number of tokens processed by an LLM in EVAPORATE-DIRECT grows *linearly* with the size of the data lake. As of March 2023, applying OpenAI's models to the 55 million Wikipedia articles would cost over \$110k (`gpt-3.5`, \$0.002/1k tokens) and \$1.1M (`text-davinci-003`, \$0.02/1k tokens) dollars [1, 49]. There are *billions* of webpages on the broader Internet [35] and the facts change over time. For instance, NBA players are added to Wikipedia, a player's `team` changes after trades, and the `points per game` metric changes after every game. Data processing is a *routine expense* (repeated by multiple data analysts), not a one-time cost [55].

Code Synthesis (Section 3.2). *Can we produce the structured table using a sublinear pass of the LLM over the documents?* We propose EVAPORATE-CODE, which splits the task into two sub-tasks: (1) identify the table schema and (2) extract values. This view allows us to exploit the distinct *redundancies* of each sub-task that occur when running LLM inference on every document:

(1) *Schema Generation.* In order to identify a schema, we only process a small sample of documents with the LLM. This succeeds because there is redundancy in the attributes mentioned across documents. For e.g., in Example 1, most reports mention a `predicate device name`.
(2) *Function Synthesis.* We prompt the LLM to synthesize (*e.g.* Pythonic) *functions*, that can be applied at scale across the documents. This works because of redundancy in the formatting of attribute-value pairs. For e.g., the FDA 510(k)s use the consistent format "`Predicate device name: k`".

The number of tokens processed by the LLM in EVAPORATE-CODE is *fixed* and does not grow with the size of the data lake (as illustrated in Figure 3), addressing the cost issues of EVAPORATE-DIRECT. However, the LLM synthesizes variable quality information extraction functions. The extractions are up to 14 points worse in Pair F1 score than those produced using EVAPORATE-DIRECT.

Code Synthesis + Aggregation. (Section 3.3) To improve quality while keeping costs low, we propose EVAPORATE-CODE+. Studying the synthesized functions, we observe some only work for a narrow slice of documents, while others exhibit syntactic and logical errors. To reduce variance, we synthesize many candidate functions, then estimate their quality and aggregate their extractions using *weak supervision*. This builds on our work [4], which broadly applies weak supervision to prompting for the first time.

Weak supervision (WS) is a statistical framework for modeling and combining noisy sources with varied coverages without any labeled data [53, 62]. However, WS is typically applied over *human-generated* functions while our setting consists of *machine-generated* functions. This presents issues when attempting to apply existing WS tools. (1) WS theoretically assumes all noisy sources are better than random performance (50% accuracy), yet 40% of our generated functions are *below 25%* (Section 3.2). (2) WS attempts to deploy functions that achieve high quality on narrow slices of data (high precision), and allow the function to *abstain* on data external to the slice (low recall). While humans can express when functions should abstain, the machine-generated functions do not contain this logic. To handle the *open* WS setting, we introduce a new algorithm for ensembling the functions (Algorithm 1).

We summarize our overall contributions as follows.

(1) **Our system offers new capabilities for the long-studied structured view generation problem.** Existing systems require in-domain training and handle limited document formats (e.g. HTML [16, 22, 24, 43]). EVAPORATE requires no training and succeeds on different document formats (HTML, PDF, TXT) off-the-shelf. (Section 4).
(2) **We study a new tradeoff space between direct extraction and code synthesis for data tasks.** EVAPORATE *asymptotically* reduces the number of tokens that the LLM needs to process to generate the outputs. At 10k documents per evaluation setting, this amounts to a 110x cost reduction. Further, prior works using LLMs for data tasks require users to manually write prompts [46]. EVAPORATE is built with task-agnostic prompts that generalize across settings.
(3) **We present an algorithm and theoretical analysis for applying weak supervision to open-ended functions and extraction tasks.** Although EVAPORATE-CODE is more efficient, EVAPORATE-DIRECT achieves significantly higher quality. Using our algorithm, EVAPORATE-CODE+ outperforms EVAPORATE-DIRECT, which directly processes every document, by 10.1 F1 points (18%) (Table 3).
(4) **We extensively validate the system on 16 data settings from 5 domains and 3 data formats, and across 4 LLMs.** (1) EVAPORATE outperforms the SoTA *learned* baseline systems by 3.2 F1 points (6%) when generating tables (both schema generation and extraction) end-to-end, and 6.7 F1 points (10%) on the extraction step. (2) EVAPORATE-CODE+ achieves a 10.1 F1 point increase over EVAPORATE-DIRECT, using text-davinci-003. (3) Across four unique LLMs we show the relative quality of EVAPORATE-DIRECT vs. EVAPORATE-CODE+ remains consistent.

We define the problem in Section 2. We present EVAPORATE in Section 3, evaluations in Section 4, and related works in Section 5.

2 PRELIMINARIES

We first define the problem setting and system desiderata.

2.1 Problem Setting

We study the problem of constructing a structured view (*i.e.* database table) of a set of semi-structured documents (*e.g.* HTML, PDF, TXT). Formally, we define the problem as follows:

- **Input:** User provides a set of n semi-structured documents $D = \{d_1, d_2, ...d_n\}$ (*e.g.* A collection of FDA 510(k) reviews for premarket notification submission for medical devices).
- **Output:** System outputs a table defined by a set of attribute names $A = \{a_1, a_2, ...a_m\}$ (*e.g.* a_1 =`indications for use`, a_2=`classification`) and a set of n extracted records for $R = \{r_1, r_2, ...r_n\}$, one per document, where r_i is an m-tuple (*e.g.* r_1 = ("fracture", "x-ray")).

Unlike prior work which proposes systems that rely on manual labeling [57] or manual prompt tuning [46, 59], we aim to develop *automated* solutions, which require no user effort.

Measuring System Quality We compare the generated table (A, R) to a manually curated "ground-truth" table $(\hat{A}, \hat{R})$. The coverage of an attribute refers to the fraction of documents that include the attribute and its value. Following prior work, we prioritize attributes with high *coverage*, which tend to be useful for analysis [16, 18]. We measure agreement between the tables using Pair F1. For additional details on our evaluation setup, see Section 4.3.

2.2 System Desiderata

Current systems for producing structured views are limited in their generality, cost/flexibility, and quality/usability [16, 18, 48, 67]. Here we review the existing systems.

Generality. *The ideal system will generalize across document formats and domains, without manually engineered rules or task-specific training.* This is important because the input documents D could focus on any imaginable topic or use any file format [67]. Existing systems featurize documents by tagging the named entities (NER), dependency parse tree, and part-of-speech (POS), and train a model to predict whether a span of text is a useful fact [39]. Unfortunately, the performance of the parse, NER, and POS tags drastically degrade on semi-structured data (e.g. HTML elements) and longer sequences of text (i.e. full documents) [67]. We provide detailed error analysis [5]. A specialized class of systems focuses on processing semi-structured web HTML documents by leveraging the HTML DOM tree as features [13, 16, 22, 25, 44, inter alia.]. However, the systems thus do not support other document formats.

Cost. *The ideal system will enable users to manage a cost-coverage tradeoff, rather than requiring them to extract "all-or-nothing".* The existing systems are built to extract *all* possible facts in the documents, without prioritizing important attributes or allowing the user to influence what is extracted [21, 67]. Processing every line of every document can be expensive. To mitigate this, the user can define the attributes of interest then apply a closed IE system for extraction, however this requires upfront human effort. **Desiderata**: The ideal system will enable users to manage a cost-coverage tradeoff, rather than requiring them extract "all-or-nothing".

Quality. *The ideal system will output a table* (A, R) *with full columns (i.e. high-coverage attributes) and accurate, consistently formatted extractions.* Existing OpenIE systems commonly extract tuples in unnormalized forms directly from documents [21]. This can make the resulting extractions difficult to use for analysis, requiring advanced systems or user-defined post-processing code for resolving subject, objects, and predicates to a canonical form [15].

2.3 Background on Large Language Models

In this section, we provide background on *large language models* (LLMs), which are central to our work.

Definition 1 (Large Language Model). *A machine learning model,* $\mathcal{F}$*, trained on a self-supervised task (e.g. next word prediction) over a massive corpus of text [29]. Language models can be used to generate new text based on provided context. For example:*

$$\mathcal{F}(\textit{All that glitters}) \rightarrow \textit{is not gold.}$$

Numerous studies have demonstrated LLMs capable of solving new tasks without updating any model parameters, a phenomenon termed *in-context learning* [2, 14, 46]. Specifically, these studies show that when passed an appropriate description of the task, the model often generates text completing the task.

Definition 2 (Prompt). *A natural language task-specification used to elicit a particular generation from an LLM. Prompts often include demonstrations of the task. For example, the prompt below elicits the translation of the word cheese into French:*

$$\mathcal{F}(\underbrace{\textit{Translate. Eng: hello, Fr: bonjour; Eng: cheese, Fr:}}_{\textit{Prompt}}) \rightarrow \underbrace{\textit{fromage}}_{\textit{Generation}}$$

Examples of prompts used in this work are provided in Figures 2 and 4. All prompts used in the system are provided in [5].

3 EVAPORATE: A PROTOTYPE SYSTEM POWERED BY LANGUAGE MODELS

We introduce Evaporate, a prototype system that uses LLMs to materialize a structured view of a heterogeneous, semi-structured data lake. Compared to prior systems, which rely on manual labeling [57] or tuning prompts to a domain [46], Evaporate exposes a remarkably *general* interface: the user inputs documents and the system automatically outputs a structured view of those documents, without any domain specific training or prompt customization.

Overview. We instantiate the Evaporate interface with three different implementations. We can feed every document to the LLM and prompt it to extract values directly (*direct extraction*, Figure 2), or feed a small sample of documents to the LLM and prompt it to write *code* to do the extraction (*code extraction*, Figure 4). In Section 3.1 and Section 3.2, we describe baseline implementations of these two strategies, Evaporate-Direct and Evaporate-Code. We find that these two implementations tradeoff cost and quality. Then, in Section 3.3, we propose a code extraction implementation that uses weak supervision to improve quality and retain low cost.

Prompt Management Evaporate applies a set of task-agnostic prompts, which are all provided verbatim in the technical report [5]. These tasks are not modified for different tasks. Within the system, the prompts are Python f-strings, with placeholders for inputs chunks of text from the particular dataset being processed. The LLM is prompted with the formatted strings. We use a caching tool that we helped develop called Manifest [50] to store input and completion pairs from the LLM prompting in a local SQLite database, where keys are the prompt-inputs and values are the completions . Therefore, if users repeatedly run the system on the same dataset, they do not incur the LLM inference costs again.

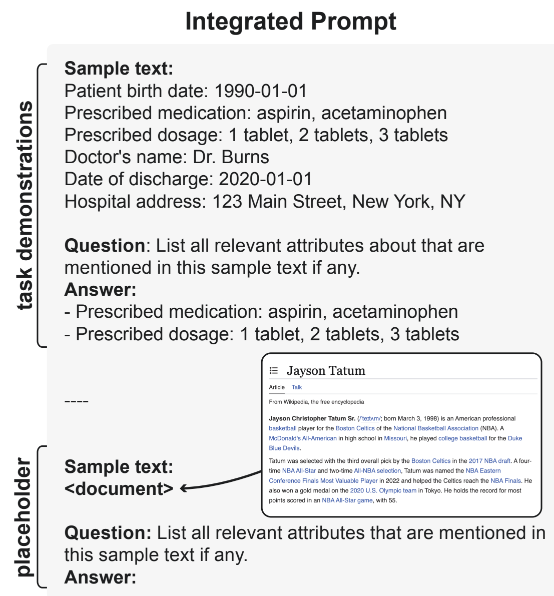

Figure 2: Prompt for Evaporate-Direct structured. The prompt template, which includes placeholders for in-context examples and and the inference example (i.e., data lake documents), is applied to each document in the data lake.

3.1 Evaporate-Direct

In this section, we describe a simple *direct extraction* implementation, Evaporate-Direct that applies a single prompt template to every document. This prompt template, which is included in Figure 2, instructs the LLM to both identify the schema and extract values (see [5] for the full prompt). It consists of a few in-context examples that are general, *i.e.* are not customized to a particular format, domain, or document.

Below we discuss how we (1) manage long documents that cannot fit in the LLM's context window, (2) process the LLM's textual outputs, (3) prioritize the most useful attributes according to principles described in prior work [16].

Managing long documents. The input to Evaporate is a file path to raw documents, which can be several pages long. For instance the Medical FDA reports in Example 1 are ~20 pages long. However, the underlying Transformer architecture of modern LLMs is limited to processing a fixed number of tokens (*e.g.* a few thousand tokens), referred to as the *context window*, during each inference call. Evaporate therefore splits the raw documents such that each piece is within the context window. Each chunk is inserted into the prompt in turn as shown in Figure 2.

Processing text outputs. Language models output open ended text so the last step is to convert this to a usable table. To facilitate this data transformation, we can specify formats in our prompt demonstrations to encourage the LLM to organize the output in a similar structure. For instance, the demonstration in Figure 2 specifies a list format with `<attribute>: <value(s)>` per entry. EVAPORATE outputs in this format can be de-serialize into a table.

Prioritizing common attributes. The list of extracted attributes and values can contain the niche attributes for specific documents, whereas a common database design principle is to capture the high frequency attributes [15]. Therefore EVAPORATE takes the union of attributes outputted across documents and ranks by frequency to enable prioritizing head attributes.

Analysis. We analyze this *direct extraction* implementation, EVAPORATE-DIRECT, along the axes of our three desiderata. Results processing the documents with EVAPORATE-DIRECT are reported in Table 3 and are discussed in detail in Section 4.

Overall, the quality matches or exceeds the baseline systems (described in Section 4), on 8 of the 16 settings. This is surprising given the simplicity — i.e. EVAPORATE-DIRECT uses *one* fixed prompt to process *all 16 settings*. However, fundamental cost limitations impede the real-world deployment of this approach.

However, the high cost of this implementation limits its applicability to large, recurring workloads. The number of tokens processed by the LLM scales linearly with the size of the data lake, $O(n)$. Data lakes can contain *billions* of documents [35, 47]. Further, in most organizations, data processing is not a one time cost. Data lakes are dynamically changing, so EVAPORATE-DIRECT would need to be *repeatedly* applied.

3.2 EVAPORATE-CODE

In this section, we present EVAPORATE-CODE, which significantly reduces cost compared to EVAPORATE-DIRECT. Here, we perform schema identification separately from value extraction, which allows us to exploit fundamental differences between the sub-tasks to reduce cost. In schema identification, we find that we only need to process a small sample of documents because attributes are consistent across documents. On the other hand, in order to extract values, we must process every document. However, the ways in which values appear across documents (*i.e.* their relative positions in the document) tend to be consistent, meaning the extraction logic is consistent across documents.

The two steps of the decomposed implementation are:

(1) **Schema synthesis.** (Section 3.2.1) We observe that the attribute outputs contain relatively consistent `<attributes>`, even though the `values` differ from document to document. To exploit this redundancy, EVAPORATE-DIRECT prompts an LLM to analyze a small sample of documents to identify attributes for the output schema For example, given a sample of the Medical Device FDA Reports, the LLM outputs a devices table with attributes like `"510(k) number"`.

(2) **Function synthesis** (Section 3.2.2). We observe consistencies in how attributes are embedded across documents. E.g., the `510(k) code` in the FDA documents always starts with the letter “k” and the `player position` attribute is always in the HTML “infobox” element in NBA player Wiki pages. A researcher would likely exploit such redundancies when manually scraping the documents for analysis. In EVAPORATE-CODE, we propose to use the LLM to automatically synthesize a data-lake-specific suite of *functions*, that can then be applied at scale to process many documents.

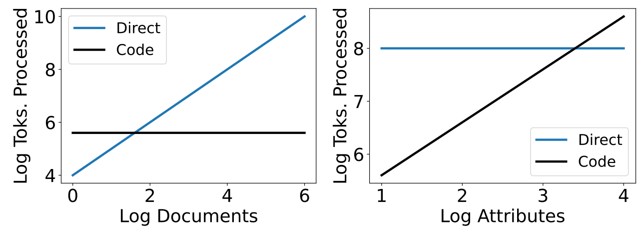

Figure 3: Tradeoffs between processing the documents via direct prompting (Direct) versus code synthesis (Code). For small data lakes and large numbers of attributes, Direct is sufficient. As the number of documents grows, Code is orders-of-magnitude more efficient. Left is evaluated at 10 attributes, Right at 10K documents, assuming 10K tokens per document.

Next, we provide details for each sub-task.

3.2.1 Schema Synthesis . EVAPORATE first uses an LLM to identify attributes $A = \{a_1, a_2, ...a_m\}$ for the output schema.

Generating candidate attributes Concretely, we sample a set $\tilde{D}$ of $k << n$ documents from D. For each, we prompt the LLM to extract the most useful attributes from the document as in EVAPORATE-DIRECT. Recall this yields a set of attributes ranked by how frequently they were extracted across documents. We retain attributes that are explicitly mentioned in the document to ensure provenance in schema identification.

Re-ranking candidate attributes Because EVAPORATE now identifies the attributes from a small set of documents, we observe that EVAPORATE's ranking is noisier than when every document was processed in EVAPORATE-DIRECT, i.e. an important attribute may be selected by the LLM a few times amongst the k documents. Thus, we show the LLM a union of extracted attributes and prompt it to identify the most useful attributes (prompt in [5]). The frequency-based rank is upweighted if the attribute is in the LLM output.

3.2.2 Function Synthesis. Given the attributes $A = \{a_1, a_2...a_m\}$, the objective of EVAPORATE-CODE's second phase is to extract the values of the attributes for each document $d_i \in D$. Our key insight, as discussed, is that attribute-values are expressed in similar ways from document to document. To exploit this, instead of processing every document with the LLM to extract values for attribute a_i, we propose to use the LLM to *generate code* that can then be reused to process many documents.

Figure 4 shows an EVAPORATE function synthesis prompt. The in-context examples show pairs of text snippets and functions to extract an attribute of interest. EVAPORATE searches the data lake via a simple keyword search for document portions that mention a_i, and includes this in the prompt. EVAPORATE synthesizes functions for attributes following the rank-order of attributes derived during schema synthesis. This means that values of the most relevant (and

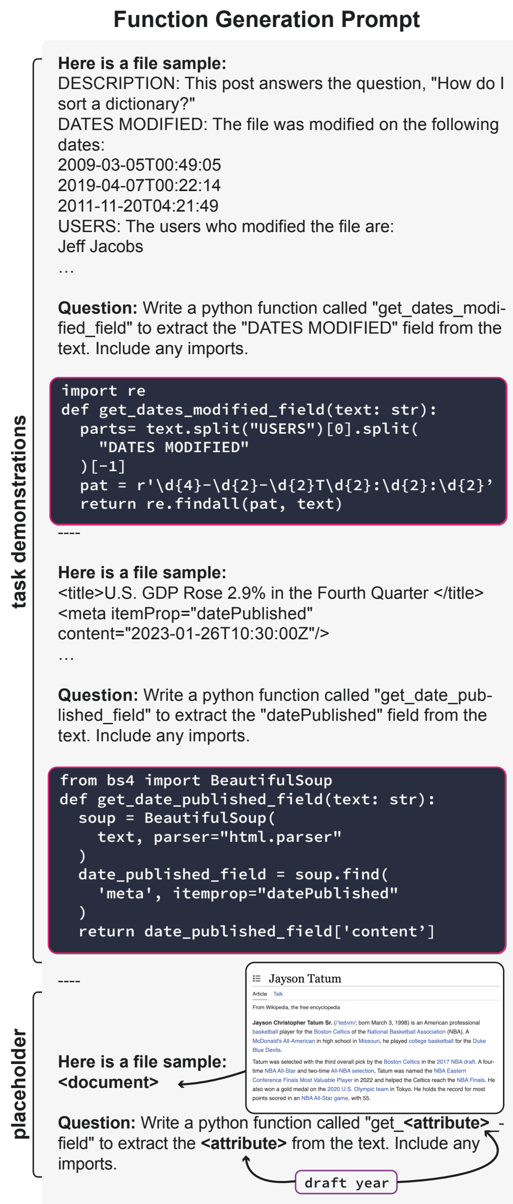

Figure 4: A representative prompt for function synthesis, containing two data lake agnostic in-context examples.

frequent) attributes as determined by Evaporate are extracted first. The user can stop the synthesis when desired.

Analysis. We briefly analyze the Evaporate-Code implementation along the axes of our three desiderata. Results processing the documents with Evaporate-Direct are reported in Table 3 and are discussed in detail in Section 4.

Cost. Figure 3 demonstrates the asymptotic differences in cost between Evaporate-Direct and Evaporate-Code. Evaporate-Code is asymptotically more efficient as a function of the number of documents: the number of LLM calls required with function generation is proportional to the number of attributes, not the number of documents. The crossover point is at 40 documents. Meanwhile, Evaporate-Direct has the potential to extract multiple attributes from the in-context document per inference call, while Evaporate-Code requires generating new functions for each attribute. Thus, the cost of Evaporate-Code grows with the number of attributes, while the cost of Evaporate-Direct approach is constant. The crossover point is at 2,500 attributes (Figure 3).

Quality. The tables generated by Evaporate-Code are on average 21.9 pair F1 points worse than those produced using Evaporate-Direct on the SWDE datasets (Table 2). This suggests that there is a cost-quality tradeoff between the two implementations, since Evaporate-Code is much cheaper.

3.3 Evaporate-Code+

In this section we discuss an extension of Evaporate-Code, which enables significant quality improvements while keeping costs low. This implementation, which we call Evaporate-Code+, synthesizes *many* candidate functions and ensembles their extractions using weak supervision. We decompose the task into three parts:

(1) **Schema identification.** (Section 3.2.1) Same as in Evaporate-Code.
(2) **Function synthesis.** (Section 3.3.1) Same as in Evaporate-Code, except instead of generating a single function per attribute, we generate many *candidate functions*. Below we describe techniques to encourage diversity among candidates.
(3) **Function Aggregation.** (Section 3.3.2) The synthesized candidate functions have varying qualities and coverages, making them unreliable. We then introduce a weak supervision (WS) based algorithm to aggregate over their different predictions for the attribute values across documents.

3.3.1 Synthesizing Diverse Candidate Functions. We find that the quality of LLM-generated functions varies significantly depending on the document chunk and in-context examples used in the prompts. To address the variability in function quality, we adopt the strategy we previously proposed in Arora et al. [4]. This strategy curates multiple diverse prompt templates for the same task (i.e. multiple function generation prompts in the style of Figure 4) and prompts the LLM with each in turn to produce a diverse set of *function candidates* $F = \{f_1, f_2, ... f_k\}$.

Evaporate permits the use of multiple function generation prompts. We use P_A and P_B (included in [5]) in this work. P_A has zero in-context examples and a task description that encourages the LLM to use regex. P_B has two in-context examples and a task description that encourages the LLM to import and use any Python library. We find that neither consistently outperforms the other. P_A produces higher quality functions on 69%, 45%, 60%, 91%, and 31% of attributes on the 8 SWDE Movie, 5 SWDE University, FDA reports, Enron, and Wikipedia player pages settings respectively. Designing a single "perfect" prompt can be challenging so Evaporate aggregates results from multiple prompts.

3.3.2 Aggregating Candidate Functions. Next, we discuss how to combine the aggregations of the candidate functions.

Background: Methods for Unsupervised Aggregation Because we lack ground truth labels in our setting, it is not possible to directly evaluate the quality of the candidate functions. A popular unsupervised aggregation strategy is to take the Majority Vote (MV) across function outputs [63]. Formally, MV treats the functions as independent of one another and assigns equal weight to all function outputs. However, the functions are not of equal quality — over 40% of synthesized functions result in less than 25 Text F1 in extraction quality. Therefore, EVAPORATE uses weak supervision (WS), a popular standard statistical framework for modeling the accuracies and correlations between noisy sources of information without any labeled data [27, 53]. In WS, we learn a *label model* that is parametrized by the accuracies and correlations of the candidate functions. WS is widely used in industry [53].

Unfortunately, existing WS setups make the following assumptions that do not apply in our setting. The standard setup assumes *human-designed* functions while our setting uses *machine-generated* functions that output non-standardized extracted text.

(1) **Assumption 1: Functions will abstain on examples where they do not apply [27, 53].** The attribute value returned by a function for a document could be `null` for two reasons: (1) the attribute does not exist in the document (e.g. a Wikipedia page may be missing a college attribute since the players did not attended college) or (2) the attribute exists but the function was not sophisticated enough to extract it (e.g. the `product code` attribute could start with a lowercase "k" of uppercase "K" in FDA reports, but the particular function is only designed to extract for lowercase, resulting in empty strings for uppercase documents). Note that in (1), if the function outputs a value, it has low precision and we would want to ignore the function. Note that in (2), the function has high precision and ideally our system learns to utilize such functions *selectively*. Unfortunately, it is challenging to determine whether the function outputs `null` for reason (1) or (2) in our setting, whereas in the traditional WS setup with human-provided functions, humans specify this logic directly (e.g. "If the email has a URL, "vote" that it contains spam, otherwise abstain" [58]).

(2) **Assumption 2: Functions are correlated with the gold label y at better than random performance [27, 53].** While this is reasonable when functions are human-designed, EVAPORATE uses machine-generated functions. We find 51% of generated functions are below 50 Text F1.

(3) **Assumption 3: Weak supervision is typically applied to tasks with well defined classes in a classification setting [27, 53].** In our case, the output of the functions are extracted text, and thus there is a virtually unconstrained output space of possible extractions that vary from document to document (e.g. NBA players have varied `date of birth` values). The *number* of unique extractions collected by the functions can also differ across documents.

We propose the following approach to be able to leverage WS. Let D_{eval} be a small sample of documents from the data lake $\mathcal{D}$. We have the set of generated functions F and LLM $\mathcal{F}$.

Handling function abstentions. To estimate the probability that an empty output from a function is an abstension, we propose to measure the fraction e of the D_{eval} documents for which $\mathcal{F}$ extracts a value. Intuitively, when e is high, our prior should be that the attribute appears in a large fraction of documents, so we should assume functions are *abstaining* when they output empty values. When e is low, the attribute appears in few documents, so we should assume the functions are *predicting* empty values. We can use e to guide both our function evaluation and downstream aggregation. Note that it is possible for $\mathcal{F}$ to abstain or hallucinate values, affecting the estimate of e.

Algorithm 1 Function Aggregation (from EVAPORATE-CODE+)

1: **Input:** Documents $\mathcal{D}$, candidate functions F, LLM $\mathcal{F}$.
Output: Predicted extractions $\hat{y}_i, ..., \hat{y}_n$ for documents.
2: **Collect sample predictions** Sample $\mathcal{D}_{eval} \subset \mathcal{D}$ and apply the functions $f_j \in F$ and LLM $\mathcal{F}$ to obtain $\hat{y}_{ij}$ and $\hat{y}_{i\mathcal{F}}$ for document d_i.
3: **Handle abstensions**: For empty $\hat{y}_{ij}$, we need to determine if they represent *function abstensions* or *predictions* that d_i has no-value for the attribute. Use $\mathcal{F}$ to decide between cases: compute e as the fraction of $d_i \in \mathcal{D}_{eval}$ with non-empty $\hat{y}_{i\mathcal{F}}$
4: **Score functions**: Compute a score $\hat{a}_j$ for f_j using metric function $m(\cdot)$ based on e.
if $e > \tau$ **then**
$\hat{a}_j = \sum_{i=1}^{i=n} m(\hat{y}_{i\mathcal{F}}, \hat{y}_{ij}) | \hat{y}_{i\mathcal{F}} \neq \emptyset$
else
$\hat{a}_j = \sum_{i=1}^{i=n} m(\hat{y}_{i\mathcal{F}}, \hat{y}_{ij})$
5: **Filter low quality functions** Remove $f_j \in F$ with $\hat{a}_j \leq 0.5$ to create F'.
6: **Collect votes** Apply $f \in F'$ to all $d_i \in \mathcal{D}$ to collect "votes" for the attribute-value in d_i. Post process empty votes as *abstensions* or no attribute *predictions* depending on e.
7: **Aggregation** Use weak supervision to obtain the final prediction $\hat{y}_i$ given the function votes $\{\hat{y}_{ij} | f_j \in F'\}$. =0

Handling functions with worse than random quality. We propose to utilize the extractions fro $\mathcal{F}$ on a small set of documents D_{eval} (e.g. we use $|D_{eval}| \leq 10$) as an estimate of the ground truth extractions for those documents. We can then estimate the quality, $\hat{a}_j$, of function f_j by comparing its outputs against the outputs of $\mathcal{F}$ on document $d_i \in \mathcal{D}_{eval}$. If we are in the low e regime, we should evaluate the outputs on all $d \in \mathcal{D}_{eval}$. In the high e regime, we should evaluate the outputs on only the $d \in \mathcal{D}_{eval}$ for which f_j extracted a value. We finally filter f_j if $\hat{a}_j \leq 0.5$, where 0.5 derives from the typical WS assumptions [27, 53, 61].

Note that $\mathcal{F}$ is an LLM with its own error rate e, affecting the estimate $\hat{a}_j$. We theoretically study the impact of e on the label model learned via WS. We provide the proof in [5].

Proposition 1: We have m functions with empirical accuracies $\hat{a}$, evaluated against noisy labels with error rate e, the function accuracies estimated by the weak supervision label model are $\tilde{a}$, and the measured error is below some threshold ϵ. Then, if each function labels a minimum of

$$n \geq \frac{1}{2(\gamma - \epsilon - e)} \log(\frac{2m}{\delta})$$

datapoints, the weak supervision label model will succeed in learning accuracies such that $||a^* - \tilde{a}||_\infty < \gamma$ with a probability $1 - \delta$.

Handling unconstrained output spaces. The k generated functions can produce $[0..k]$ unique prediction votes for a single unlabeled document d_i, and the number of unique votes can differ from document d_i to d_j. Therefore, for each $d_i \in \mathcal{D}$, we bucket the unique votes and take the b buckets representing the most frequently occurring votes. The votes for functions that outputted values outside the top-b are marked as abstensions. If the number of unique votes is $< b$, placeholder values are inserted into the top-b. Finally, as the "classes" differ across documents, we introduce a constraint to the objective function encouraging the class-conditional accuracies to be equal.

After addressing these assumptions, we can leverage prior approaches to aggregate the noisy extractions from the function candidates into higher-quality extractions as in [4, 53]. Under WS, the output of each function is viewed as a "vote" for the true label and the objective is to construct a latent graphical model to account for the varied accuracies and correlations amongst the functions, without access to any labeled data. Our aggregation method is summarized in Algorithm 1.

Analysis. We briefly analyze the Evaporate-Code+ implementation along the axes of our three desiderata. Results processing the documents with Evaporate-Code+ are reported in Table 3 and are discussed in detail in Section 4.

Cost. As with Evaporate-Code, the number of tokens processed by the LLM in Evaporate-Code+ is fixed with respect to the number of documents. Figure 3 demonstrates the asymptotic differences in cost between Evaporate-Direct and Evaporate-Code. The number of tokens that must be processed by the LLM grows only by a constant factor: the number of function candidates generated. The user can set this number to balance cost and quality.

Quality. Of the three implementations, Evaporate-Code+ produces the highest quality tables. Evaporate-Code+ outperforms Evaporate-Direct by 12.1 F1 points (22%) on average, while using far fewer computational resources. Using function aggregation leads to an improvement of 25.1 F1 points over Evaporate-Code.

4 EVALUATIONS

We now evaluate Evaporate, validating the following claims:

- **Function synthesis enables asymptotic cost reductions for processing data with LLMs.** There has been significant recent interest in developing various data management applications with LLMs [17, 36, 40, 46]. Prior work directly processes data with the LLM. Evaporate-Code+ reduces the number of tokens the LLM needs to process by 110x relative to Evaporate-Direct.
- **Function synthesis + aggregation results in higher quality than direct extraction.** Despite the fact that Evaporate-Direct processes each document with the LLM directly, Evaporate-Code+ performs 10.1 F1 points (18%) better on average. Based on comparisons with Evaporate-Code, which only synthesizes one function, we show that function aggregation is key in enabling the improvements.
- **Evaporate achieves higher quality than state-of-the-art baselines, while exposing a more general interface.** Evaporate-Code+ expresses tasks via merely *six* natural language prompts (all provided in [5]) and uses no training. Yet, it exceeds SoTA systems by 3.2 F1 (6%) points when generating tables from scratch and 6.7 points (10%) when extracting pre-defined gold attributes. Meanwhile, it supports a broader range of settings than any of these baselines.
- **The identified tradeoffs hold across language models.** We evaluate on four models from three unique providers [6, 42, 49]. We find Evaporate-Direct and Evaporate-Code+ remain competitive in quality across LLMs.

4.1 Experimental Setup

We primarily evaluate Evaporate on the end-to-end task of *structured view generation*. For the purpose of comparison to prior work, we also evaluate on the sub-task of *closed information extraction*. We first define these tasks, their metrics, and the baselines. We then provide implementation details for Evaporate.

Structured view generation task. This captures the end-to-end task of identifying the schema and populating the output table. This task is often discussed as a vision system [16], and given the difficulty of this task, there are limited comparable works. We therefore compare to the closest line of work, OpenIE systems, where the task is to extract all facts from documents [7, 48]. We compare to two sets of baselines: (1) Deng et al. [22], Lockard et al. [43, 44] for HTML-specific OpenIE, and (2) Kolluru et al. [39] for generic unstructured text. The former models explicitly use the HTML-DOM tree structure to process the page, assuming attribute values are leaf nodes, and explicitly train on documents from the domain of interest. The latter class of systems first label sentences using linguistic tools (*i.e.* dependency parsers, part of speech taggers, and named entity taggers), and fine tune LLMs over these features to perform the task [67].

Metrics. The standard metric is Pair F1 [22, 43], an F1 score applied to the predicted vs. gold sets of tuples of the form (document ID d_i, attribute a_j, value r_i, j). The tuple must exactly match a tuple in the ground truth to be marked correct. Since Evaporate ranks the attributes and generates functions in this order, for fair comparison, we report OpenIE scores for all tuples up to k attributes, where k is the number of gold attributes for the setting. We note that the prior systems extract all-or-no tuples, in contrast.

Closed information extraction task. This captures the setting where the user provides a pre-defined schema and Evaporate is used to populate the table. We compare to state-of-the-art approaches for closed IE including: (1) Deng et al. [22], Lockard et al. [43, 44] for HTML-specific ClosedIE and (2) Clark et al. [19], He et al. [33] for generic unstructured text. The former models explicitly use the HTML-DOM tree structure to process the page, assuming attribute values are leaf nodes, and explicitly train on documents from the test domain. The latter are pretrained LLMs that have been fine tuned on massive amounts of labeled `(attribute, value)` pairs [52]. We report ClosedIE results using the Text F1 metric on a value-by-value basis across each document.

Evaporate Implementation Details. In the following experiments, we instantiate Evaporate with currently popular, LLM APIs. Experiments in Sections 4.3 and 4.4.1 use `text-davinci-003` from OpenAI. In Section 4.4.2, we evaluate additional LLMs from three model providers. For experiments, we use 10 sample documents per data lake for the schema synthesis, function synthesis, and function verification. We apply Algorithm 1 over the top-10 scoring functions that are synthesized for each attribute and data lake. The prompts remain constant across data lakes and models. In [5], we provide ablations that show how the system's quality changes as we vary the number of sample documents and top-k functions.

When the measuring cost for alternate implementations of Evaporate, we compute total number of tokens processed by the LLM to perform the end-to-end task (*i.e.* the sum of the number of tokens in the prompt and model generation). We use this metric because the wall-clock time and dollar cost of a model fluctuate, but both should be proportional to the number of tokens processed.

4.2 Evaluation Settings

We evaluate Evaporate on 16 settings representing a range of real-world data lakes. First, we use a benchmark suite of 13 Movie and University websites to compare Evaporate to state-of-the-art information extraction systems [22, 32, 43]. Next, to evaluate on more unstructured data (*i.e.* non-HTML) we turn to: **Enron** a corporate email corpus that has been analyzed in over three thousand academic papers [3, 34, 37], **FDA 510(k)** reviews for premarket notification submissions for medical devices, which have been the subject of multiple important research studies [64, 68], and **NBA** Wikipedia pages for NBA players, which include more complex HTML than the existing benchmarks [22]. We release the benchmarks and provide additional details in [5]. Here we briefly describe the properties we aim to study with each setting:

(1) **Benchmark Suite: SWDE Movies & Universities** SWDE is the standard benchmark for document-level IE in prior work [22, 32, 43, 44, inter alia.]. There are 8 sets of webpages for Movies (e.g. IMDB) and 5 sets of webpages for Universities (e.g. US News). For each website, the benchmark contains 1063-2000 pages and annotations for 8-274 attributes. We use SWDE to compare to the state-of-the-art and test on a range of attribute types, e.g. simpler Movie "runtime" through complex Movie "cast" and popular Movie "director" through infrequent "second assistant director".
(2) **Complex HTML: NBA** As SWDE attributes always occur in separate leaf nodes of the HTML-DOM tree, we use NBA Player Wikipedia pages to evaluate on more complex HTML. E.g., the NBA draft attribute contains the draft round, year, pick number, and team by which the player was selected. We evaluate on 100 randomly selected player pages (spanning the 1940s-present) and 19 attribute annotations.
(3) **Unstructured Text: Enron and FDA** We observe a lack of existing benchmarks for document-level IE over unstructured text — intuitively, this setting has been challenging with prior generations of models due to the lack of *any* grounding structure whatsoever (*i.e.* recall current systems rely on HTML-DOM elements or sentence-level NER, dependency, and POS tags). We turn to the Enron and FDA settings described above. The Enron setting contains 15 gold attributes and 500k documents. The FDA setting contains 16 gold attributes and 100 PDF documents, which are up to 20 pages long, randomly sampled from FDA 510(k).

Dataset Protocols Because the baselines we compare against on SWDE require training data, they perform website-wise cross validation (*i.e.* train on some websites and evaluate on others). They do this for several combinations such that every website appears in the evaluation set. Evaporate, in contrast, does not require any training data. We simply evaluate Evaporate on all websites so that our method is evaluated on the same examples as the baselines.

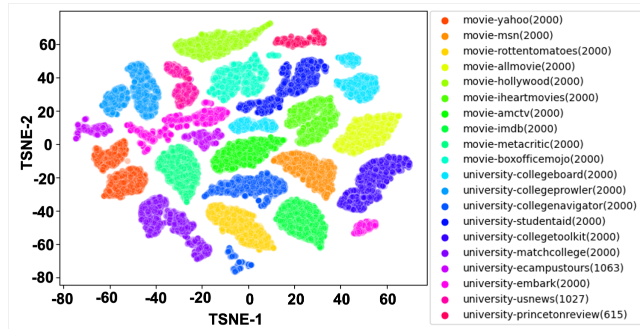

Figure 5: T-SNE Visualization of documents in the SWDE dataset. T-SNE is performed on the first 16 principal components of TF-IDF vectors. Colors indicate the source website.

We process each dataset with Evaporate separately, to match the protocol used by the baselines [22]. However, we may not have access to the sources of the documents in a real-world data lake – they may be mixed together. Using a standard TF-IDF vectorizer and K-means clustering to the mixture of documents, we verify we can perfectly recover the document sources without any labeled data or supervision (Figure 5, details in [5]). Intuitively, clustering semi-structured data may be simple due to the rich formatting.

4.3 Comparing Evaporate to Baselines

First we validate that Evaporate outperforms baselines, defined in Section 4.1, both in generality (i.e. the flexibility to support data from different domains and formats) and quality. We then compare the efficiency of Evaporate-Code+ vs. the baselines.

4.3.1 Quality and generality comparisons.

Systems for semi-structured text. Shown in Table 2, Evaporate outperforms the state-of-the-art on SWDE. We compare to the metrics reported the baseline works. Recall Evaporate uses no training whatsoever and can be applied across document formats (HTML, PDF, TXT). In contrast, the baselines are limited to HTML and explicitly perform supervised learning using labels from webpages within the Movie and University domains respectively [22, 44]. E.g., Deng et al. [22] assumes attribute values are the leaf-nodes of the HTML-DOM tree and thus does not work on non-HTML.

The baseline systems restrict scope to attributes that are specifically mentioned in the HTML <body> text, even though attributes are frequently mentioned in the HTML header (e.g. within <title> elements) and tags (e.g. <a href='year/2012'>). We validate Evaporate can identify and extract attributes mentioned anywhere in the document. We extend the SWDE benchmark to include the attributes scattered throughout the full HTML and find Evaporate achieves 52.2 and 49.0 on Movies and University respectively on the more challenging setting. We release the new annotations.

Table 1: Quality of EVAPORATE-CODE+ evaluated on ClosedIE in Text F1 and OpenIE in Pair F1 using text-davinci-003.

Source (Format)	CLOSEDIE	OPENIE		
	F1	*R*	*P*	*F1*
FDA (TXT)	80.1	62.0	68.1	64.9
Enron Emails (TXT)	93.3	80.3	94.6	86.9
Wiki NBA (HTML)	84.7	55.7	88.2	68.2
SWDE Movie (HTML)	79.5	48.5	71.0	56.8
SWDE University (HTML)	73.7	50.9	71.4	59.0
Average	**82.3**	**58.9**	**78.5**	**66.7**

Table 2: Comparisons to state-of-the-art on ClosedIE in Text-F1 and OpenIE in Pair F1. The baselines train on in-domain documents, while EVAPORATE uses no training [22].

System	SWDE MOVIE		SWDE University	
	Closed	*Open*	*Closed*	*Open*
ZeroShot Ceres [44]	-	50.0	-	50.0
RoBERTa-Base	49.3	35.6	36.6	38.0
RoBERTa-Structural	47.7	39.9	46.5	42.3
DOM-LM [22]	71.9	54.1	68.0	55.2
EVAPORATE-DIRECT	**84.4**	45.2	72.6	53.8
EVAPORATE-CODE	55.0	33.0	40.5	22.2
EVAPORATE-CODE+	79.5	**56.8**	**73.7**	**59.0**

Systems for unstructured text. We are not aware of strong baselines that apply beyond HTML document formats. The most relevant baseline is the OpenIE6 system for performing OpenIE over any unstructured text from Kolluru et al. [39]. We find the system only handles well formatted sentences and struggles to extend to heterogeneous data types. We find that even when documents contain full sentences, the system extracts an extremely large set of relations and does not enforce consistent extractions across documents. For instance, on a sample FDA 510(k) document, OpenIE6 extracts 427 relations with 184 relations having a confidence level at 0.99. We include a detailed error analysis in [5].

4.3.2 Efficiency comparisons. We compare the efficiency of EVAPORATE with the (estimated [28]) 175B parameter OpenAI model vs. the baseline, which uses a pretrained 125M parameter RoBERTa model [22], decomposed in terms of pretraining, fine-tuning, inference, and parameter (memory) cost. Using FLOPS as reported in the OpenAI reference work Brown et al. [14], for data settings with n documents and m attributes, the costs are:

- **RoBERTa** The model is 125M parameters, total pretraining FLOPS is 1.50E+21, and inference FLOPS per token is 2 $\times$ Number of Parameters, which is 0.250 GFLOPS. The total inference cost is computed by
$$n \times \frac{\text{tokens}}{\textit{document}} \times 0.250 \text{ GFLOP}$$
- **GPT3-175B** The model is 175B parameters, total pretraining FLOPS is 3.14E+23, and inference FLOPS per token is 2$\times$Number of Parameters, which is 350 GFLOPS. The total inference cost is computed by
$$m \times P \times \frac{\text{tokens}}{\textit{chunk}} \times 350 \text{ GFLOP}$$
where P is the number of prompts per attribute. Note that for our evaluated implementation $P \approx 10c$ since function generation is performed on 10 documents.

EVAPORATE uses a model with 1,400x more parameters and 300x higher pretraining cost. However, users of the baselines likely need to locally fine-tune and host the models. The inference costs of the baseline and EVAPORATE on our datasets are in the same order of magnitude (summarized in [5]). The extended cost comparison and analysis are provided in [5]. Users should select a method depending on the data setting, i.e. the number of documents and attributes. We note that EVAPORATE allows users to tradeoff quality and efficiency by changing the underlying language model to smaller variants.

4.4 Comparing Implementations of EVAPORATE

This work proposes a fundamental tradeoff space between directly processing data workloads with LLMs vs synthesizing code that does the processing. We first discuss the tradeoffs for a fixed LLM (`text-davinci-003`), which is the current best-in-class LLM [41] (Section 4.4.1), and next across a range of LLMs trained by three distinct model providers (Section 4.4.2).

4.4.1 Tradeoffs between EVAPORATE Implementations. As detailed in Section 3.2, the base routine ("EVAPORATE-DIRECT") in EVAPORATE entails directly processing documents with the LLM, while the optimized routine ("EVAPORATE-CODE") synthesizes functions for processing. Next we evaluate these along our desiderata.

Generality is maintained. LLMs take text as input and provide text as output — this unified natural language interface means EVAPORATE-DIRECT and EVAPORATE-CODE can ingest any document format without additional engineering. *Critically, our results with EVAPORATE require no user effort, no training whatsoever, and no customization when applied to the 16 different settings.*

Asymptotic cost reduction. Figure 3 demonstrates the asymptotic differences in cost between directly processing the data lake with EVAPORATE-DIRECT vs. with EVAPORATE-CODE+. (Figure 3 Left) EVAPORATE-DIRECT is asymptotically more efficient as a function of the number of documents in the data lake. The number of LLM calls required with function generation is proportional to the number of attributes to be extracted, not the number of documents. The crossover point is at ~40 documents.

(Figure 3 Right) EVAPORATE-DIRECT can extract multiple (*i.e.* every) attribute in the in-context documents in a single inference call, while EVAPORATE-CODE+ synthesizes new functions for each attribute. Thus, the cost of function synthesis grows with the number of attributes, while the cost of EVAPORATE-DIRECT is constant. The crossover point is at ~2,500 attributes.

Empirically across our settings, EVAPORATE-CODE+ realizes a 110x average reduction in the number of tokens the LLM needs to process (assuming 10k documents per setting and 378$\times$ given the true benchmark sizes) in the number of tokens the LLM must process compared to EVAPORATE-DIRECT (Table 3). Further, data lakes are constantly changing and functions can be reused while EVAPORATE-DIRECT would need to be re-run, multiplying the cost.

In runtime, we observe that the generated functions are efficient in processing the documents. For example, over the 9,500 function

Table 3: Quality (OpenIE Pair F1) and cost (number of tokens processed by the LLM) for producing the structured views. We compare the direct prompting and code synthesis implementations using text-davinci-003. Evaporate-Code+ is evaluated on the full datasets, while Evaporate-Direct is evaluated on a randomly sampled proportion of documents due to the cost (20% of FDA and Wiki, 2% of SWDE, and 0.0002% of Enron, given the respective sizes).

Source (Format)	Evaporate-Direct			Evaporate-Code+			Relative Performance	
	Quality F1	Cost / 10K Documents Tokens (M)	Cost / 10K Documents Cost ($)	Quality F1	Cost / 10K Documents Tokens (M)	Cost / 10K Documents Cost ($)	Quality	Cost Reduction
FDA (TXT)	45.5	145.6	2,900	62.8	1.9	38	+17.3	77x
Enron Emails (TXT)	93.8	21.2	425	86.9	0.6	12	-6.9	35x
Wiki NBA (HTML)	44.8	650.1	13,000	68.2	3.0	60	+23.4	217x
SWDE Movie (HTML)	45.2	282.9	5,660	56.8	2.3	46	+11.6	123x
SWDE University (HTML)	53.8	190.1	3,800	59.0	1.9	38	+5.2	100x
Average	**56.6**	**258**	**5,157**	**66.7**	**1.9**	**39**	**+10.1**	**110x**

runs (from 95 functions evaluated on 100 documents each) in the FDA 510(k) setting, we find that the average time to run one function over one document is 0.00025s on a 2 CPU machine.

Improved quality and reliability. Even though Evaporate-Direct directly processes each document with the LLM, Evaporate-Code+ surprisingly performs 10.1 F1 (18%) better (Table 3).

What are the failure modes of Evaporate-Direct? The method yields inconsistent generations. On the Medical FDA report setting:(1) The LLM misses an average of 4.4 attributes that are present in the gold schema (27.5% of gold attributes) per document. Among the gold attributes that are missed, all *are extracted* in at least one document. (2) Further, the LLM outputs an average of 9.7 attributes or values that are not explicitly mentioned in the documents. (3) Finally, attributes are reworded in diverse ways across documents — the attribute `classification` is extracted in 4 different ways across the sample of 10 documents (i.e. "classification", "device classification", "regulatory information", missing). Since the error modes are quite varied, it is unclear how to improve quality.

Why does Evaporate-Code+ improve quality? We validate that our Algorithm 1 for selecting and aggregating functions leads to the quality improvements over Evaporate-Direct.

Synthesizing diverse functions We find that using diverse prompts helps address the lack of reliability in function synthesis. To synthesize functions, Evaporate-Code+ uses a prompt template that includes one-to-two in-context examples and a placeholder for the inference example, i.e. document text (Figure 4). We can produce multiple prompts in the template by swapping the in-context examples or sampling more documents (change the inference example). We find both means of increasing diversity benefit quality:

- **In-context demonstrations** Our implementation (Table 1) instantiates two prompts by swapping in-context demonstrations , P_A and P_B. Quality using P_A or P_B alone is 8.5 and 8.0 F1 points worse than using both to synthesize functions on SWDE Movie and SWDE University respectively.
- **Inference documents** Using three versus five sample documents in the prompts for Evaporate-Code+, the ClosedIE and OpenIE quality improve by 6.8 F1 points (9%) and 6.5 F1 points (14%) respectively, averaged across the 16 settings.

Estimating function quality using the LLM. In Table 4, we first evaluate the two unsupervised aggregation baselines in prior work off-the-shelf: Majority Vote (MV) and Weak Supervision (WS) [4, 53, 63]. Next we measure the effect of filtering functions and handling abstentions as proposed in Algorithm 1.

Table 4: Quality under alternate approaches of aggregating the synthesized functions. Baselines are in the left columns: Majority Vote (MV) and Weak Supervision (WS). Components of Algorithm 1 are in the right columns: "Abstain" accounts for abstensions and "Filter" filters low quality functions.

Source	MV	WS	WS Filter	WS Abstain + Filter
FDA (TXT)	52.9	51.1	55.0	62.8
Enron Emails (TXT)	81.4	82.7	86.9	86.9
Wiki NBA (HTML)	59.5	64.9	68.4	68.2
SWDE Movie (HTML)	44.3	46.3	56.6	56.8
SWDE University (HTML)	42.7	43.5	57.3	59.0
Average	**56.2**	**57.7**	**64.8**	**66.7**

In Table 4, we observe WS with filtering provides a consistent boost across settings compared to WS — 7.1 F1 point higher average quality and up to 13.8 F1 points on the SWDE University setting. Additionally handling abstensions leads to a 1.9 F1 point increase in average quality over WS with filtering, with up to 7.8 F1 points on the FDA setting. Qualitatively, accounting for abstensions is helpful when attributes are expressed in diverse ways across documents, which is not applicable to all settings such as Enron. These results highlight the importance of Evaporate-Code+'s aggregation approach for the system's overall reliability. Without Algorithm 1, quality does not improve over Evaporate-Direct.

4.4.2 Understanding the Tradeoff Space across Varied Language Models. The are an increasing number of LLMs being made available. These models are trained by various providers each using distinct protocols [41]. To understand whether the tradeoffs we identified hold for different LLMs, we evaluate Evaporate using three additional LLMs from three different providers: (1) **GPT-4** [49], (2) **Anthropic Claude-V1** [6], and (3) **Jurassic Jumbo-2-Instruct** [42]. Results are summarized in Table 5.

Overall results. The quality with `gpt-4` is comparable to that obtained using `text-davinci-003`. Both the Evaporate-Direct and Evaporate-Code+ quality decrease with `claude` and `jumbo`, consistent with the results of large-scale benchmarking efforts [41], however the *relative* quality of the two implementations are similar to Table 3. Both appear to remain competitive in quality and the quality of the approaches appear to increase together.

Table 5: OpenIE (Pair F1) results evaluating EVAPORATE using alternate LMs from three model providers. For cost reasons, we apply EVAPORATE-DIRECT to samples of 10 documents each. For fair comparison, we report the score of EVAPORATE-CODE+ on the same sample instead of the full set of documents. k is the number of gold attributes for the setting.

Source (Format)	EVAPORATE-DIRECT					EVAPORATE-CODE+					SCHEMA ID
	FDA	*Wiki*	*Movie*	*University*	*Enron*	*FDA*	*Wiki*	*Movie*	*University*	*Enron*	*F1@k*
OpenAI GPT-4 [49]	59.2	40.5	35.1	56.1	92.7	57.5	61.4	54.9	57.2	85.5	67.3
Anthropic Claude-V1 [6]	45.1	20.6	27.5	44.3	88.1	44.4	33.5	38.7	30.4	84.7	69.0
Jurassic Jumbo-2-Instruct [42]	25.9	0.0	13.3	29.2	90.3	1.2	0.0	20.6	18.6	85.7	62.3

We find the precision of EVAPORATE-CODE+ remains high across models. Algorithm 1 helps EVAPORATE filter the low quality functions and if this eliminates all the candidate functions, the attribute is excluded from the output table. We find that when an attribute *is* included in the output, it has high precision, consistent with Table 1 where the precision with `text-davinci-003` is almost 20 points higher than the recall. The average precision scores corresponding to EVAPORATE-CODE+ in Table 5 are 70.9 (`gpt-4`), 67.6 (`claude`), and 50.9 (`jumbo`) using EVAPORATE-CODE+ and in contrast are 61.9 (`gpt-4`), 55.1 (`claude`), and 49.9 (`jumbo`) using EVAPORATE-DIRECT, emphasizing a precision-recall tradeoff between approaches.

Understanding the errors. Overall, EVAPORATE relies on versatile reasoning capabilities (*i.e.* to identify the schema and extract attribute values *directly* from *noisy* provided context, and the ability to synthesize code) and excitingly, the results validate that these capabilities co-exist within multiple model families. We investigate which of the required reasoning capabilities contributes to lower quality in comparison to `text-davinci-003`. We find that the schema synthesis step plays a small role. Considering the top ranked schema attributes according to EVAPORATE, we measure the average F1@k between the predicted and gold sets of attributes, where k is the number of gold attributes per setting. The average F1@k for `text-davinci-003` is 71.9, and the right-hand column of Table 5 shows the alternate models perform comparably.

We find the two main sources of errors are (1) the inability to generate a function for particular attributes, and (2) occasionally, low quality *direct* extractions in particular cases (e.g., `claude` may respond "I'm not sure, please give me more information." in a ChatBot style, when prompted to extract an attribute value). The models we evaluate are optimized for ChatBot applications [38].

5 RELATED WORK

Structured Querying of Heterogeneous Data. Converting heterogeneous data to structured databases is a long standing data management problem [12, 16, 31, inter alia.]. In contrast to systems for knowledge base construction (KBC) or closed information extraction (IE) [57, 66], which assume there is a predefined schema and focus on populating the database according to the schema, the setup we focus on relies on OpenIE. OpenIE is the task of extracting useful facts without access to a predefined ontology (i.e. the types or categories of facts to be extracted) [7, 20]. Given the breadth of input documents, the ability to construct a schema and populate the corresponding database on-the-fly is useful.

Existing systems for this problem introduce assumptions about the data-domain [22, 51, 60], file-format (e.g., XML files) [30], or the syntactic patterns of useful facts [12, 15, 25, 31, 39, 45, 48, 67]. For instance, in early systems, Cafarella et al. [15] focuses on facts expressed as triples (two entities with a descriptive string of their relationship in between) with hypernym "is-a" relationships between the entities. The recent deep learning based systems (1) require domain- and document-format specific training, (2) focus on reasoning over sentences, in contrast to long documents, and (3) rely on high quality linguistic tools (e.g. dependency parse, POS, NER) to help introduce structure over unstructured text [39, 67].

For the narrower problem of generating structured views from *web data* [15, 23, 25], the current state-of-the-art approaches use (1) distant supervision to train site-specific extraction models [43] (**domain specific**), and (2) rely on assumptions about where in the HTML-DOM attributes and values are located (**format specific**) Deng et al. [22], Lockard et al. [44]. We investigate the feasibility of a domain and document-format agnostic approach.

Language Models for Data Management. Given the recency of in-context learning, there are few works exploring the benefits for data processing. Most closely related, Chen et al. [17] presents a system for querying heterogeneous data lakes with in-context learning. The proposed approach involves processing *every document* with the LLM to extract values of interest. We propose an alternate approach and tradeoff space for processing data with LLMs.

Other recent work applies language models for tasks such as data wrangling [46] or code generation for SQL queries [59]. Unlike EVAPORATE, supporting various data formats, these prior approaches focus on relational data only and design manual prompts to demonstrate high quality.

Data Programming. We build on work in data programming and weak supervision [53]. EVAPORATE automatically generates functions rather than using human-designed functions. We use WS on open ended tasks in contrast to the classification tasks considered in the prior work on automated WS [9, 61]. We show how to use the LLM to handle abstensions and filter low quality functions.

6 CONCLUSION

We propose EVAPORATE, a system that uses LLM in-context learning to generate structured views of semi-structured data lakes. We identify and explore a cost-quality tradeoff between processing data directly with an LLM versus synthesizing and aggregating multiple code snippets for data processing. We present an algorithm and theoretical analysis for applying weak supervision to aggregate the code snippets. The code-based approach aims to exploit the structural redundancies that occur in corpora of semi-structured documents. We validate EVAPORATE on 16 unique data settings spanning 5 domains and 3 document formats, considering the cost, quality, and generality of the system. Our study highlights the promise of LLM-based data management systems.

REFERENCES

[1] April 2023. Wikipedia Statistics. https://en.wikipedia.org/wiki/Special:Statistics

[2] Monica Agrawal, Stefan Hegselmann, Hunter Lang, Yoon Kim, and David Sontag. 2022. Large Language Models are Few-Shot Clinical Information Extractors. *The 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2022).

[3] Simran Arora, Patrick Lewis, Angela Fan, Jacob Kahn, and Christopher Ré. 2023. Reasoning over Public and Private Data in Retrieval-Based Systems. *Transactions of Computational Linguistics (TACL)* (2023).

[4] Simran Arora, Avanika Narayan, Mayee F. Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. 2023. Ask Me Anything: A simple strategy for prompting language models. *International Conference on Learning Representations (ICLR)* (2023).

[5] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. (2023). https://www.dropbox.com/scl/fi/3gt3ixdbvp986ptyz5j4t/VLDB_Revision.pdf?rlkey=mxi2kqp7rqx0frm9s7bpttwcq&dl=0

[6] Amanda Askell, Yushi Bai, Anna Chen, Dawn Drain, Deep Ganguli, T. J. Henighan, Andy Jones, and Nicholas Joseph et al. 2021. A General Language Assistant as a Laboratory for Alignment. *arXiv:2112.00861v3* (2021).

[7] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew G Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. *IJCAI* (2007).

[8] David W Bates, David M Levine, Hojjat Salmasian, Ania Syrowatka, David M Shahian, Stuart Lipsitz, Jonathan P Zebrowski, Laura C Myers, Merranda S Logan, Christopher G Roy, et al. 2023. The Safety of Inpatient Health Care. *New England Journal of Medicine* 388, 2 (2023), 142–153.

[9] Benedikt Boecking, Willie Neiswanger, Eric Xing, and Artur Dubrawski. 2021. Interactive weak supervision: Learning useful heuristics for data labeling.

[10] Rideout J. R. Dillon M. R. Bokulich N. A. Abnet C. C. Al-Ghalith G. A. Alexander H. Alm E. J. Arumugam M. et al. Bolyen, E. 2019. Reproducible, interactive, scalable and extensible microbiome data science using qiime 2. In *Nature biotechnology*.

[11] Rishi Bommasani, Drew A. Hudson, E. Adeli, Russ Altman, Simran Arora, S. von Arx, Michael S. Bernstein, Jeanette Bohg, A. Bosselut, Emma Brunskill, and et al. 2021. On the opportunities and risks of foundation models. *arXiv:2108.07258* (2021).

[12] S. Brin. 1998. Extracting patterns and relations from the WorldWide Web. In *WebDB*.

[13] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. 2013. Extraction and integration of partially overlapping web sources. *PVLDB* (2013).

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[15] Michael J. Cafarella, Christopher Re, Dan Suciu, Oren Etzioni, and Michele Banko. 2007. Structured Querying of Web Text. In *Conference on Innovative Data Systems Research (CIDR)*.

[16] Michael J Cafarella, Dan Suciu, and Oren Etzioni. 2007. Navigating Extracted Data with Schema Discovery.. In *WebDB*. 1–6.

[17] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Sam Madden, and Nan Tang. 2023. Symphony: Towards Natural Language Query Answering over Multi-modal Data Lakes. *CIDR* (2023).

[18] Eric Chu, Akanksha Baid, Ting Chen, AnHai Doan, and Jeffrey Naughton. 2007. A Relational Approach to Incrementally Extracting and Querying Structure in Unstructured Data. In *VLDB*.

[19] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations (ICLR)*.

[20] W. Cohen. 2004. Information extraction and integration: An overview. *IJCAI* (2004).

[21] Lei Cui, Furu Wei, and Ming Zhou. 2022. Neural Open Information Extraction. (2022).

[22] Xiang Deng, Prashant Shiralkar, Colin Lockard, Binxuan Huang, and Huan Sun. 2022. DOM-LM: Learning Generalizable Representations for HTML Documents. (2022).

[23] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S Weld. 2008. Open information extraction from the web. *Commun. ACM* 51, 12 (2008), 68–74.

[24] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. 2004. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th international conference on World Wide Web*. 100–110.

[25] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. 2004. Unsupervised named-entity extraction from the Web: An experimental study. In *AAAI*.

[26] J. H. Faghmous and V Kumar. 2014. A big data guide to understanding climate change: The case for theory-guided data science. In *Big data*.

[27] Daniel Fu, Mayee Chen, Frederic Sala, Sarah Hooper, Kayvon Fatahalian, and Christopher Re. 2020. Fast and Three-rious: Speeding Up Weak Supervision with Triplet Methods. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 119. PMLR, 3280–3291.

[28] Leo Gao. 2021. On the Sizes of OpenAI API Models. https://blog.eleuther.ai/gpt3-model-sizes/

[29] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2021. The Pile: An 800GB Dataset of Diverse Text for Language Modeling.

[30] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, Sridhar Seshadri, and Kyuseok Shim. 2000. XTRACT: A system for extracting document type descriptors from XML documents. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 165–176.

[31] Eugene Agichtein Luis Gravano. 2000. Snowball: Extracting Relations from Large Plain-Text Collections. In *DL '00: Proceedings of the fifth ACM conference on Digital libraries*.

[32] Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. 2011. From one tree to a forest: a unified solution for structured web data extraction. *SIGIR* (2011).

[33] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-Enhanced BERT With Disentangled Attention. In *International Conference on Learning Representations*.

[34] Nathan Heller. 2017. What the Enron E-mails Say About Us. https://www.newyorker.com/magazine/2017/07/24/what-the-enron-e-mails-say-about-us

[35] Nick Huss. 2023. How Many Websites Are There in the World?

[36] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2022. Demonstrate-Search-Predict: Composing Retrieval and Language Models for Knowledge-Intensive NLP. *arXiv preprint arXiv:2212.14024* (2022).

[37] B. Klimt and Y. Yang. 2004. Introducing the enron corpus. In *Proceedings of the 1st Conference on Email and Anti-Spam (CEAS)*.

[38] Jan Kocoń, Igor Cichecki, Oliwier Kaszyca, Mateusz Kochanek, Dominika Szydło, Joanna Baran, Julita Bielaniewicz, Marcin Gruza, Arkadiusz Janz, Kamil Kanclerz, et al. 2023. ChatGPT: Jack of all trades, master of none. *arXiv preprint arXiv:2302.10724* (2023).

[39] Keshav Kolluru, Vaibhav Adlakha, Samarth Aggarwal, Mausam, and Soumen Chakrabarti. 2020. OpenIE6: Iterative Grid Labeling and Coordination Analysis for Open Information Extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[40] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation. *ArXiv* abs/2211.11501 (2022).

[41] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, and more. 2022. Holistic Evaluation of Language Models. *ArXiv* abs/2211.09110 (2022).

[42] Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. 2021. Jurassic-1: Technical details and evaluation. (2021).

[43] Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. 2019. OpenCeres: When Open Information Extraction Meets the Semi-Structured Web. *Proceedings of NAACL-HLT* (2019).

[44] Colin Lockard, Prashant Shiralkar, Xin Luna Dong, and Hannaneh Hajishirzi. 2020. ZeroShotCeres: Zero-Shot Relation Extraction from Semi-Structured Webpages. *ACL* (2020).

[45] Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. 2012. Open Language Learning for Information Extraction. *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

[46] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proceedings of the VLDB Endowment International Conference on Very Large Databases* (2022).

[47] Fatemeh Nargesian, Erkang Zhu, Reneé J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proceedings of the VLDB Endowment* (2019).

[48] Christina Niklaus, Matthias Cetto, André Freitas, and Siegfried Handschuh. 2018. A Survey on Open Information Extraction. In *Proceedings of the 27th International Conference on Computational Linguistics*.

[49] OpenAI. March 2023. OpenAI API. https://openai.com/api/

[50] Laurel Orr. 2022. Manifest. https://github.com/HazyResearch/manifest.

[51] F. Chen A. Doan P. DeRose, W. Shen and R. Ramakrishnan. 2007. Building structured web community portals: A top-down, compositional, and incremental approach. *VLDB* (2007).

[52] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv:1606.05250* (2016).

[53] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré . 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proceedings of the VLDB Endowment (VLDB)* (2017).

[54] C. Romero and S. Ventura. 2013. Data mining in education. In *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*.

[55] Shreya Shankar, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. 2022. Operationalizing Machine Learning: An Interview Study. *arXiv:2209.09125* (2022).

[56] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E Gonzalez, et al. 2023. High-throughput Generative Inference of Large Language Models with a Single GPU. *arXiv preprint arXiv:2303.06865* (2023).

[57] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. 2015. Incremental knowledge base construction using deepdive. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases (VLDB)*.

[58] Ryan Smith, Jason A. Fries, Braden Hancock, and Stephen H. Bach. 2022. Language Models in the Loop: Incorporating Prompting into Weak Supervision. *arXiv:2205.02318v1* (2022).

[59] Immanuel Trummer. 2022. CodexDB: synthesizing code for query processing from natural language instructions using GPT-3 codex. *Proceedings of the VLDB Endowment* 11 (2022). https://doi.org/10.14778/3551793.3551841

[60] S. Raghavan S. Vaithyanathan T.S. Jayram, R. Krishnamurthy and H. Zhu. 2006. Avatar information extraction system. *IEEE Data Eng. Bull* (2006).

[61] Paroma Varma and Christopher Ré. 2018. Snuba: Automating Weak Supervision to Label Training Data.

[62] Paroma Varma, Frederic Sala, Ann He, Alexander Ratner, and Christopher Re. 2019. Learning Dependency Structures for Weak Supervision Models. *Proceedings of the 36th International Conference on Machine Learning (ICML)*.

[63] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le Le, Ed H. Cho, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv:2203.11171v2*.

[64] Eric Wu, Kevin Wu, Roxana Daneshjou, David Ouyang, Daniel Ho, and James Zou. 2021. How medical AI devices are evaluated: limitations and recommendations from an analysis of FDA approvals. *Nature Medicine* 27 (04 2021), 1–3.

[65] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *CHI Conference on Human Factors in Computing Systems*. 1–22.

[66] Wael M.S. Yafooz, Siti Z.Z. Abidin, Nasiroh Omar, and Zanariah Idrus. 2013. Managing unstructured data in relational databases. In *2013 IEEE Conference on Systems, Process Control (ICSPC)*.

[67] Shaowen Zhou, Bowen Yu, Aixin Sun, Cheng Long, Jingyang Li, Haiyang Yu, Jian Sun, and Yongbin Li. 2022. A Survey on Neural Open Information Extraction: Current Status and Future Directions. *IJCAI22* (2022).

[68] Diana M. Zuckerman, Paul Brown, and Steven E. Nissen. 2011. Medical Device Recalls and the FDA Approval Process. *Archives of Internal Medicine* 171, 11 (06 2011), 1006–1011.