LaTeXSolver: a Hierarchical Semantic Parsing of LaTeX Document into Code for an Assistive Optimization Modeling Application

Rindra Ramamonjison[1], **Timothy T. Yu**[1], **Linzi Xing**[1,2], **Mahdi Mostajabdaveh**[1], **Xiaorui Li**[1], **Xiaojin Fu**[3], **Xiongwei Han**[3], **Yuanzhe Chen**[3], **Ren Li**[4], **Kun Mao**[5], **Yong Zhang**[1]
[1]Huawei Technologies Canada, [2]University of British Columbia
[3]Huawei Noah's Ark Lab, [4]Huawei's UCD Center, [5]Huawei Cloud Computing Technologies
{rindranirina.ramamonjison, timothyt.yu, linzi.xing, mahdi.mostajabdaveh1, xiaorui.li, fuxiaojin, hanxiongwei, chenyuanzhe, liren09, maokun, yong.zhang3}@huawei.com

Abstract

We demonstrate an interactive system to help operations research (OR) practitioners convert the mathematical formulation of optimization problems from LaTeX document format into the solver modeling language. In practice, a manual translation is cumbersome and time-consuming. Moreover, it requires an in-depth understanding of the problem description and a technical expertise to produce the modeling code. Thus, our proposed system LaTeXSolver helps partially automate this conversion and help the users build optimization models more efficiently. In this paper, we describe its interface and the components of the hierarchical parsing system. A video demo walk-through is available online.[1]

1 Introduction

Operations Research (OR) is a useful yet complex framework for optimal decision-making. For instance, OR has been used to increase bike-share ridership and efficiency (Beairsto et al., 2021), or to optimize wastewater collection and treatment (Tao et al., 2020) in cities. Despite its importance in many fields, the OR process is both time-consuming and knowledge-intensive. First, the problem specifications provided by domain experts must be formulated in mathematical form (Carter and Price, 2017). Then, the formulation needs to be converted into a model code that optimization solvers can interpret. Next, data parameters must be collected and used to instantiate the optimization model. Finally, a proper solver needs to be selected to solve the given problem and find an optimal solution. Traditionally, the domain expert hires an OR expert to handle these strenuous tasks and build the right model for the problem of interest.

There are two shortcomings in the above process, which increases the project cost and duration.

1. First, the formulation needs to be written twice. OR experts typically write the problem formulation as a LaTeX document containing both natural language (NL) descriptions and math formulas. Then, they translate it into code using a solver-specific modeling language. This manual work creates a bottleneck and a mental overhead for the OR experts.

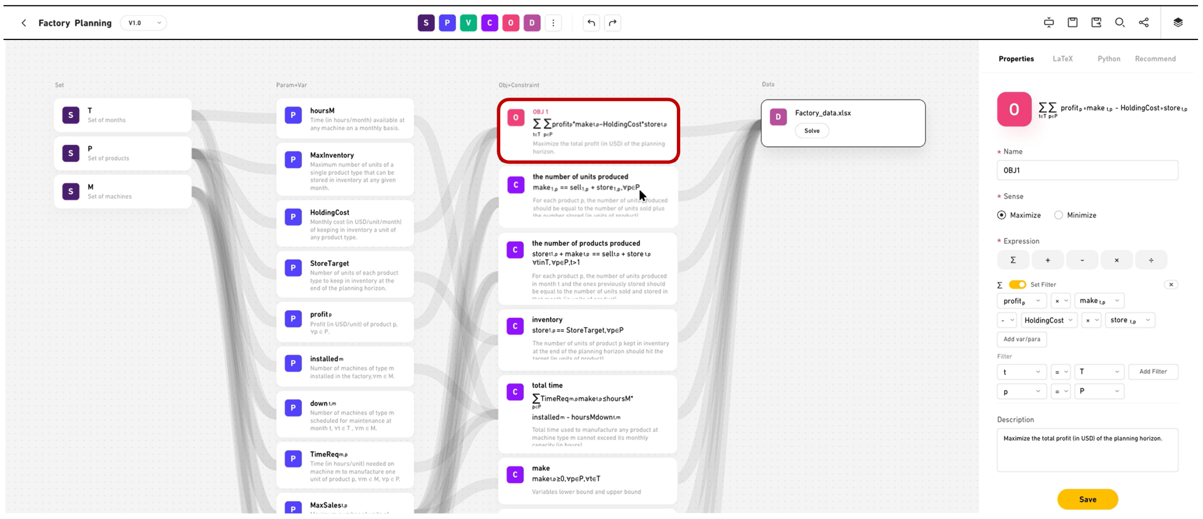
2. Second, the optimization models are saved in two different formats namely the LaTeX document format and the modeling code format. This makes it difficult to manage, edit or share the optimization models. Even if software versioning systems can be used to track the document or code changes, they are quite limited and cumbersome for this purpose.

To address these shortcomings, we introduce LaTeXSolver, an interactive system that takes as input the problem description of a model in LaTeX and partially automates its translation into modeling code. To the best of our knowledge, this is the first modeling tool that accepts an unstructured and multi-modal LaTeX document as input format.

Moreover, we introduce a unified *symbolic model* representation, which decouples the translation procedure into two stages. First, our system combines information extraction methods (i.e., text segmentation, entity recognition, relation extraction) with grammar-based parsing to extract the symbolic model. In the second stage, our system uses the actual data parameters to instantiate the symbolic model and generate the modeling code.

Finally, our intuitive user interface displays the symbolic model as a graph of the model elements as shown in Figure 1. Each node shows the formula and metadata of an element and allows the user to review and edit it before it is turned into modeling code. This added flexibility puts the user in control.

[1]https://bit.ly/3kuOm3x



T
Set of months

P
Set of products

M
Set of machines

hoursM
Time (in hours/month) available at any machine on a monthly basis.

MaxInventory
Maximum number of units of a single product type that can be stored in inventory at any given month.

HoldingCost
Monthly cost (in USD/unit/month) of keeping in inventory a unit of any product type.

StoreTarget
Number of units of each product type to keep in inventory at the end of the planning horizon.

$profit_p$
Profit (in USD/unit) of product p, $\forall p \in P$.

$installed_m$
Number of machines of type m installed in the factory, $\forall m \in M$.

$down_{t,m}$
Number of machines of type m scheduled for maintenance at month t, $\forall t \in T$, $\forall m \in M$.

$TimeReq_{m,p}$
Time (in hours/unit) needed on machine m to manufacture one unit of product p, $\forall m \in M$, $\forall p \in P$.

$MaxSales_{t,p}$

OBJ 1
$\sum_{t \in T} \sum_{p \in P} profit_p * make_{t,p} - HoldingCost * store_{t,p}$
Maximize the total profit (in USD) of the planning horizon.

the number of units produced
$make_{1,p} == sell_{1,p} + store_{1,p}, \forall p \in P$
For each product p, the number of units produced should be equal to the number of units sold plus the number stored (in units of product)

the number of products produced
$store_{t-1,p} + make_{t,p} == sell_{t,p} + store_{t,p}$ $\forall t in T, \forall p \in P, t>1$
For each product p, the number of units produced in month t and the ones previously stored should be equal to the number of units sold and stored in that month (in units of product)

inventory
$store_{1,p} == StoreTarget, \forall p \in P$
The number of units of product p kept in inventory at the end of the planning horizon should hit the target (in units of product)

total time
$\sum_{p \in P} TimeReq_{m,p} make_{t,p} \leq hoursM * installed_m - hoursM down_{t,m}$
Total time used to manufacture any product at machine type m cannot exceed its monthly capacity (in hours)

make
$make_{t,p} \geq 0, \forall p \in P, \forall t \in T$
Variables lower bound and upper bound

Factory_data.xlsx

$\sum_{t \in T} \sum_{p \in P} profit_p * make_{t,p} - HoldingCost * store_{t,p}$

* Name
OBJ1

* Sense
[✓ Maximize] Minimize

* Expression
$profit_p$ × $make_{t,p}$
- HoldingCost × $store_{t,p}$

t = T
p = P

Description
Maximize the total profit (in USD) of the planning horizon.

Figure 1: Screenshot of the LATEXSOLVER interactive application. The system displays the parsed symbolic model graphically where the user can interact with the cards representing different modeling components. The right panel empowers the user to revise each component efficiently and with ease.

2 Related Work

Modeling aid tools for operations research. Past efforts to improve the experience of model builders has centered around creating better modeling languages. While they can enable rapid prototyping (Koch, 2004), the modeler still needs to learn these languages and manually convert the mathematical formulation into an equivalent code.

To alleviate the technical barriers and make solvers more accessible, alternative input formats have also been proposed such as Excel spreadsheets (Lin and Schrage, 2009), web forms (Triantafyllidis and Papageorgiou, 2018; Esche et al., 2017), and natural language (IBM, 2022). In comparison, our system is the first to accept a multi-modal markup document that contains natural language descriptions as well as mathematical equations.

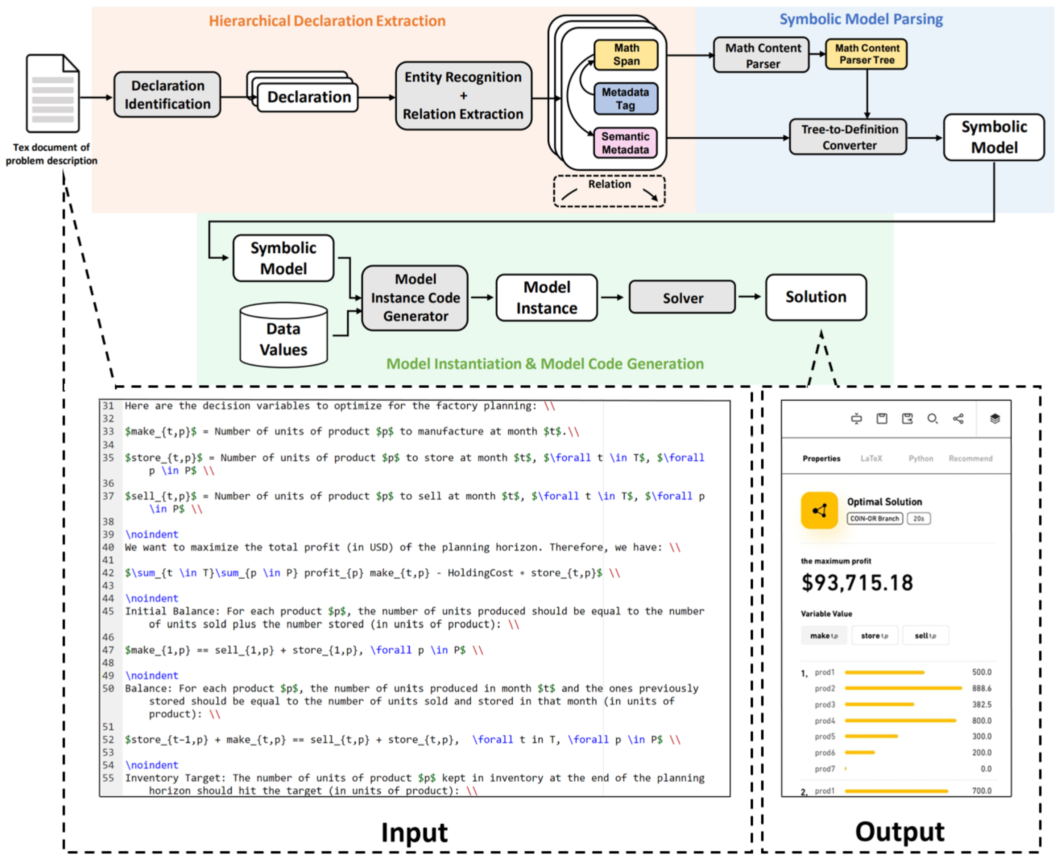
Information extraction from scientific documents. Recently, information extraction from scientific documents has received increasing research interests. In fact, scientific documents are different from natural language texts due to the syntactic difference and to the presence of symbols and formulas (Lai et al., 2022). (Beltagy et al., 2019) proposed SciBERT is an example of a pre-trained language model on corpora for different scientific domains. SciBERT was used in Lee and Na (2022) and Popovic et al. (2022) as the backbone for entity recognition and relation extraction tasks. Moreover, Lee and Na (2022) reframed entity recognition and relation extraction tasks as machine reading comprehension to leverage the mathematical knowledge learned by SciBERT. For our system, we also adopt SciBERT and fine-tune it using our labeled dataset for the information extraction tasks. In addition, we use a neural text segmentation and labeling model as the first step to divide the input markup document into declaration segments.

Program synthesis and math word problems. Language models pretrained on source code have shown some promising performance in generating or summarizing computer programs (et al., 2021). These models have also been used to generate programs that solve math problems (Drori et al., 2022). Nonetheless, recent studies have shown that even the largest models such as Codex can still hallucinate and produce erroneous code (Xu et al., 2022). Direct translation from an unstructured markup language to code is an under-explored task and current techniques does not deliver consistently accurate results. Instead, we divide it into smaller tasks and simplify the parsing by leveraging grammar-based parsers.

3 System Overview

Figure 1 showcases the graphical user interface of our LATEXSOLVER web application built using the Vue frontend framework (Vue.js, 2014).

This interface enables users to upload an optimization problem description in LaTeX, composed of both natural language and mathematical formulas. After going through the parsing process, the input LaTeX document will be transformed into a symbolic model that serves as a united repre-


Hierarchical Declaration Extraction
Tex document of problem description
Declaration Identification
Declaration
Entity Recognition + Relation Extraction
Math Span
Metadata Tag
Semantic Metadata
Relation
Symbolic Model Parsing
Math Content Parser
Math Content Parser Tree
Tree-to-Definition Converter
Symbolic Model
Symbolic Model
Data Values
Model Instance Code Generator
Model Instance
Solver
Solution
Model Instantiation & Model Code Generation
Input
Output
Properties
LaTeX
Python
Recommend

```
Here are the decision variables to optimize for the factory planning: \\

$make_{t,p}$ = Number of units of product $p$ to manufacture at month $t$.\\

$store_{t,p}$ = Number of units of product $p$ to store at month $t$, $\forall t \in T$, $\forall
    p \in P$ \\

$sell_{t,p}$ = Number of units of product $p$ to sell at month $t$, $\forall t \in T$, $\forall p
    \in P$ \\

\noindent
We want to maximize the total profit (in USD) of the planning horizon. Therefore, we have: \\

$\sum_{t \in T}\sum_{p \in P} profit_{p} make_{t,p} - HoldingCost * store_{t,p}$ \\

\noindent
Initial Balance: For each product $p$, the number of units produced should be equal to the number
    of units sold plus the number stored (in units of product): \\

$make_{1,p} == sell_{1,p} + store_{1,p}, \forall p \in P$ \\

\noindent
Balance: For each product $p$, the number of units produced in month $t$ and the ones previously
    stored should be equal to the number of units sold and stored in that month (in units of
    product): \\

$store_{t-1,p} + make_{t,p} == sell_{t,p} + store_{t,p},  \forall t in T, \forall p \in P$ \\

\noindent
Inventory Target: The number of units of product $p$ kept in inventory at the end of the planning
    horizon should hit the target (in units of product): \\
```

Optimal Solution
COIN-OR Branch 20s

the maximum profit
$93,715.18

Variable Value
make t,p store t,p sell t,p

```
1. prod1  500.0
   prod2  888.6
   prod3  382.5
   prod4  800.0
   prod5  300.0
   prod6  200.0
   prod7    0.0
2. prod1  700.0
```

Figure 2: System diagram for LATEXSOLVER.

sentation of the input optimization problem. As an example shown in the main panel of Figure 1, a symbolic model comprises model elements extracted from the document and each element is categorized as one of {*Set*, *Parameter*, *Objective*, *Variable*, *Constraint*, *Objective*} (listed on top of the interface). All parameters and variables in the symbolic model are represented by symbols rather than actual values. Our user interface is designed to display the extracted symbolic model as a graph, in which each node contains the formula and metadata of a model element and dynamic links are used to highlight the relationships between symbolic elements. To enhance user engagement and system flexibility, we allow users to review and edit the displayed model elements. Users can click on each model element card to reveal its detailed properties in the right-side panel of the interface, where they can edit the properties as desired. Furthermore, users have the ability to add or delete elements as needed. Once the user is satisfied with the updated symbolic model, our system will guide the user to upload data values to instantiate the symbolic model and generate the corresponding model code instance for the optimization solver to compute for solutions.

The backend workflow of LATEXSOLVER is illustrated in Figure 2. The system's pipeline consists of three major stages, namely *hierarchical declaration extraction*, *symbolic model parsing* and *model code generation*. Given a problem description in LaTex as input, the system first segments it into a set of declarations, each of which describes a specific model element and includes declaration entities linked by corresponding relations. This stage employs three neural models that were initially introduced for three NLP tasks: text segmentation and labeling, entity recognition, and relation extraction respectively. In the subsequent stage, the extracted declarations are transformed into a symbolic optimization model using a context-free grammar-based parser, which leverages the metadata obtained in the previous stage as supplementary information. The resulting symbolic model is then passed into a solver API-specified model code instance generator, together with user-specified data values, to generate the model code that is ready to be processed by optimization solvers.

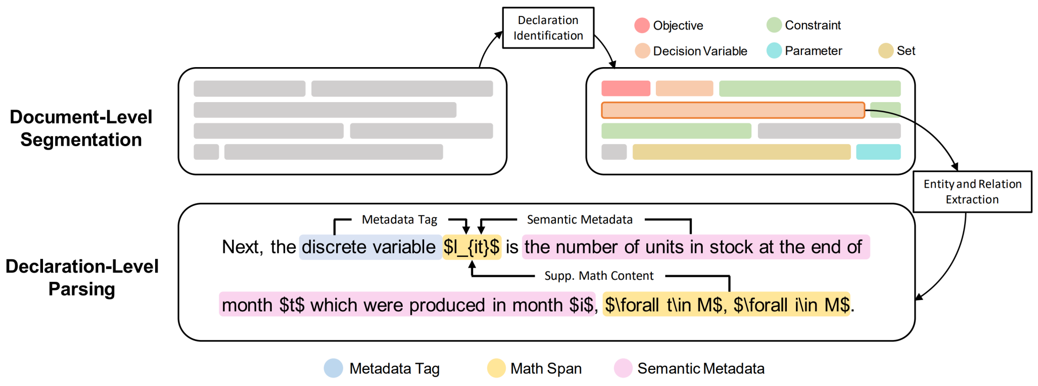

Document-Level Segmentation
Declaration Identification
Objective
Constraint
Decision Variable
Parameter
Set
Entity and Relation Extraction
Declaration-Level Parsing
Metadata Tag
Semantic Metadata
Next, the discrete variable I_{it} is the number of units in stock at the end of month t which were produced in month i, $\forall t\in M$, $\forall i\in M$.
Supp. Math Content
Metadata Tag
Math Span
Semantic Metadata

Figure 3: The illustration of Hierarchical Declaration Extraction.

4 Hierarchical Declaration Extraction

A declaration in this paper is, by definition, a multi-modal segment consisting of one or more sentences written in a mixture of text and math content, as exemplified in Figure 3. Each declaration typically describes one particular model element, such as an *objective*, *constraint*, *decision variable*, *parameter* or *set*. As the first stage of LATEXSOLVER, the system takes a LaTeX document as input and extracts declarations from it in a hierarchical manner. Specifically, the system first performs document-level declaration identification to extract and label all the text segments in the document, each associated with one declaration (§4.1). Next, our system performs entity recognition and entity linking within each extracted declaration segment (§4.2).

4.1 Document-Level Declaration Identification

In order to identify all declarations contained in the document, as well as assign each identified declaration a label indicating the model element it contributes to, we propose to re-purpose a neural model originally proposed in Barrow et al. (2020) for text segmentation and labeling. In this work, we employ this model for declaration segmentation and labeling.

Figure 4 illustrates the high-level architecture of the neural model we used for declaration segmentation and labeling. In practice, the system accepts the input document formalized as a sequence of consecutive sentences[2]. However, the structure of a LaTeX document is usually not flat but contains nested content blocks organized as bullet lists, or covered in captions of figures and tables. Therefore,

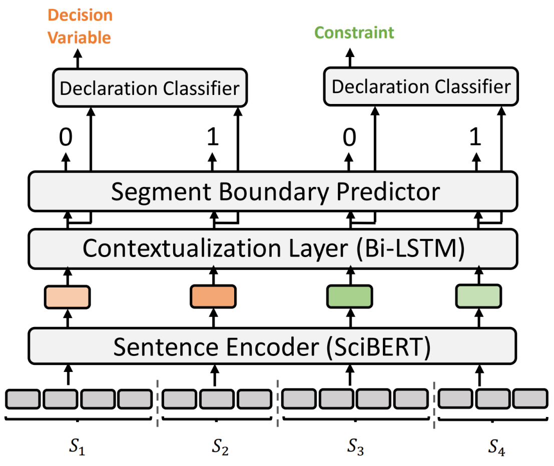


Figure 4: The model architecture for declaration segmentation and labeling.

we initially carry out a rule-based pre-processing step to detect and eliminate these structures by converting the content in the nested form to the flat form before passing them into the model. As the first layer of the neural model, a sentence encoder is in place to yield low-level features (embeddings) for the input sentences. Taking into consideration that (1) documents in LaTex are more likely in the scientific domain, and (2) sentences within these documents are likely to have both text and mathematical content, we choose SciBERT (Beltagy et al., 2019) as our sentence encoder, which is equipped with a rule-based symbol tokenizer proposed in Lee and Na (2022) to alleviate the limitation of SciBERT's tokenizer in detecting the boundaries of mathematical symbols.

Given the sentence embeddings obtained from the SciBERT sentence encoder, a document-level contextualization layer (Bi-LSTM) returns an ordered set of sentence hidden states for two objectives: (1) declaration segment boundary prediction

[2]We applied `nltk.sent_tokenize` for sentence segmentation.

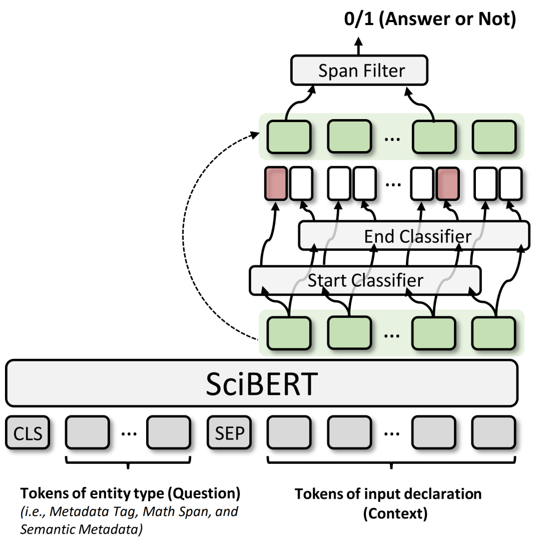

0/1 (Answer or Not)
Span Filter
End Classifier
Start Classifier
SciBERT
CLS
SEP
Tokens of entity type (Question)
(i.e., Metadata Tag, Math Span, and Semantic Metadata)
Tokens of input declaration (Context)

Figure 5: The model architecture for entity recognition.

and (2) declaration labeling. Concretely, a multi-layer perceptron (MLP) followed by softmax activation serves as a binary boundary predictor, where label "1" means the corresponding sentence is the end of a segment, and "0" otherwise. After segmenting the document, we further apply mean pooling over the sentence hidden states within each predicted segment to predict the declaration label for the segment. Another MLP+softmax layer is utilized to classify each declaration segment into one of the following classes: {*Objective*, *Constraint*, *Decision Variable*, *Parameter*, *Set*, *Others*}. This framework is optimized by minimizing the cross-entropy losses for both objectives of declaration segmentation and labeling.

4.2 Entity Recognition and Relation Extraction

Once the underlying declarations have been extracted from the input LaTeX document, the next step entails extracting entities within each declaration, as well as the relations linking up these entities. To achieve this, we leverage the entity recognition and relation extraction models proposed in Lee and Na (2022), initially devised for machine reading comprehension on documents containing mathematical symbols.

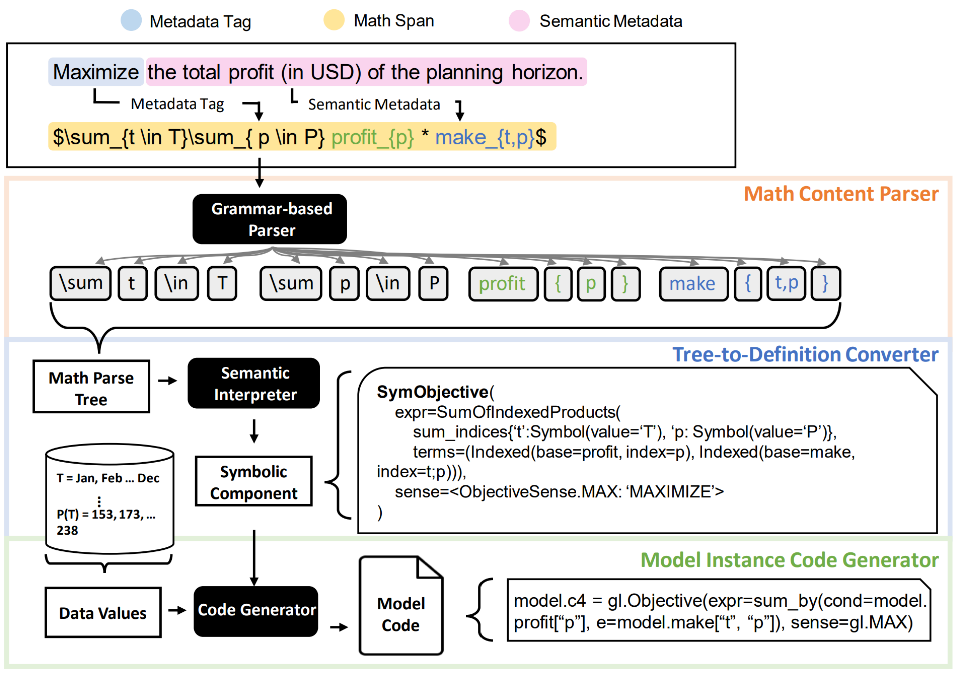
The entity recognition model formulates the process of extracting entities as a machine reading comprehension task by providing an entity type as a question and utilizing SciBERT as the backbone to extract mentions of this entity type in a declaration segment as answers. As shown in Figure 5, a given input declaration is deemed as context and concatenated with each of the three pre-defined entities types (i.e., *Metadata Tag*, *Math Span*, *Semantic Metadata*) in our labeled corpus. Subsequently, the Question+Context concatenation is passed into SciBERT, and the output hidden representations of the tokens covered by the declaration (context) are used to estimate the probability of each token being the start or end (i.e., i_{start} or i_{end}) of a mention (answer) of the concatenated entity type (question). Next, for any span of (i_{start}, i_{end}), another binary classifier is applied to predict whether the span is the answer to extract.

Similar to the entity recognition model described above, we also leverage SciBERT to perform relation extraction between pairs of entities within each declaration segment by simply encoding entities and doing relation prediction based on the concatenation of entity representations (mean of token embeddings covered in the text span of entities) obtained from SciBERT (Lee and Na, 2022). We pre-define four types of relations, namely *Metadata Tag*, *Semantic Metadata* and *Supplementary Math Content* and *NIL*, where *NIL* indicates that no relation exists between the two entities.

For both entity recognition and relation extraction, we set some heuristic rules to refine models' predictions by cleaning up the entities which are unlikely to co-occur within a declaration, as well as relations with the type unlikely to appear to link two certain entities.

5 Symbolic Model Parsing and Model Code Generation

This section describes the process of converting the detected entities and their relations into a symbolic model which is eventually generated into modeling code, as shown in Figure 6. The symbolic model and concrete data values, such as sets and parameter values, are passed to the model instance code generator which converts them into code based on the target modeling language or solver API. The generated code consists of the model components and their corresponding data values. By decoupling the symbolic model and data values, we allow the users to evaluate their model and problem more efficiently. They can modify the model to evaluate variants of the model or easily change data values to examine different scenarios of the problem.



```
SymObjective(
    expr=SumOfIndexedProducts(
        sum_indices{'t':Symbol(value='T'), 'p: Symbol(value='P')},
        terms=(Indexed(base=profit, index=p), Indexed(base=make,
index=t;p))),
    sense=<ObjectiveSense.MAX: 'MAXIMIZE'>
)
```

```
model.c4 = gl.Objective(expr=sum_by(cond=model.
profit["p"], e=model.make["t", "p"]), sense=gl.MAX)
```

Figure 6: The illustration of symbolic model parsing and model code generation process.

5.1 Symbolic Model Parsing

As part of LATEXSOLVER, we have implemented a symbolic model, which is an intermediate representation of the problem that lies between the natural language & math problem description and model code. This symbolic representation of the model maps the math formulas into a nested structure of symbolic elements before data instantiation. The symbolic model parsing is performed by two modules: (1) a math content parser, and (2) a tree-to-definition converter. An example of both modules is shown in Figure 6.

First, the math content parser converts the math content spans of each model component declaration detected by the declaration extraction stage into a parse tree. To do so, the LaTeX math formulas are parsed using the grammar-based parser generator ANTLR4 (Parr, 2013). ANTLR4 detects the required attributes of modeling components. Specifically, indexed parameters and variables are parsed along with the set that the indices are bound to. The parser also parses constraint math formulas into the constraint expression, comparison operator (`<`, `<=`, `>`, `>=`, `==`), and the right-hand-side equation. Finally, the objective math formula would contain the objective sense and the objective function. These math formulas are processed into a parse tree where the leaves contain the important fields required to populate the symbolic model. Figure 6 shows a graphical representation of the output parse tree of the grammar-based parser.

Next, the tree-to-definition converter loops through the parse tree, it processes the detected model components, and creates a new model component in the symbolic model class. As these elements are detected, they populate the attributes of the corresponding components in the symbolic model class in a nested manner.

5.2 Model Code Generation

The model instance code generator leverages Python's metaprogramming mechanism to generate the modeling code during runtime. The pipeline begins with a template code string that imports the required Python libraries for the solver API. The model code generator adds to the model based on the specifications of the API. In our implementation, we used the Huawei OptVerse solver and its Grassland API (Li et al., 2021). The model instance code generator reads in the data values from spreadsheet files using predefined sheet references and adds the data initialization commands in the generated code. Then, it will loop through the elements of the symbolic model to translate it into the solver API code to declare the sets, variables, objective and constraints.

6 Limitations and Future Works

The LATEXSOLVER system is a useful tool that partially automates the conversion of a problem description into an optimization model, but some limitations highlight potential areas for improvement. First, it was assumed that each declaration contains all complete information that needs to be parsed into a model element. While our proposed system can handle unstructured text at the declaration-level, it is a more challenging problem if information about one model component is scattered across declarations. We believe that in most cases, the input documents would satisfy this assumption. Second, this system requires the LaTeX document input to be accurate and errors may cause issues that cascade through the steps of the system (e.g., an error in segmenting and labeling the declarations may yield errors in the entity recognition and relation extraction tasks). The interactive application addresses this by keeping the human in the loop allowing the user to interact with the detected components and their relations. Finally, with the rapid advancements in LLMs (Li et al., 2023; OpenAI, 2023), domain-specific tools such as LaTeXSolver could be used in conjunction with general-purpose language models.

7 Conclusion

In this paper, we introduce LaTeXSolver, an interactive system to help operations research practitioners efficiently convert the mathematical formulation of optimization problems from the unstructured and multi-modal LaTeX document format into the solver modeling language. The system follows a two-step process of converting the LaTeX document to a symbolic model and then to model code. Users can easily review and edit the symbolic model automatically extracted from the LaTeX document with an intuitive interface to ensure the system's reliability.

Ethics Statement

The LaTeXSolver system presented in this paper aims to partially automate the process of converting problem descriptions in LaTeX to model codes, thereby helping OR experts build optimization models more efficiently. As the system's input and output are transparent to users and users can control the model-building procedure by interacting with the system, the harm to users resulting from the errors produced by the system is limited. However, our system may be used in certain circumstances considered sensitive or critical, such as power grid or flights scheduling. In such cases, the system should be used with caution and the modeling process should be investigated by domain experts. Additionally, given the historic application of operations research in tactical military operations, it is critical to understand the potential negative impact of misapplying this technology to society. Therefore, users of our system must be aware of any ethical concern come with military applications of this technology.

References

Joe Barrow, Rajiv Jain, Vlad Morariu, Varun Manjunatha, Douglas Oard, and Philip Resnik. 2020. A joint model for document segmentation and segment labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 313–322, Online. Association for Computational Linguistics.

Jeneva Beairsto, Yufan Tian, Linyu Zheng, Qunshan Zhao, and Jinhyun Hong. 2021. Identifying locations for new bike-sharing stations in glasgow: an analysis of spatial equity and demand factors. *Annals of GIS*, 0(0):1–16.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Michael W Carter and Camille C Price. 2017. *Operations research: a practical introduction*. Crc Press.

Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, Roman Wang, Nikhil Singh, Taylor L. Patti, Jayson Lynch, Avi Shporer, Nakul Verma, Eugene Wu, and Gilbert Strang. 2022. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences (PNAS)*, 119(32).

Erik Esche, Christian Hoffmann, Markus Illner, David Müller, Sandra Fillinger, Gregor Tolksdorf, Henning Bonart, Günter Wozny, and Jens-Uwe Repke. 2017. Mosaic–enabling large-scale equation-based flow sheet optimization. *Chemie Ingenieur Technik*, 89(5):620–635.

Chen et al. 2021. Evaluating large language models trained on code.

IBM. 2022. Modeling assistant models (decision optimization).

Thorsten Koch. 2004. *Rapid Mathematical Prototyping*. Ph.D. thesis, Technische Universität Berlin.

Viet Lai, Amir Pouran Ben Veyseh, Franck Dernoncourt, and Thien Nguyen. 2022. SemEval 2022 task 12: Symlink - linking mathematical symbols to their descriptions. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 1671–1678, Seattle, United States. Association for Computational Linguistics.

Sung-Min Lee and Seung-Hoon Na. 2022. JBNU-CCLab at SemEval-2022 task 12: Machine reading comprehension and span pair classification for linking mathematical symbols to their descriptions. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 1679–1686, Seattle, United States. Association for Computational Linguistics.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. Starcoder: may the source be with you!

Xihan Li, Xiongwei Han, Zhishuo Zhou, Mingxuan Yuan, Jia Zeng, and Jun Wang. 2021. Grassland: A rapid algebraic modeling system for million-variable optimization.

Youdong Lin and Linus Schrage. 2009. The global solver in the lindo api. *Optimization Methods & Software*, 24(4-5):657–668.

OpenAI. 2023. Gpt-4 technical report.

Terence Parr. 2013. The definitive antlr 4 reference. *The Definitive ANTLR 4 Reference*, pages 1–326.

Nicholas Popovic, Walter Laurito, and Michael Färber. 2022. AIFB-WebScience at SemEval-2022 task 12: Relation extraction first - using relation extraction to identify entities. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 1687–1694, Seattle, United States. Association for Computational Linguistics.

Diana Qing Tao, Martin Pleau, et al. 2020. Analytics and Optimization Reduce Sewage Overflows to Protect Community Waterways in Kentucky. *Interfaces*, 50(1):7–20.

Charalampos P Triantafyllidis and Lazaros G Papageorgiou. 2018. An integrated platform for intuitive mathematical programming modeling using latex. *PeerJ Computer Science*, 4:e161.

Vue.js. 2014. Vue.js – the progressive javascript framework. `https://vuejs.org/`. Accessed: 2023-05-20.

Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, MAPS 2022, page 1–10, New York, NY, USA. Association for Computing Machinery.