WIQOR: A DATASET FOR WHAT-IF ANALYSIS OF OPERATIONS RESEARCH PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

We formalize the *mathematical program modification* (MPM) task, in which the goal is to revise a mathematical program according to an inquiry expressed in natural language. These inquiries, which we refer to as *what-if questions*, express a desire to understand how the optimal solution to an optimization problem changes with the addition, deletion or revision of constraints. In detail, each MPM instance is a triple consisting of: 1) a natural language *specification* that summarizes an optimization problem, 2) the *canonical* formulation of the problem, and 3) a natural language what-if question. The goal is to predict the updated canonical formulation with respect to the question. To support the study of this task, we construct WIQOR, a dataset of 1,946 MPM instances, derived from NL4OPT (Ramamonjison et al., 2023), but with the number of decision variables extended to more than 30 for some problems. In experiments, we observe that Llama 3.1 70B instruct under the in-context learning paradigm achieves 69% accuracy on the easiest test instances, but only 36% accuracy on the most complicated problems. We release WIQOR in the hopes of spurring additional study of MPM and ultimately enabling non-technical users to conduct what-if analyses without the help of technical experts.

1 INTRODUCTION

Mathematical programming is the centerpiece of decision making in a many industries. For example, firms routinely employ mathematical optimization to set product prices (Ferreira et al., 2016), find optimal transportation routes, or determine production schedules . Tools for mathematical optimization are commonly utilized in academic peer review in order to match papers to reviewers (Taylor, 2008; Stelmakh et al., 2019).

Due in part to the complexity of real-world optimization problems, constructing formal problem representations that can be solved by industry-grade optimizers is typically a cumbersome process requiring collaboration between a two parties: a domain expert and a technical expert. In these cases, the domain expert, e.g., a vendor who seeks to optimize their product prices, does not possess the requisite technical expertise to translate their specific problem into a mathematically rigorous objective function, decision variables, and the associated set of constraints. Instead, the domain expert provides a detailed account of their problem to a technical expert, who uses these details to construct a problem representation that can be sent as input to an off-the-shelf solver (e.g., Gurobi), which returns the solution (Gurobi Optimization, LLC, 2024).

In addition to posing a challenge during program construction, the necessary communication between the domain and technical experts slows analysis of the program and its optimized solution. For example, consider that upon inspection of set of optimized prices, the domain expert seeks to understand how a new constraint on the maximum price for a given product would affect the optimal prices. In such cases, the domain expert must transmit the new constraint to the technical expert, who then modifies the initial program and solves it anew, transmitting the optimal solution back to the domain expert. Needless to say, reliance on the technical expert significantly slows analysis and may lead the domain expert to skip the analysis entirely.

In order to support the development of methods that enable non-technical users to engage in such analysis, we formalize and study a task we term *mathematical program modification*, MPM. In this task, the goal is to update an existing mathematical program according to a natural language

inquiry. In detail, the input in MPM is a triple consisting of: i) a natural language *specification*, or summary, of an optimization problem, ii) the problem expressed formally (mathematically) in *canonical form*, and iii) a natural language inquiry about how a desired change in the constraints would modify optimal solution to the problem, which we call a *what-if question*. The goal of the task is to correctly modify the provided canonical form with respect to the what-if question.

To facilitate the study of this task, we present the WIQOR (**W**hat-**I**f **Q**uestions for **O**perations **R**esearch problems) benchmark dataset. This dataset comprises 1,946 instances of the MPM task, and includes what-if questions that correspond to 4 different types of constraint changes, which can be applied to 7 different constraint types. Each MPM instance in WIQOR is seeded by a canonical formulation that is borrowed from the NL4OPT dataset Ramamonjison et al. (2023). The accompanying problem specification, what-if question, and updated canonical formulation are constructed using a combination of heuristics and large language model inference. All MPM instances undergo manual evaluation before being added to the dataset.

To better capture real-world complexity, WIQOR includes 4 test splits with increasing complexity—as measured by the number of decision variables and constraints. In the `base` (easiest) test split, MPM problems have on average 2 decision variables and 2 constraints. However, WIQOR also includes additional test splits which include 100 base problems that have been extended by adding 5, 10, 20 or 30 additional decision variables and corresponding constraints, respectively. Expansion of canonical form to include additional variables and constraints is accomplished using a sequence of carefully designed heuristics and logic.

Empirically, we evaluate the Llama 3.1 instruct family of models on the WIQOR dataset under the in-context learning paradigm. Our results reveal that the smallest models struggle to achieve more than 10% accuracy on the base test instances. As the number of decision variables increase, even the 70 billion parameter model shows a decline in performance, achieving only 36% accuracy on the most difficult split. We hope that WIQOR helps support the development of tools that will empower domain experts conduct what-if analysis without the need to enlist the help of external technical experts.

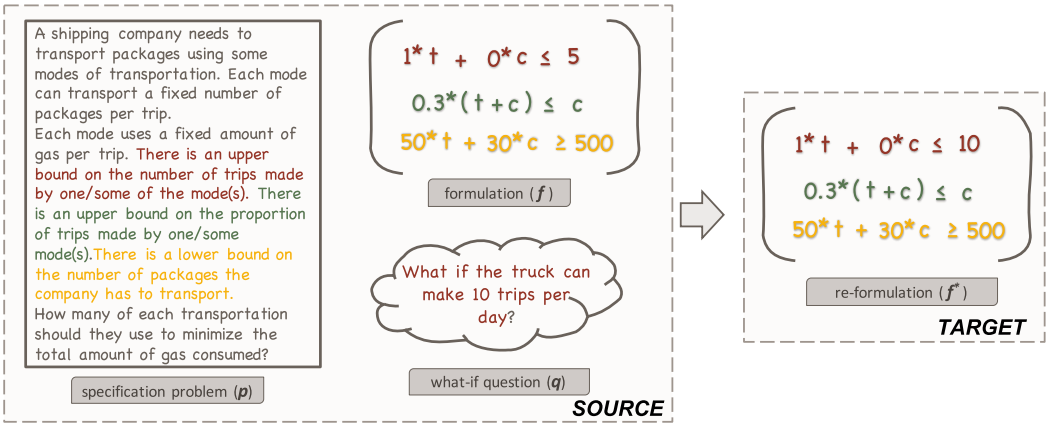

Figure 1: WIQOR dataset overview: a sample from the WIQOR dataset. The left box shows the input triple consisting of problem specification and it's formulation and a what-if question, while the right side demonstrates the reformulation based on that what-if question. The transportation modes (truck and car) are represented as 't' and 'c' in the formulation matrices. **NOTE**: In the actual dataset these matrices contain numerical coefficients for each constraint inequality, all expressed in less-than-or-equal-to form with an order mapping from variable names to column indices.

2 Related Work

LLMs for Operations Research The integration of Large Language Models (LLMs) in Operations Research (OR) has led to significant advancements in the formulation and solving of optimization

problems. WIQOR builds on this trend by addressing a new challenge: revising existing mathematical programs based on natural language requests.

A foundational work for WIQOR is the NL4Opt dataset by Ramamonjison et al. (2023), which focuses on translating natural language descriptions into optimization formulations. WIQOR extends this by introducing problem modification, allowing non-experts to revise constraints through natural language requests.

Similarly, Yang et al. (2024) introduce Benchmarking LLMs for Optimization Modeling. Their study, through the E-OPT benchmark, shows how LLMs can solve optimization problems by understanding human-readable inputs and outputs. They propose the Reverse Socratic Synthesis (ReSocratic) method, which incrementally synthesizes mathematical formulations and back-translates them into problem descriptions. WIQOR's challenge of revising constraints through natural language extends the problem-solving paradigm proposed by Yang et al. (2024).

Furthermore, AhmadiTeshnizi et al. (2024) in OptiMUS propose a scalable system where LLMs develop mathematical models for mixed integer linear programming (MILP) problems. Their framework processes long and complex problem descriptions, using LLMs to write and debug solver code. This research shares a thematic overlap with WIQOR in handling complex optimization tasks; however, WIQOR goes further by focusing on the transformation of existing problem constraints, introducing additional complexity in understanding how to modify mathematical programs based on natural language revisions. Chain-of-Experts, discussed by Xiao et al. (2024), explores how LLMs can be combined in expert chains to tackle complex OR problems. This approach leverages multiple models for specialized tasks, a useful insight for the design of systems like WIQOR, which could potentially benefit from multi-model collaboration to improve accuracy in the modification of complex, variable-dense mathematical programs. In LM4OPT, Ahmed & Choudhury (2024) explore LLMs' ability to formulate optimization problems. WIQOR advances this by focusing on refining those formulations through revision requests, filling an important gap in existing research.

One notable work in this area is by Prasath & Karande (2023), who propose methods for the synthesis of mathematical programs directly from natural language specifications. In their study, they address the challenges of translating natural language into mathematical formulations, focusing on the precise interpretation of linguistic structures to map them onto well-defined mathematical operations. This is highly relevant to the MPM task, as it underscores the inherent complexity of processing ambiguous natural language inputs and converting them into formal optimization structures with accuracy and consistency **?**.

Tsouros et al. (2023), in Holy Grail 2.0: From Natural Language to Constraint Models, push this research further by investigating how large language models (LLMs) and constraint programming can be integrated to develop constraint models based on natural language inputs. Their work emphasizes the need for automated translation of natural language into structured constraint-based frameworks, drawing parallels with our MPM task, where similar transformations are required for revising existing mathematical programs based on user-provided specifications.

Xing et al. (2024) extend this exploration into the realm of linear programming word problems by aligning human-like evaluation metrics to such tasks. Their work focuses on creating benchmarks that test the capability of systems to understand and solve linear programming problems posed in natural language, which aligns with the MPM task's focus on understanding problem constraints and effectively revising them based on natural language modification requests. Their evaluation approach offers insights into how the performance of systems solving mathematical program modifications could be quantitatively assessed.

Lawless et al. (2024), in their paper on enabling interactive decision support using large language models and constraint programming, explore the synergy between LLMs and decision-making frameworks in generating interactive, user-friendly solutions to complex problems. Their focus on interactive decision support resonates with our goal of enabling non-technical users to modify mathematical programs via natural language inputs, particularly when users need to iteratively adjust constraints and variables in optimization models.

Counterfactual reasoning with LLMs

- Counterfactual Story Reasoning and Generation (Qin et al., 2019)

3 THE MATHEMATICAL PROGRAM MODIFICATION TASK (MPM)

Recall that our goal is to facilitate counterfactual analysis (of mathematical programs) by non-experts. To this end, we endeavor to support the creation of tools that enable interactive program revision via natural language. As a first step, we formulate the *mathematical program modification* (MPM) task. In an instance of MPM, the input is a triple consisting of: i) a *specification*, $\mathcal{S}$, i.e., brief summary of an optimization problem, including classes of variables and constraints; ii) a corresponding, fully specified mathematical program, $\mathcal{P}$; and iii) a *what-if question*, $\mathcal{Q}$, that implies a revision of some components (i.e., constraints or variables) of the program. The goal is to revise the initial program according to the requested revision. Formally,

$$\text{MPM} \coloneqq (\mathcal{S} \times \mathcal{P} \times \mathcal{Q}) \to \mathcal{P}^\star \tag{1}$$

where $\mathcal{P}^\star$ is the modified mathematical program. In the following subsections, we detail each component of an instance of MPM, define corresponding notation, and introduce associated terminology.

3.1 OPTIMIZATION PROBLEM SPECIFICATIONS

Real-world optimization problems include hundreds of decision variables and constraints, or more. Therefore, when communicating about such problems, it is common to construct a *specification*, or summary of the problem . Since the specifications are intended to be short even though the underlying problems are complex (measured by the number of decision variables and constraints), the specification often describe the relationships between variables and constraints, rather than identifying each variable and constraint individually. []

Through the problem specifications the focus is on understanding the core structure of the problem rather than getting bogged down in excessive detail about every variable and constraint. In practice, enumerating every single variable and constraint for large-scale problems—such as those in supply chain, logistics, or operational planning—quickly becomes infeasible. Instead, it is more effective to abstract and simplify these problems into specifications, capturing their essential relationships and objectives while avoiding unnecessary complexity. This abstraction not only makes the problem easier to communicate and interpret but also aligns with how real-world decision-makers approach optimization tasks, particularly when the number of variables grows significantly.

Consider Figure 1 for an example of an optimization problem, completely spelled out in natural language and it's specification. The original problem involving a shipping company transporting packages illustrates the complexity of real-world optimization problems, with numerous specific details such as the number of packages each vehicle can transport, fuel consumption, trip limits, and percentage constraints. If there were more modes of transport operated by the shopping company then enumerating each detail about every mode becomes cumbersome.

In contrast, the specification problem abstracts the essential relationships, focusing on core elements like modes of transportation, fuel consumption, trip limits, and transportation requirements, without getting bogged down in the specifics of each mode of transportation. This approach highlights the problem's structure and objectives, making it easier to communicate and solve while maintaining the integrity of the original problem. Also, avoiding mentioning the number or name of the modes of transportation (variables) allows for expanding to any number of variables formulated through the mathematical program.

3.2 MATHEMATICAL PROGRAMS

The problem specification summarizes an underlying (constrained) optimization problem. To solve the optimization, it is typically represented as a *mathematical program*, $\mathcal{P}$, which serves as the input for an off-the-shelf solver. Briefly, each program we study includes an *objective function*, a collection of *decision variables* and a set of *constraints*. The objective is a function of the decision variables; constraints also utilize these variables. The solver's role is to find a setting of the decision variables that minimizes[1] the objective, subject to the constraints. Such solvers employ various algorithms and heuristics to efficiently optimize a wide array of objectives. We refer the interested read to one of the definitive references on the subject **?**.

[1] Without loss of generality.

In our work, we assume that each program is expressed in *canonical form*. For each program in canonical form: i the objective is a maximization of a linear combination of decision variables, and ii the constraint matrix is written as $Ax \leq b$, where A_{ij} is the coefficient corresponding to decision variable j in constraint i, x is a vector of decision variables, and b is a vector of constants. To rewrite any objective or constraint in canonical form, we perform algebraic manipulations as necessary. We assume that there exists a mapping of variable names to positions in x, which we call the *order map* for $\mathcal{P}$. The order map is critical in MPM since it is the only object that allows the model to translate between variables named in what-if questions and columns in the constraint matrix.

For an example of conversion to a canonical form, refer to Figure **??**.

3.3 What-if Questions

In many cases, it may be desirable to modify an encoded mathematical optimization. For example, in response to changes in wholesale costs, a vendor who seeks optimized product prices may determine that the price for a particular object can no longer be less than some threshold. This translates to adding or modifying a constraint in the encoded problem. Similarly, the vendor may want to understand how removing a constraint changes the optimal solution. Both of these cases can be described as instances of *counterfactual analysis*.

Ultimately, our goal is to enable this type of analysis via natural language, regardless of technical expertise. As such, we define a *what-if question*, $\mathcal{Q}$, to be natural language inquiries regarding how the optimal solution to a mathematical program changes if some of its constraints change. As the name suggests, all such inquiries begin with the hypothetical marker phrase *What if*, for example: *What if the price of product p must be greater than x?*

In detail, all of our *what-if* questions include a requested change to the optimization problem, and subsequently the mathematical program (in canonical form), through natural language in the form of a question. These modification requests can result in changes to a particular constraint in the following ways:

- **Limit change**: This alters the bounds of a constraint. For example, *What if the maximum number of truck trips in a week was 10 instead of 5?*
- **Constant change**: This modifies a constant parameter value within the constraint. For example, *What if the discount on product p in region r was 30%?*
- **Constraint type reversal**: This modifies the direction of a constraint, such as changing an upper bound to a lower bound constraint. For example, *What if the truck should make at least 5 trips in a week instead of at most 5 trips?*
- **Variable interaction modification**: This affects how the variables interact within a constraint altering the sign of a variable's coefficient. For example, *What if the difference between the two variables is upper bounded instead of their sum?* (e.g., changing $x_1 + x_2 \leq 50$ to $x_1 - x_2 \leq 50$).

4 The WIQOR Dataset

In this section, we detail the construction of the WIQOR, the first dataset for the MPM task. Starting from a seed problem in NL4Opt Ramamonjison et al. (2023), we utilize a combination of heuristics, large language model (LLM) inference, and human evaluation to construct and filter all components of each MPM instance in WIQOR. Finally, we discuss dataset statistics, focusing on problem types and complexity, as measured by the number of decision variables.

4.1 Optimization Problems in Canonical Form

At the heart of each MPM instance in WIQOR is an optimization problem, written in canonical form. To construct these canonical formulations for WIQOR, we use—and optionally modify—problems from NL4Opt. NL4Opt is a dataset of optimization problems, where the goal is to take a natural language description of the optimization problem and generate the corresponding objective function and constraint matrix, in canonical form (Section 3.2). For each problem in NL4Opt, we utilize the target canonical formulation and provided order map.

Increasing complexity. Real-world optimization problems may include hundreds or thousands of decision variables and constraints. In contrast, NL4OPT has an average of 2.08 variables and 2.63 constraints per problem. Therefore, we augment MPM instances in the test split of the WIQOR with 5, 10, 20, or 30 new decision variables. At a high level, this is done by adding columns to the constraint matrix, and making corresponding changes to the order map. When adding new variables, we also modify existing constraints and add new constraints that resemble those that are already present. For example, if each variable in the original problem has an upper bound constraint, we introduce new upper bound constraints for the added variables. As another example, we expand existing sum constraints with new variables as well. Significant care must be taken so that the resulting problems are sensible. For a detailed accounting of our handling of corner cases, refer to Appendix **??**.

4.2 PROBLEM SPECIFICATIONS

In addition to optimization problems in canonical form, each MPM instance in WIQOR includes a problem specification. Recall that these specifications are natural language summaries of the problems (Section 3.3). Unlike the NL4OPT problems, our specifications do not explicitly name *any* of the decision variables or detail the constraints. In other words, there is insufficient information in a specification to generate the canonical form of the problem. Importantly, by virtue of being summaries, these specifications need not be modified when canonical formulations are extended with new variables.

We generate specification problems using a large language model (LLM). To do so, we begin by manually creating specifications to accompany the original problems in NL4OPT. We combine an optimization problem expressed in natural language from NL4OPT with the corresponding hand-written specification to form each in-context learning (ICL) example. Then, we employ Llama 3.1 70B Instruct model with ICL examples Dubey et al. (2024). The generated specifications are manually evaluated for quality and to ensure they do not mention variable names or counts, parameter values, or constants included in the constraint matrix. For examples of the specific prompt, see Appendix **??**.

Including Mathematics in Specifications. In practice, there are instances of specifications that include mathematical formulations of some of the problem constraints. For example, a specification might include text that implies the existence of an upper bound constraint on each variable of the form, To emulate this, we also include algebraic constraints formulated in LaTeXand include them in our specifications. To do so, we construct the set of constraint types in the constraint matrix, and supply generic, LaTeXformatted expressions of that each of those constraint types, and include them in the specification. For example, if the problem includes upper bound constraints, we append the text `The problem has the following constraint type:` $x_i \leq b$, to the end of the specification. For additional details, see Appendix **??**.

4.3 WHAT-IF QUESTIONS & MODIFIED CANONICAL FORMS

The final 2 pieces of a MPM task are the what-if question, $\mathcal{Q}$, and modified canonical form, $\mathcal{P}^\star$. We generate the both using the same reverse engineering procedure. Specifically, we begin by heuristically modifying the problems canonical form. For example, we might change a parameter value or a constraint limit. The full suite of possible modifications that can be made correspond to the suite of what-if question types (Section **??**). This change results in a modified canonical form that serves as $\mathcal{P}^\star$.

To generate the what-if question, $\mathcal{Q}$, the corresponds to the change, we utilize LLM inference. Again, we utilize Llama 3.1 70B Instruct, but this type using ICL examples that follow the ReAct paradigm Yao et al. (2023). As before, we begin by creating ReAct-style inputs. Then, we create ICL examples that consist of: i) NL4OPT natural language problem, ii) the corresponding canonical form, and iii) the modified canonical form, $\mathcal{P}^\star$. The model is tasked with generating the what-if question, $\mathcal{Q}$, that matches the change in the $\mathcal{P}$. We utilize demonstrations to generate each what-if question. All generated what-if questions are manually evaluated for correcteness. Specifically, we ensure that the what-if question accurately reflects the change between the original and modified canonical forms ($\mathcal{P}$ and $\mathcal{P}^\star$). We found that 92% of generated what-if questions are of sufficiently high quality to be included in the dataset; the rest are either filtered out or modified and included.

All generated what-if questions—including low quality generations before human modification—are included in a supplement to the dataset.

4.4 Dataset Statistics

The WIQOR dataset consists of 1,946 instances of the MPM task. As described above, the creation of each task instance is seeded with an example from NL4Opt. However, MPM instances include generated specifications rather than full descriptions of all variables and constraints; canonical forms may have an increased number of variables; each instance includes a generated what-if question; and the goal is to predict a modified canonical formulation, which matches the generated what-if question.

Table 1 provides a detailed breakdown of the dataset across training, development, and test splits, highlighting the number of instances for each split and the distribution of what-if question types. The majority of the dataset is composed of *constant change* and *limit change* questions. The remaining portion is comprised of more nuanced modifications, such as *constraint direction reversals* and *variable interaction modifications*. Table 2 focuses on the types of constraints impacted by these what-if questions. We note the prevalence of *linear* constraints in the dataset, which dominate the modifications, while ratio constraints and specialized forms such as xy and xby are less frequent.

Table 1: Distribution of data splits and types of *what-if* questions in the WIQOR dataset

Split type	**Count**	**What-If Question type**	**Count**
Train	994	Constant change	444
Dev	176	Limit change	520
Test	796	Constraint type reversal	286
		Variable interaction modification	315

Table 2: Distribution of constraint types in the WIQOR dataset

Constraint type	**Count**	**Constraint type**	**Count**
Sum	324	Linear	765
Lower Bound	186	xby	24
Upper Bound	169	xy	20
Ratio	77		

5 Experiments

We report performance of LLMs in the Llama 3.1 family on the WIQOR dataset under the ICL paradigm. Specifically, we experiment with the 8 and 70 billion parameter instruct variants as well as the 34 billion parameter code variant. We include Code Llama in our experiments since they have been reported to perform well with mathematics as well as reasoning tasks . The models are prompted using few-shot learning, with the demonstrations selected in 2 ways:

- **Random:** exemplars are randomly sampled from a set of human-written examples uniformly at random; and
- **Similarity:** exemplars are selected such that their corresponding what-if question is semantically similar to the test what-if question.

For exemplars selected via the Similarity approach, we also include chain-of-thought ...

Augmenting decision variables In addition to our experiments on the WIQOR test split, we created a separate dataset consisting of augmented problems where additional decision variables were introduced. This dataset, containing 100 data points, was manually evaluated. We tested the performance of Llama-3.1-70B Instruct on this augmented subset. The number of additional variables was varied across four different sets: 5, 10, 20, 30. For each subset, we assessed how the model performed as the number of decision variables increased, providing insight into its scalability and adaptability to more complex problem formulations.

6 RESULTS & ANALYSIS

Model Size	ICL Variant	Accuracy (%)
8B	Random	12.28
8B	Similarity	10.69
70B	Random	75.7
70B	Similarity	78.78

Table 3: Comparison of ICL strategies for different Llama models

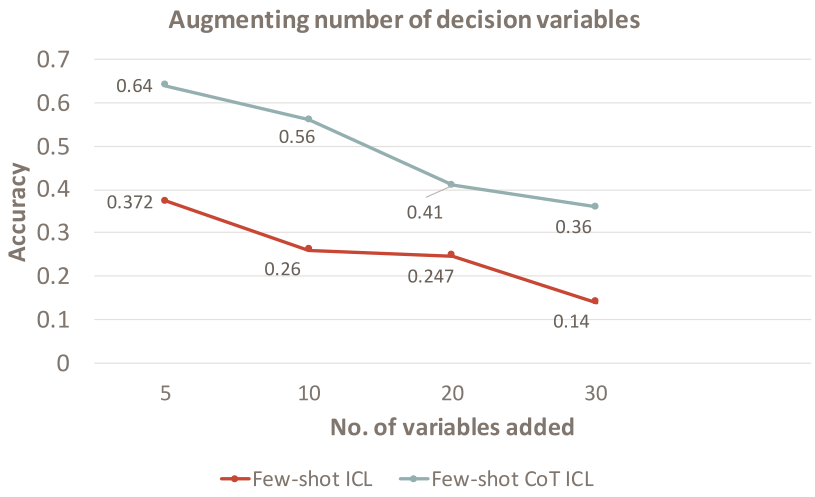

Figure 2: Accuracy v/s number of augmented variables

7 CONCLUSIONS

We introduce MPM, which is the task of generating a canonical formulation of an optimization problem from a problem specification, initial formulation, and a subject to a given what-if question. As such, the MPM task embodies the type of "what-if" analysis that is desirable among domain experts but is typically inaccessible. In addition, we present WIQOR, the first dataset of MPM instances. The dataset includes over 1000 problems, 7 constraint types, 4 what-if question types and varying levels of complexity, measured by the number of decision variables in the problem. Our experiments reveal that 8 billion parameter Llama 3.1 model only solves 10% of the easiest WIQOR problems. While the 70 billion parameter Llama 3.1 instruct model achieves 78% accuracy on the simplest problems in the dataset, when the number of decision variables increase, its performance drops to 36%. As such, we conclude that there is significant room for improvement before LLMs can assist in what-if analysis in real-world, industrial situations, where problems typically have one or two orders of magnitude more decision variables. We hope that our work will lead to techniques that empower domain experts to analyze the solutions to their optimization problems, without requiring cumbersome communication round-trips with third party technical experts.

8 FUTURE WORK

There are several promising directions for future work to extend the scope of this research and further evaluate the performance of large language models (LLMs) on optimization problem reformulation tasks. One potential area is the incorporation of more complex and diverse types of constraints. For example, we can introduce conditional constraints, where only one out of two constraints must be satisfied, but not both. This would increase the complexity of the problems and provide a more rigorous test for LLM capabilities.

Another avenue to explore is the impact of modifying multiple constraints simultaneously, allowing for a chain of what-if scenarios to be evaluated. By testing how LLMs handle sequences of modifications, we can better understand their ability to reason about cascading changes in problem formulations.

Fine-tuning LLMs on the training split of our dataset is another approach that could significantly improve performance. By allowing models to learn from this specific task, we can better assess their capabilities and limitations in reformulating optimization problems. Exploring these avenues could lead to meaningful advancements in the application of LLMs for optimization and what-if analysis tasks.

9 LIMITATIONS

AUTHOR CONTRIBUTIONS

If you'd like to, you may include a section for author contributions as is done in many journals. This is optional and at the discretion of the authors.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (mi)lp solvers and large language models, 2024. URL https://arxiv.org/abs/2402.10172.

Tasnim Ahmed and Salimur Choudhury. Lm4opt: Unveiling the potential of large language models in formulating mathematical optimization problems, 2024. URL https://arxiv.org/abs/2403.01342.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

Kris Johnson Ferreira, Bin Hong Alex Lee, and David Simchi-Levi. Analytics for an online retailer: Demand forecasting and price optimization. *Manufacturing & service operations management*, 18(1):69–88, 2016.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

Connor Lawless, Jakob Schoeffer, Lindy Le, Kael Rowan, Shilad Sen, Cristina St. Hill, Jina Suh, and Bahareh Sarrafzadeh. "i want it that way": Enabling interactive decision support using large language models and constraint programming, 2024. URL https://arxiv.org/abs/2312.06908.

Ganesh Prasath and Shirish Karande. Synthesis of mathematical programs from natural language specifications, 2023. URL https://arxiv.org/abs/2304.03287.

Lianhui Qin, Antoine Bosselut, Ari Holtzman, Chandra Bhagavatula, Elizabeth Clark, and Yejin Choi. Counterfactual story reasoning and generation. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5043–5053, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1509. URL https://aclanthology.org/D19-1509.

Rindranirina Ramamonjison, Timothy T. Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions, 2023. URL https://arxiv.org/abs/2303.08233.

Ivan Stelmakh, Nihar B Shah, and Aarti Singh. Peerreview4all: Fair and accurate reviewer assignment in peer review. In *Algorithmic Learning Theory*, pp. 828–856. PMLR, 2019.

Camillo J Taylor. On the optimal assignment of conference papers to reviewers. *University of Pennsylvania Department of Computer and Information Science Technical Report*, 1(1):3–1, 2008.

Dimos Tsouros, Hélène Verhaeghe, Serdar Kadıoğlu, and Tias Guns. Holy grail 2.0: From natural language to constraint models, 2023. URL https://arxiv.org/abs/2308.01589.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ.

Linzi Xing, Xinglu Wang, Yuxi Feng, Zhenan Fan, Jing Xiong, Zhijiang Guo, Xiaojin Fu, Rindra Ramamonjison, Mahdi Mostajabdaveh, Xiongwei Han, Zirui Zhou, and Yong Zhang. Towards human-aligned evaluation for linear programming word problems. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 16550–16556, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.1438.

Zhicheng Yang, Yinya Huang, Wei Shi, Liang Feng, Linqi Song, Yiwei Wang, Xiaodan Liang, and Jing Tang. Benchmarking llms for optimization modeling and enhancing reasoning via reverse socratic synthesis, 2024. URL https://arxiv.org/abs/2407.09887.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

A Appendix

You may include other additional sections here.