Training LLMs for Optimization Modeling via Iterative Data Synthesis and Structured Validation

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have revolutionized various domains but encounter substantial challenges in tackling optimization modeling tasks for Operations Research (OR), particularly when dealing with complex problem. In this work, we propose Step-Opt-Instruct, a framework that augments existing datasets and generates high-quality fine-tuning data tailored to optimization modeling. Step-Opt-Instruct employs iterative problem generation to systematically increase problem complexity and stepwise validation to rigorously verify data, preventing error propagation and ensuring the quality of the generated dataset. Leveraging this framework, we fine-tune open-source LLMs, including LLaMA-3-8B and Mistral-7B, to develop Step-Opt—a model that achieves state-of-the-art performance on benchmarks such as NL4OPT, MAMO, and IndustryOR. Extensive experiments demonstrate the superior performance of Step-Opt, especially in addressing complex OR tasks, with a notable 17.01% improvement in micro average accuracy on difficult problems. These findings highlight the effectiveness of combining structured validation with gradual problem refinement to advance the automation of decision-making processes using LLMs. The code and dataset are available at https://anonymous.4open.science/r/Step-Opt.

1 Introduction

Operations Research (OR) is a valuable discipline for addressing complex decision-making problems, widely applied in fields such as economics, engineering, and computer science (Bertsimas et al., 2019; Belgacem et al., 2020; Pereira et al., 2022). Effective implementation of OR involves two essential steps: modeling real-world problems and solving them. Despite significant advancements in solution techniques and the development of more efficient solvers, constructing appropriate models remains a challenge. Such a task requires formulating natural language descriptions into mathematical models, which is labor-intensive and demands domain-specific expertise as well as a deep understanding of modeling methodologies. These requirements greatly restrict the broader application of OR, particularly in real-world scenarios.

Recent developments in Large Language Models (LLMs) have enhanced the feasibility of automating optimization modeling. Approaches like Chain-of-Experts (CoE) (Xiao et al., 2023) and OptiMUS (AhmadiTeshnizi et al., 2024) employ well-crafted prompts and multi-agent systems to enhance the construction of optimization models and corresponding programs. However, these approaches rely on general-purpose LLMs, which, though powerful, are not specifically tailored for OR, limiting their effectiveness in addressing specialized challenges. Additionally, the need to upload sensitive data poses additional privacy concerns. In response, ORLM (Tang et al., 2024) presents an alternative by fine-tuning open-source LLMs using a dataset of 30K examples generated from 686 industry cases. While this improves the model's performance for OR modeling, ORLM remains semi-automated, requiring significant manual post-processing to achieve satisfactory results. Moreover, its prompt design lacks the precision needed to manage problem complexity and diversity, resulting in suboptimal outputs. Furthermore, modeling errors are not identified in real-time, allowing inaccuracies to persist and propagate. While rule-based post-processing can address minor errors, it often fails to rectify deeper logical and structural issues, further compromising data quality.

Utilizing high-quality training data is vital for improving the modeling capabilities of LLMs. However, current methods not only rely heavily on manual post-processing but also struggle to ensure data reliability. To address these limitations, we propose an approach from two primary perspectives. First, we enhance the prompt design and in-

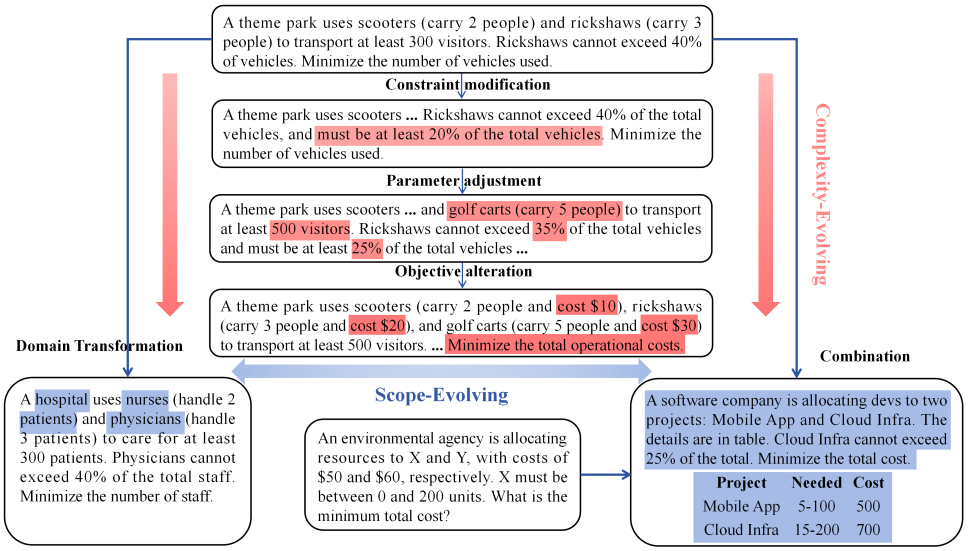

Figure 1: Examples of Iterative Problem Generation. It includes two types of methods: Complexity-Evolving, which refines problem complexity through constraint modification, parameter adjustment, and objective alteration; and Scope-Evolving, which enhances diversity via domain transformations and problem combinations. Red and blue backgrounds indicate changes introduced by Complexity- and Scope-Evolving, respectively. Repeated content is replaced with "..." for clarity.

troduce an Iterative Problem Generation, as shown in Figure 1. This method incrementally increases the complexity and scope of the problems, allowing the dataset to retain varying levels of difficulty and breadth. This diversity plays a crucial role in improving the model's generalization capabilities, as WizardLM (Xu et al., 2024) suggested. Second, we incorporate a stepwise validation mechanism that performs real-time checks throughout the generation process, effectively filtering out low-quality or erroneous data. This prevents errors from entering and propagating through the seed dataset. We refer to this framework as **Step-Opt-Instruct**. Our framework eliminates the need for post-processing, enabling fully automated generation while reducing API costs by utilizing only high-quality data for future iterations.

Step-Opt-Instruct consists of two key components: Iterative Problem Generation and Stepwise Validation Mechanism. The Iterative Problem Generation is specifically designed to address the unique challenges of OR-specific tasks, such as complex variable definitions and strict constraint implementation. By employing tailored methods such as Complexity-Evolving and Scope-Evolving, this approach generates a dataset enriched with enhanced complexity and diversity from the given one, facilitating the fine-tuning of LLMs to enhance their modeling ability for OR problems. As illustrated in Figure 1, Complexity-Evolving increases the complexity of the problem refining constraints, objectives, or parameters, while Scope-Evolving expands linguistic diversity and problem scope by adapting problems to new contexts or merging scenarios. This approach ensures that the generated dataset captures a wide range of complexities and provides robust coverage.

As new generated problems become increasingly complex, current LLMs often struggle to solve them accurately, resulting in errors. If these errors remain undetected and uncorrected, they will propagate through the iterative process, ultimately affecting the quality of the generated data. To mitigate this, the stepwise validation mechanism is implemented to not only prevent errors but also guarantee the accurate application of essential modeling techniques. Problems are first validated via a description checker for completeness, followed by checks on variables, constraints, and programs. Identified issues are resolved via feedback loops, with advanced techniques like the Big-M method verified using specially designed prompts that guide the LLM step-by-step to confirm accurate implementation. This validation process enables the generation of reliable and high-quality datasets, which are crucial for fine-tuning LLMs and enhancing their modeling ability for OR problems.

In order to evaluate the effectiveness of Step-Opt-Instruct, we collect 260 seed cases and generate nearly 4.5K examples. This data is then ap-

plied to train LLaMA-3-8B (AI@Meta, 2024) and Mistral-7B (Jiang et al., 2023), producing a model named Step-Opt. Furthermore, we manually review benchmarks including NL4OPT (Ramamonjison et al., 2023), MAMO (Huang et al., 2024), and IndustryOR (Tang et al., 2024), correcting a large number of examples with error labels. Experiments across these benchmarks indicate that our method outperforms existing approaches, achieving a 6.07% improvement in the micro average and a 7.93% enhancement in the macro average. Notably, when focusing on more complex components, Step-Opt exhibits a more significant advantage, attaining improvements of 17.01% and 12.26% in micro and macro averages, respectively. This substantial lead underscores our method's capability to manage complex problems effectively.

Our contributions are as follows:

• Introduction of advanced feedback mechanisms and real-time data updates, significantly reducing error propagation, eliminating the need for extensive manual post-processing.

• Development of Step-Opt-Instruct, a novel framework specifically designed to enhance the capabilities of open-source LLMs for effectively modeling OR problems.

• Proposal of the Step-Opt model, which achieves state-of-the-art performance across several benchmarks and particularly for complex problems, with additional manual corrections applied to errors in established benchmarks such as NL4OPT, MAMO, and IndustryOR.

2 Related Work

LLM-based Automated Modeling for OR is an emerging field that uses LLMs to generate mathematical models for OR problems. Existing methods can be categorized into prompt-engineering and fine-tuning. Approaches like Chain-of-Thought (Wei et al., 2022) and Reflexion (Shinn et al., 2024) improve performance but are not specialized for OR. More advanced methods, including OptiGuide (Li et al., 2023a), Chain-of-Experts (Xiao et al., 2023), and OptiMUS (AhmadiTeshnizi et al., 2024), employ multi-agent systems with LLM to construct models but encounter difficulties with complex problems due to LLM's limitations. ORLM (Tang et al., 2024), conversely, utilizes dataset generated from industry cases and GPT-4, coupled with rule-based post-processing, to fine-tune LLMs and improve outcomes. However, it lacks precise prompt and effective filtering mechanisms. Our framework addresses these limitations by iterative-based generation and real-time validation to control complexity and minimize errors, thereby enhancing the performance.

Data Augmentation improves LLM performance by generating synthetic datasets, often used when real-world data is insufficient for complex tasks(Wang et al., 2022; An et al., 2023; Oh et al., 2023; Pan et al., 2023; Gandhi et al., 2024; Xu et al., 2024; Zhou et al., 2024). In operations research, data augmentation approaches like (Prasath and Karande, 2023; Li et al., 2023b) focus on synthesizing optimization problems from natural language descriptions, but with limited complexity. ORLM (Tang et al., 2024) expands industry case datasets through modifications and rephrasings, while ReSocratic (Yang et al., 2024b) takes a reverse data synthesis approach, generating optimization scenarios from solutions. Among all these works, the closest to ours is Evol-Instruct (Xu et al., 2024), which uses In-depth Evolving and In-breadth Evolving to generate instruction data. However, as OR modeling presents unique challenges, we propose a stepwise validation mechanism to ensure accuracy and avoid error propagation in generated data.

3 Method

This section outlines the proposed framework. As depicted in Figure 2, It comprises two primary components: generators and a stepwise validation mechanism. The details of the generators are provided in Sec. 3.2, while the stepwise validation mechanism is detailed in Sec. 3.3.

3.1 Preliminary

We start generation from a given initial dataset, denoted as $D = \{(q_i, m_i)\}_{i=1}^{K}$, where each instance includes a problem description q_i and its associated mathematical model and program m_i. A valid q_i must contain an objective function, constraints, and all relevant parameters with specified numerical values. The model m_i implements the constraints and objective functions defined in q_i and produces executable code. An example of the training data is provided in Appendix A.1. The parameter K denotes the size of the initial seed dataset.

3.2 Generators

The problem generator adopts an iterative methodology, progressively producing problems with in-

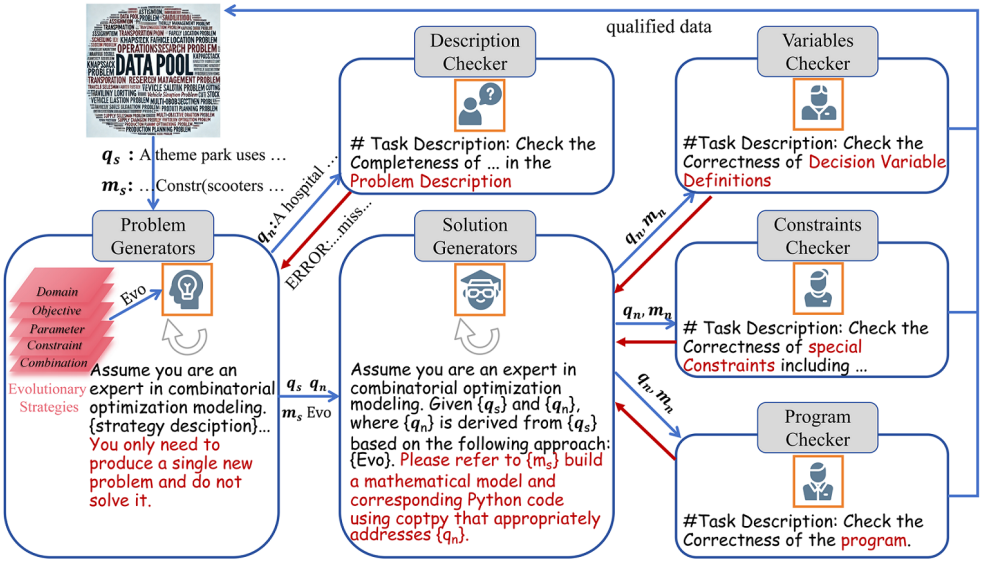

Figure 2: The framework of Step-Opt-Instruct. Each iteration begins by sampling seed data from an initial dataset. The Problem Generator uses evolving prompts to create a problem description, which is refined through feedback from the Description Checker. Once approved, the Solution Generator produces a solution, validated by the Variables, Constraints, and Program Checkers. Detected errors are fed back for revision. Only verified descriptions and solutions are added to the dataset. All components rely on tailored prompts to ensure quality. Red text indicates prompt customizations. The resulting dataset is then used to fine-tune LLMs, improving modeling capabilities.

creasing complexity and diversity. In each iteration, a seed data (q_s, m_s) is randomly sampled. A specific evolving method, denoted as f_e, is then applied to create a new problem description $q_n = f_e(q_s)$. This process employs prompt-based LLM methods to refine problem descriptions and systematically expand their scope. These methods can be categorized into two types: **Complexity-Evolving** and **Scope-Evolving**, as detailed below.

Complexity-Evolving increases complexity by modifying existing conditions or introducing new elements. Considering the specific characteristics of OR problems, three approaches are included: constraint modification, objective alteration, and parameter adjustment. These incrementally raise complexity while preserving logical integrity.

Constraint modification revises existing constraints or adds new ones to enhance the problem, with the core principle being to *"modify constraints based on the given problem while retaining its logical structure."* This ensures that the essential logic of the problem remains intact as complexity increases. Similarly, objective alteration either modifies objectives or introduces new ones, with the restriction that changes cannot merely adjust coefficients. Parameter adjustment changes values or adds elements. These approaches, while tailored to specific contexts, share the principle of preserving the underlying structure. Together, they enhance problem difficulty from various perspectives.

Nevertheless, the generated problems may become so complex that they exceed the processing capabilities of LLMs. To manage this, modifications to constraints or objectives are limited to one at a time, and parameter adjustments introduce at most one new entity. Specifically, the prompt for constraint modification is subject to strict limitations: only one constraint can be modified or added per iteration to control the growth in complexity. These restrictions not only ensure a balanced dataset with varying difficulty levels but also exclude excessively challenging examples, thereby improving the model's generalization capabilities. Prompt templates are provided in Appendix A.3.

Scope-Evolving broadens topic coverage and diversity by transforming the seed example into a different domain or combining it with another example to create a novel scenario. Domain transformation transfers the basic structure of the original problem to a new domain, while preserving its logic and constraints, thereby increasing linguistic and contextual diversity. To ensure practical relevance, we define a list of reference domains. Alternatively, the combination approach merges two distinct problems to create a new one that belongs to a different domain and contains unique details. This approach

introduces more substantial changes. To control complexity, the new problem is required to be of a similar length to one of the originals, maintaining manageable difficulty. Prompt templates for Scope-Evolving are provided in Appendix A.4.

As the Complexity- and Scope-Evolving progress, the complexity, scope, and diversity of generated data expand, ensuring comprehensive coverage across dimensions. All approaches use two-shot examples to maintain consistency.

Solution generator g produces a mathematical model and program m_n for a valid problem q_n. It generates $m_n = g(q_n, q_s, m_s, f_e)$ using q_s, m_s and evolving method f_e as references. Since LLMs may struggle with complex models, we embed the instruction *"ensuring the format and structure are as consistent as possible with the provided q_s and m_s"* into the meta-prompt to enforce consistency.

3.3 Stepwise Validation Mechanism

While the aforementioned generation methods can produce descriptions and solutions, the complexity of OR problem modeling poses significant challenges for current LLMs, often causing missing parameters, ambiguous objectives, or misused advanced techniques. Without sufficient supervision and error-correction mechanisms, such issues tend to persist, gradually undermining dataset quality and negatively impacting model performance.

To address these challenges, we design a stepwise validation mechanism that checks throughout the generation process, eliminating low-quality or erroneous data to maintain dataset integrity. This mechanism comprises four checkers, each focusing on a specific aspect: description completeness, variable definition, constraint implementation, and program quality. The description checker evaluates whether the generated q_n contains essential components. If any is missing, the checker provides feedback, prompting regeneration until validation succeeds or the attempt limit is reached. Only after passing this check does the solution generator produce the mathematical model and program.

Subsequently, additional checkers cross-reference q_n and m_n to conduct assessments. For variables, step-by-step instructions are provided, along with examples covering common types, enabling the checker to verify variable definitions.

The constraint checker ensures constraints are correctly formulated and aligned with the problem description. It follows a systematic process: identifying constraints, then verifying their consistency with the problem's content, similar to variable validation. While all constraints are reviewed, special attention is given to advanced techniques such as the Big-M method and K-way selection constraints. These serve as specialized checks, with other advanced techniques also applicable. Finally, the program checker extracts and executes the program, capturing outputs or errors, and providing feedback to the solution generator as needed.

When errors are identified in m_n, they are relayed back to the solution generator with the prompt: *"Please regenerate the solution based on the 'Error'. Ensure that the new solution correctly addresses the problem while maintaining the format and structure, with only the necessary corrections and improvements."* The revised solution undergoes further testing until it passes all validation stages. If the retry limit is reached, the problem will be discarded. This validation process ensures both q_n and m_n are error-free. Only data that pass all assessments are integrated into the dataset D for future iterations. This minimizes errors within D, thereby preventing the propagation of inaccuracies in future generations and safeguarding overall dataset quality. Details of the checkers and regeneration are provided in Appendix A.5

4 Experiment

4.1 Experimental Setup

Dataset. We assess our method using a range of datasets, spanning simple ones like NL4OPT (Ramamonjison et al., 2023) and MAMO EasyLP (Huang et al., 2024), and complex ones such as MAMO ComplexLP (Huang et al., 2024) and IndustryOR (Tang et al., 2024). Answers were manually revised when needed. Examples are shown in Appendix A.2.

NL4OPT originates from the NeurIPS 2022 NL4Opt competition and includes 1,101 simple linear programming problems (LPs), 289 used for evaluation. We correct 16 inaccurate instances.

MAMO contains two sub-datasets: EasyLP and ComplexLP. The former contains 652 simple LPs, and the latter 211 complex ones, all are paired with optimal solutions. We rectify 78 inaccuracies.

IndustryOR consists of 100 complex OR problems. Many lack essential information or accurate values, leading to 50 corrections and removal of 23 instances that fail to meet modeling criteria.

Baselines *tag-BART* (Kani and Gangwar, 2022) is a pre-trained language model (PLM) that won

Table 1: Performance comparison of methods. Values marked with a * are directly copied from original papers.

	Method	NL4OPT	MAMO EasyLP	MAMO ComplexLP	IndustryOR	Micro Avg	Macro Avg
PLMs	tag-BART	47.90%*	-	-	-	-	-
GPT-3.5	Standard	13.06%	35.58%	10.90%	6.49%	24.64%	16.51%
	CoT	33.06%	66.56%	13.27%	12.99%	46.67%	31.47%
	Reflexion	43.67%	67.64%	14.22%	15.58%	49.79%	35.28%
	CoE	52.24%	61.81%	17.06%	18.18%	49.03%	37.32%
GPT-4	Standard	72.65%	81.13%	24.64%	25.97%	65.74%	51.10%
	CoT	76.73%	84.97%	29.86%	25.97%	69.62%	54.38%
	Reflexion	78.78%	85.12%	36.02%	27.27%	71.05%	56.49%
	CoE	76.73%	84.36%	40.28%	31.17%	71.48%	58.14%
Fine-tune	ORLM	78.37%	84.20%	38.39%	35.06%	71.65%	59.01%
	Step-Opt-Mistral-7B	72.65%	82.06%	52.61%	**40.26%**	72.15%	61.90%
	Step-Opt-LLaMA-3-8B	**84.49%**	**85.28%**	**61.61%**	36.36%	**77.72%**	**66.94%**

1st place in the NL4Opt competition.

Standard, *CoT* (Chain-of-Thought) (Wei et al., 2022), and *Reflexion* (Shinn et al., 2024) represent typical prompting strategies, including direct generation, intermediate reasoning, and iterative feedback-based refinement.

Chain-of-Experts (CoE) (Xiao et al., 2023) is a multi-agent prompting framework leveraging interactions among LLMs to enhance problem-solving.

ORLM (Tang et al., 2024) is a fine-tuned model using a checkpoint from Hugging Face [1], along with 3K training examples[2], which we also use in ablation studies.

To ensure fairness, all methods are evaluated with temperature set to 0. Fine-tuned models use greedy decoding in a zero-shot context, selecting the top-1 completion as the solution. Step-Opt and ORLM use the COPT solver to ensure alignment with the raw data format. Prompt engineering methods are evaluated using GPT-3.5 (gpt-3.5-turbo-1106) and GPT-4 (gpt-4-turbo-2024-04-09), respectively. For additional comparison, we evaluate more advanced LLMs such as GPT-4o and Qwen2.5 (Yang et al., 2024a) on the MAMO ComplexLP, with results provided in Appendix A.6.

Details To construct the dataset, we begin with 260 examples and perform 8,400 iterations using GPT-4-turbo-0409, resulting in 4,464 examples. Further details on the instance generation can be found in Appendix A.7. We then fine-tune LLaMA-3-8B (AI@Meta, 2024) and Mistral-7B (Jiang et al., 2023) utilizing the LLaMA-Factory framework (Zheng et al., 2024) with the Alpaca format template (Taori et al., 2023), applying the LoRA technique (Hu et al., 2021) for efficient parameter adaptation. In this setup, the input consists of a fixed prompt with a problem description, and the output includes mathematical models and the corresponding programs. Hyperparameter details are provided in Appendix A.8. During inference, we employ greedy search in a zero-shot context, setting the max generation length to 2,048 tokens.

Metric. Considering the potential for minor discrepancies in numerical solutions, we define a comparison rule to account for small inaccuracies. Let o be the output of generated programs from different methods, and g denote the ground truth. The comparison is governed by the following criterion:

$$\left|\frac{o-g}{g+\epsilon}\right| \leq 10^{-4}, \tag{1}$$

Where ϵ is a small number to avoid division errors; o and g are equivalent if they satisfy Eq. 1.

4.2 Comparison Analysis

As shown in Table 1, Step-Opts based on LLaMA-3-8B and Mistral-7B significantly outperform baselines by a large margin, including tag-BART which achieves only 47.90% on NL4OPT despite requiring extensive manual constraint validation, lagging far behind LLM-based approaches. The best-performing Step-Opt, trained on LLaMA-3-8B, achieves state-of-the-art results on all benchmarks. This demonstrates its superior modeling capability. Notably, fine-tuned LLMs exceed the prompt engineering methods on average. However, the differences are less pronounced in the easier datasets, NL4OPT and MAMO EasyLP. The reason lies in the straightforward modeling requirements of these problems, which primarily require understanding

[1] https://huggingface.co/CardinalOperations/ORLM-LLaMA-3-8B

[2] https://huggingface.co/datasets/CardinalOperations/OR-Instruct-Data-3K

Table 2: Ablation Study on different evolving methods

Method	NL4OPT	MAMO EasyLP	MAMO ComplexLP	IndustryOR
Step-Opt	77.55%	85.43%	**36.02%**	**23.38%**
w/o Constraint Modification	75.92%	85.58%	19.91%	15.58%
w/o Objective Alteration	77.55%	**85.89%**	25.12%	19.48%
w/o Parameter Adjustment	73.06%	83.59%	26.07%	22.08%
w/o Domain Transformation	73.88%	83.13%	20.38%	18.18%
w/o Combination	**77.96%**	85.12%	33.65%	22.08%

Table 3: Comparison of Step-Opt and ORLM with 3K examples.

Method	NL4OPT	MAMO EasyLP	MAMO ComplexLP	IndustryOR	Micro Avg	Macro Avg
Step-Opt	**78.37%**	84.51%	**44.08%**	**32.47%**	**72.66%**	**59.86%**
ORLM	75.92%	**88.19%**	28.91%	25.97%	71.05%	54.75%

problem descriptions—a strength of models like ChatGPT and GPT-4. In contrast, for more complex datasets, the performance of fine-tuned models significantly improves, greatly exceeding that of prompt engineering methods. This indicates that fine-tuned models possess enhanced modeling capabilities. A prominent example is MAMO ComplexLP, where the advantage of Step-Opt-LLaMA-3-8B reaches 21.33%.

To emphasize the distinctions, we analyze results across simple and complex datasets using GPT-4 prompt engineering as the baseline compared with the top-performing Step-Opt model. As shown in Figure 3, nearly all methods perform well on simple datasets, with most achieving over 80% accuracy, except for the Standard method. The differences between methods on simple datasets are relatively minor. In contrast, the results for complex datasets demonstrate that advanced prompt engineering techniques, such as CoE, significantly outperform Standard, CoT, and Reflexion, though they still lag behind our proposed methods. Notably, Step-Opt achieves an accuracy above 50%, significantly surpassing existing methods and showcasing its superior modeling capabilities for complex problems. Given the intricate nature of complex problem descriptions and the advanced techniques required, our models exhibit a greater capacity to handle higher-order techniques.

4.3 Ablation Study

We conduct an ablation analysis to explore the effectiveness of different evolving methods and the composition of the training data, while also facilitating a fair comparison between OR-Instruct and Step-Opt-Instruct. For all ablation experiments, we set the hyper-parameters to the same and use LLaMA-3-8B as the backbone. The parameter settings can be found in the Appendix A.8. In addition, we further evaluate the Stepwise Validation Mechanism on the MAMO ComplexLP, the details are shown in Appendix A.9.

Study on evolving methods: Initially, we evaluate the survival rates of different generation methods, yielding the following results: 1,716 for constraint modification, 1,242 for objective alteration, 2,123 for parameter adjustment, 2,077 for domain transformation, and 455 for combination. The higher survival rates for parameter adjustment and domain transformation reflect their relative simplicity, allowing examples to pass evaluations more easily. Conversely, the combination is the most challenging, as it requires two sets of descriptions and solutions, often failing due to potential misalignment. The other two methods, which introduce new elements, are also more prone to errors.

Then, we randomly sample 2,000 examples without specific methods and train LLaMA-3-8B on this data. As shown in Table 2, removing domain transformation yields the worst performance, with a clear drop across all datasets, underscoring its critical importance. While parameter adjustment notably affects simpler benchmarks, its impact on complex datasets is limited. In contrast, both constraint modification and objective alteration exert a greater influence on complex datasets compared to easier ones. Particularly for constraint modification, it introduces additional constraints and increases the difficulty, facilitating the model's ability to process more complex conditions.

Study on the components of training examples: As described in Sec. 3, each training example includes a mathematical model and corresponding programs utilizing the COPT solver, though only

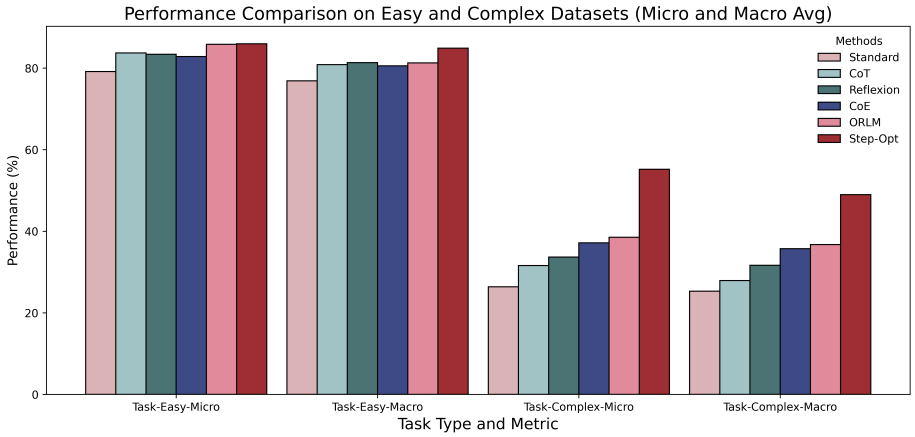

Figure 3: Performance comparison of various methods on easy and complex datasets.

Table 4: Comparison of Step-Opt and Step-Opt without mathematical model

Method	NL4OPT	MAMO EasyLP	MAMO ComplexLP	IndustryOR
Step-Opt	**84.49%**	**85.28%**	**61.61%**	**36.36%**
Step-Opt-4.73M	81.22%	84.97%	50.24%	33.77%
w/o mathematical model-4.73M	80.00%	81.44%	45.97%	29.87%

the program is used for problem-solving. To assess the impact of the mathematical model, we remove this component from the entire dataset and train LLaMA-3-8B. The results, presented in Table 4, reveal a significant performance drop upon the removal of the mathematical model. To further mitigate the influence of token count (as data without the mathematical model contain fewer tokens), we maintain a total of 4.73 million tokens across all datasets. Even with equivalent training sizes, the dataset including the mathematical model consistently outperforms the one without it. This improvement can be ascribed to the mathematical model functioning similarly to the Chain-of-Thought approach, providing a structured framework that guides the reasoning process in a systematic manner, effectively bridging the problem description and the code solution. In its absence, the model skips critical reasoning steps, leading to a significant reduction in performance.

Comparison of OR-Instruct and Step-Opt-Instruct: ORLM gathers 686 industry cases and creates 30,000 examples with the OR-Instruct framework, including 3,000 publicly available examples. To assess the performance of OR-Instruct in comparison to Step-Opt-Instruct, we randomly select 3,000 examples for evaluation. Both datasets, each comprising 3,000 examples, are used to train LLaMA-3-8B. As illustrated in Table 3, except for MAMO EasyLP, our method uniformly outperforms ORLM, achieving a 1.61% improvement in micro average and a 5.11% enhancement in macro average. The gains on more complex datasets, such as MAMO ComplexLP and IndustryOR, are even more pronounced. These advancements suggest that Step-Opt-Instruct possesses superior capabilities and generates higher-quality data, allowing LLMs to more effectively address OR problems, particularly those of greater complexity.

5 Conclusion

In this paper, we present Step-Opt-Instruct, a framework that integrates iterative problem generation with a stepwise validation mechanism to enhance the capabilities of LLMs in addressing complex OR problems. By progressively increasing problem complexity and ensuring data quality through real-time validation, Step-Opt-Instruct effectively prevents error propagation by removing low-quality data during the generation process. This approach enables full automation without relying on post-processing, ensuring high-quality datasets for fine-tuning. The resulting model, Step-Opt, achieved significant performance improvements across benchmarks such as NL4OPT, MAMO, and IndustryOR, particularly excelling in complex optimization tasks. These results highlight the effectiveness of combining systematic problem generation with structured validation to significantly enhance the modeling capabilities of LLMs.

Limitations: The proposed method faces difficulties in dealing with the wide variety of modeling techniques commonly used in OR, which limits its ability to handle the full range of possible scenarios. Moreover, the performance of the approach has not been fully tested across all types of OR problems. Finally, its broader application still needs to be tested in other fields to validate its applicability and adaptability.

References

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. 2024. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. *arXiv preprint arXiv:2402.10172*.

AI@Meta. 2024. Llama 3 model card.

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. 2023. Learning from mistakes makes llm better reasoner. *arXiv preprint arXiv:2310.20689*.

Ali Belgacem, Kadda Beghdad-Bey, Hassina Nacer, and Sofiane Bouznad. 2020. Efficient dynamic resource allocation method for cloud computing environment. *Cluster Computing*, 23(4):2871–2889.

Dimitris Bertsimas, Jack Dunn, and Nishanth Mundru. 2019. Optimal prescriptive trees. *INFORMS Journal on Optimization*, 1(2):164–183.

Saumya Gandhi, Ritu Gala, Vijay Viswanathan, Tongshuang Wu, and Graham Neubig. 2024. Better synthetic data by retrieving and transforming existing datasets. *arXiv preprint arXiv:2404.14361*.

Dongdong Ge, Qi Huangfu, Zizhuo Wang, Jian Wu, and Yinyu Ye. 2022. Cardinal optimizer (copt) user guide. *arXiv preprint arXiv:2208.14314*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2024. Mamo: a mathematical modeling benchmark with solvers. *arXiv preprint arXiv:2405.13144*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Nickvash Kani and Neeraj Gangwar. 2022. Tagged input and decode all-at-once strategy. https://github.com/MLPgroup/nl4opt-generation.

Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. 2023a. Large language models for supply chain optimization. *arXiv preprint arXiv:2307.03875*.

Qingyang Li, Lele Zhang, and Vicky Mak-Hau. 2023b. Synthesizing mixed-integer linear programming models from natural language descriptions. *arXiv preprint arXiv:2311.15271*.

Seokjin Oh, Su Ah Lee, and Woohwan Jung. 2023. Data augmentation for neural machine translation using generative language model. *arXiv preprint arXiv:2307.16833*.

Yan Pan, Davide Cadamuro, and Georg Groh. 2023. Data-augmented task-oriented dialogue response generation with domain adaptation. In *Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation*, pages 96–106.

João Luiz Junho Pereira, Guilherme Antônio Oliver, Matheus Brendon Francisco, Sebastiao Simoes Cunha Jr, and Guilherme Ferreira Gomes. 2022. A review of multi-objective optimization: methods and algorithms in mechanical engineering problems. *Archives of Computational Methods in Engineering*, 29(4):2285–2308.

Ganesh Prasath and Shirish Karande. 2023. Synthesis of mathematical programs from natural language specifications. *arXiv preprint arXiv:2304.03287*.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and 1 others. 2023. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, pages 189–203. PMLR.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Zhengyang Tang, Chenyu Huang, Xin Zheng, Shixi Hu, Zizhuo Wang, Dongdong Ge, and Benyou Wang. 2024. Orlm: Training large language models for optimization modeling. *arXiv preprint arXiv:2405.17743*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and 1 others. 2023. Chain-of-experts: When llms meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Zhicheng Yang, Yinya Huang, Wei Shi, Liang Feng, Linqi Song, Yiwei Wang, Xiaodan Liang, and Jing Tang. 2024b. Benchmarking llms for optimization modeling and enhancing reasoning via reverse socratic synthesis. *arXiv preprint arXiv:2407.09887*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

Kun Zhou, Beichen Zhang, Jiapeng Wang, Zhipeng Chen, Wayne Xin Zhao, Jing Sha, Zhichao Sheng, Shijin Wang, and Ji-Rong Wen. 2024. Jiuzhang3. 0: Efficiently improving mathematical reasoning by training small data synthesis models. *arXiv preprint arXiv:2405.14365*.

A Appendix

A.1 Example for training data

We use COPT (Ge et al., 2022) as the default solver in our experiments.

A.2 Examples for modifications of test sets

NL4OPT, Entry #228 : Wrong variable definition

Problem: A macro-counting fitness guru only eats salmon and eggs. Each bowl of salmon contains 300 calories, 15 grams of protein, and 80 mg of sodium. Each bowl of eggs contains 200 calories, 8 grams of protein, and 20 mg of sodium. Since the fitness guru has a limit to how many eggs he would like to eat, at most 40% of his meals can be eggs. The fitness guru needs to eat at least 2000 calories and 90 grams of protein. How many of each type of meal should he eat to minimize his sodium intake? **Answer:** 430.7692307692307

The answer is initially derived by treating the number of salmon and egg bowls as continuous variables. However, since the number of bowls should be integers, the correct solution is adjusted, and the actual answer is 460.

MAMO EasyLP, Entry #216 : Incorrect Handling of Absolute Value Constraint

Problem: A retail manager is planning to allocate resources across three different departments: purchasing (X), sales (Y), and logistics (Z). These departments have different cost per unit of resource allocated, with $5 for X, $3 for Y, and $4 for Z. The objective is to minimize the total cost while meeting certain operational constraints. The combined resources allocated to purchasing and sales cannot exceed 1000 units due to budget limitations. Similarly, the combined resources allocated to sales and logistics cannot exceed 800 units due to manpower availability. To ensure a balanced operation, the difference in resource allocation between purchasing and logistics should be at least 200 units. Given that each department has specific bounds on resource allocation (Purchasing can have up to 500 units, Sales up to 300 units, Logistics up to 200 units) and that allocations must be whole numbers due to indivisible nature of the resources being allocated:What is the minimum total cost required for this scenario? type of meal should he eat to minimize his sodium intake? **Answer:** 1000

The initial solution was derived without successfully establishing an absolute value constraint for "the difference in resource allocation between purchasing and logistics should be at least 200 units." Instead, only the constraint for one side (greater than or equal to 200) is retained, leading to an error. That is "model.addConstr(x - z >= 200, name=ResourceDifferenceConstraint)" in the program. The correct solution, considering both sides of the absolute value constraint, yields an actual cost of 800.

MAMO ComplexLP, Entry #216 : Incorrect Handling of Subtour Elimination

Problem: Imagine a logistics manager tasked with planning a delivery route for a truck that needs to visit four different cities to distribute goods. The cities are identified numerically as 1, 2, 3, and 4. The truck can start its journey from any of these cities but must travel to each city exactly once and then return to the starting point. The objective is to arrange this route in such a way that the total travel cost is minimized. The costs associated with traveling between the cities are as follows: The cost to travel from City 1 to City 2 is 52 units, to City 3 is 89 units, and to City 4 is 11 units. From City 2, it costs 52 units to reach City 1, 14 units to get to City 3, and 13 units to City 4. Traveling from City 3, the costs are 89 units to City 1, 14 units to City 2, and 87 units to City 4. Lastly, from City 4, it costs 11 units to go to City 1, 13 units to City 2, and 87 units to City 3. What is the minimum total travel cost for the truck to visit each city exactly once and return to the starting city? **Answer:** 50

The initial solution was derived without successfully establishing the subtour elimination constraint for the Traveling Salesman Problem (TSP). As a result, subtours were not eliminated properly, leading to an incorrect minimum total travel cost of 50 units. The correct solution, ensuring that subtours are eliminated and all cities are visited exactly once, yields an actual minimum total travel cost of 127 units.

IndustryOR, Entry #86: Missing Number

Problem: Fighter jets are important combat tools, but in order for them to be effective, there must be enough pilots. Therefore, in addition to a portion of the produced fighter jets being used directly for combat, another portion needs to be allocated for pilot training. It is known that the number of fighter jets produced each year is $a_j (j = 1, \cdots, n)$, and each fighter jet can train k pilots per year. How should the production of fighter jets be allocated each year to maximize their contribution to national defense over a period of n year? There is no numerical value for all parameters.

Problem Description q_i

A logistics company wants to allocate up to 250 vehicles between operations A and B. To ensure efficiency, the difference between twice the vehicles in A and those in B must be at least 60 units. Additionally, The number of vehicles in A plus twice the number in B should not exceed 300 units. Each vehicle costs $400 for A and $300 for B. What would be the minimum total cost (in dollars)?

Mathematical Model and Program m_i

Mathematical Model

Decision Variables:

x_A, x_B: Non-negative integer variables representing the number of vehicles allocated to A and B, respectively.

z: A binary variable to model the efficiency constraint.

Objective Function: Maximize: $Z = 400\ x_A + 300\ x_B$

Constraints:

1. Total number of vehicles constraint: $x_A + x_B \leq 250$.
2. Efficiency condition constraint (Introducing a sufficiently large constant M):

$$2x_A - x_B \geq 60 - M * z, \qquad x_B - 2x_A \geq 60 - M * (1-z),$$

3. Capacity condition: $x_A + 2x_B \leq 300$.

Program

```
import coptpy as cp
from coptpy import COPT
env = cp.Envr()
model = env.createModel("LogisticsVehicleAllocation")
x_A = model.addVar(vtype=COPT.INTEGER, name="x_A")
...
model.setObjective(400 * x_A + 300 * x_B, sense=COPT.MINIMIZE)
model.addConstr(x_A + x_B <= 250, name="TotalVehicles")
...
model.solve()
...
```

Figure 4: Examples of training data.

A.3 Prompt Templates for Complexity-Evolving

Prompt for objective alteration of Complexity-Evolving

Assume you are an expert in combinatorial optimization modeling. Modify the objective function to either transform the current objective into a different metric or add a new objective to convert it into a multi-objective optimization problem, while retaining its logical structure. The modifications or additions to the objective function should be substantial and not merely changes to coefficients. If there are already two or more objective functions, no new objectives may be added; only the existing objectives can be modified. The newly generated problems should align with real-world scenarios. **You only need to produce a single new problem and do not solve it.**

Given example1: {Here is Example1}

Given example2: {Here is Example2}

Given input: {Here is the original problem description}

Answer:

Prompt for parameter adjustment of Complexity-Evolving

Assume you are an expert in combinatorial optimization modeling. Adjust the parameters of the given problem while retaining its logical structure, constraints, and objective. When introducing a new entity, restrict the introduction to at most one new entity to control the complexity of the problem. The newly generated problems should align with real-world scenarios.
You only need to produce a single new problem and do not solve it.

Given example1: {Here is Example1}

Given example2: {Here is Example2}

Given input: {Here is the original problem description}

Answer:

A.4 Prompt Templates for Scope-Evolving

Prompt for constraint modification of Complexity-Evolving

Assume you are an expert in combinatorial optimization modeling.
Modify constraints or add new constraints based on the given problem while retaining its logical structure. Note that the modifications or additions to the constraints should be limited to a maximum of one. The newly generated problems should align with real-world scenarios.
You only need to produce a single new problem and do not solve it.

Given example1: {Here is Example1}

Given example2: {Here is Example2}

Given input: {Here is the original problem description}

Answer:

Prompt for domain transformation of Scope-Evolving

Assume you are an expert in combinatorial optimization modeling.
Transform the basic structure of the given problem into a different application domain while retaining its logical structure and constraints. The new application domain can include, but is not limited to, the following: the following: Education, Manufacturing, Logistics, Retail, Agriculture, IT Services, Healthcare, Event Planning, Construction, Entertainment, Research and Development, Hospitality, Defense, Energy Sector, Transportation, and Telecommunications.
You only need to produce a single new problem and do not solve it.

Given example1: {Here is Example1}

Given example2: {Here is Example2}

Given input: {Here is the original problem description}

Answer:

Prompt for combination of Scope-Evolving

Assume you are an expert in combinatorial optimization modeling. Given two problems (#Problem1 and #Problem2), generate a new problem. The new problem should be similar in length to one of the original problems but should belong to a different domain and have distinct specific details. The newly generated problem should align with real-world scenarios.

You only need to produce a single new problem and do not solve it.

Given example1: {Here is Example1}

Given example2: {Here is Example2}

Given input:

 # Problem1: {Here is the first problem description}

 # Problem2: {Here is the second problem description}

Answer:

A.5 Prompt Templates for checkers and regeneration

Prompt for regenerating the problem description

The #Problem is a generated problem but has some 'Error'. Please regenerate the problem description based on the 'Error'. Ensure that the new problem follows the same format and structure as #Problem, with only the necessary corrections and detail enhancements. **No solution or any other additional explanations are required.**

#Problem:

{generated_problem}

'Error':

{Error}

Example1:

#Problem:

{example1_problem}

'Error':

{example1_error}

'Regenerate':

{example1_regenerate}

Example2:

#Problem:

{example2_problem}

'Error':

{example2_error}

'Regenerate':

{example2_regenerate}

Answer:

Check the Correctness of Decision Variable Definitions

Important: The checks must be based on the problem description and common sense. No assumptions or conjectures should be made. The conclusions must be justified by the problem description or common sense.
Solution Description:To check the definitions of decision variables in the "## Mathematical Model:" for a combinatorial optimization problem, follow this structured approach:

Step 1: Extract Decision Variable Definitions
1. In the "## Mathematical Model:" section, find definitions under "### Decision Variables."
2. In the "## Python Code Solution Using coptpy:", identify definitions where model.addVar is used.

Step 2:Confirm Consistency with Problem Description
1. Ensure each variable's type and bounds align with the problem's actual meaning.

Step 3:Confirm Variable Types and Bounds
Note: The examples provided below are not exhaustive. Specific examples should be analyzed based on their actual meaning in the context of the problem.
Integer Variables (Bounds > 0): **{Examples for Integer Variables}**
Binary Variables (0 or 1): **{Examples for Binary Variables}**
Continuous Variables (0 or 1): **{Examples for Continuous Variables}**
Continuous Variables with Range: **{Examples for Continuous Variables with Range}**

Step 4: Check the Python Code Solution Using coptpy
1. For integer variables: Ensure vtype=COPT.INTEGER.
2. For continuous variables: Ensure vtype=COPT.CONTINUOUS.
3. For binary variables: Ensure vtype=COPT.BINARY.

If there are no errors, output: **"There are no errors found."**
If there are errors, output the specific errors with the format: **"ERROR: [description of error]"** and suggest how to fix them.
Please check for any errors in the variable definitions based on the steps above. **Do not repeat the prompt, only provide the errors and fixes if any, or confirm there are no errors.**

Task Description: Comprehensive Constraint Validation for OR Problems.

Important: The checks must be based on the problem description and common sense. No assumptions or conjectures should be made.
Solution Description: To verify the correctness of all constraints in the "## Mathematical Model" for an problem, follow this structured approach:

Step 1: Extract Constraint Definitions
1. In the "## Mathematical Model", identify constraints under "### Constraints."
2. In the "## Python Code Solution Using coptpy", find where "model.addConstr" is used.

Step 2: Validate Constraint Alignment with Problem Objectives
...
Step 3: Special Checks on Big-M Method Applications
1. **Absolute Value Constraints:** For constraints of the form $|x_i - x_j| \geq a$ $(a \geq 0)$,
verify the use of the Big-M method:
- Introduce a binary decision variable y for each constraint, and a sufficiently large constant M.
- Split into two constraints: $x_i - x_j \geq a - M * y$ and $x_j - x_i \geq a - M * (1 - y)$

2. **K-Way Selection Constraints:** At most K Selection (N types), confirm constraints are $\sum_{i=1}^{N} y_i \leq K$ and $x_i \leq M * y_i$, where y_i is a binary variable and M is a sufficiently large constant.
...
Step 4: Confirm Consistency with Python Code
Ensure that the constraints defined in the mathematical model are accurately translated into the code.

If no errors are found: "There are no errors found."
If errors are identified: **Output "ERROR:"** followed by the issue and advice for correction.

Prompt for regenerating the problem description

The #Problem is a generated problem but has some 'Error'. Please regenerate the problem description based on the 'Error'. Ensure that the new problem follows the same format and structure as #Problem, with only the necessary corrections and detail enhancements. **No solution or any other additional explanations are required.**

#Problem:
{generated_problem}
'Error':
{Error}
Example1:
#Problem:
{example1_problem}
'Error':
{example1_error}
'Regenerate':
{example1_regenerate}
Example2:
#Problem:
{example2_problem}
'Error':
{example2_error}
'Regenerate':
{example2_regenerate}
Answer:

Prompt for regenerating the solution

#Solution is the mathematic model and program of #Problem. An 'Error' was detected in #Solution. Please regenerate the solution based on the 'Error'. Ensure that the new solution correctly addresses the problem while maintaining the same format and structure as the original #Solution, with only the necessary corrections and improvements. No additional explanations are required.

#Problem: {Here is the generated problem description}

#Solution: {Here is the mathematic model and program for #Problem}

'Error': {Here is the error}

Given Example1: {Here is Example1}

Given Example2: {Here is Example2}

Answer:

A.6 Performance Comparison on MAMO ComplexLP

To further validate our results, we conducted comparative studies on the MAMO ComplexLP dataset, involving leading proprietary LLM GPT-4o-2024-08-06 and an advanced open-source LLM Qwen2.5-72B-Instruct. These comparisons provide additional context to the effectiveness of our Step-Opt framework. As shown in Table 5, proprietary models like GPT-4o demonstrate notable improvements, achieving a maximum accuracy of 54.03% with CoE and consistently outperforming earlier versions like GPT-4 and GPT-3.5. Similarly, open-source models such as Qwen2.5 achieve competitive results, with Reflexion reaching 47.87% and CoE achieving 51.66%. These findings indicate that open-source models are steadily narrowing the gap with proprietary counterparts, even without task-specific fine-tuning.

Despite these advancements, Step-Opt still demonstrates significant superiority, achieving the highest accuracy of 61.61% with Step-Opt-LLaMA-3-8B, surpassing GPT-4o and other baselines. Step-Opt-Mistral-7B also achieves 52.61%, further showcasing the effectiveness of our framework. These results emphasize the impact of Step-Opt's task-specific training data in elevating model performance across diverse problem formulations.

By generating high-quality, diverse datasets, Step-Opt addresses a key challenge in structured optimization tasks: enabling LLMs to better handle complex problems. The consistent performance of Step-Opt-trained models highlights the importance of integrating precise, task-specific data into fine-tuning pipelines, paving the way for more reliable and effective solutions to real-world optimization challenges.

A.7 Additional Details on Instance Generation

The instance generation involved 64K queries, and the number of tokens was 179M. On average, each generation iteration required approximately 7.66 queries, with 3.14 queries dedicated to generating and validating the problem description, and 4.52 queries used for solution generation and validation. Of the total tokens, 39M were allocated to generating and validating the problem description, while the remaining 140M were used for solution generation and validation. Additionally, 8,400 generations were conducted, yielding 4,464 samples, indicating that 46.86% of the generated samples were discarded.

A.8 Hyper-parameters for Training Step-Opt and baselines

All experiments are conducted on a single GPU server equipped with eight A100 GPUs, each with 40GB of memory. In experiment, we report the best results of all checkpoints. The maximum token is limited to 2,500. The hyper-parameters for training Step-Opts are as follows:

A.9 Study on Stepwise Validation Mechanism

To evaluate the impact of the proposed Stepwise Validation Mechanism, we conducted experiments on the MAMO Complex dataset. The results are summarized in Table 8. The results demonstrate that the Stepwise Validation Mechanism delivers consistent improvements across different GPT models when compared to Standard, CoT, and Reflexion methods. For GPT-4, our framework achieves the highest accuracy (42.18%), outperforming all other methods. However, CoE remains superior for GPT-3.5 and GPT-4o, reflecting the strength of its iterative reflection mechanism in these cases.

Table 5: Performance Comparison of Various Methods on MAMO ComplexLP

Method	Standard	CoT	Reflexion	CoE	Fine-Tuning
GPT-3.5	10.90%	13.27%	14.22%	17.06%	-
GPT-4	24.64%	29.86%	36.02%	40.28%	-
GPT-4o	46.92%	49.29%	48.34%	54.03%	-
Qwen2.5-72B-Instruct	46.45%	45.97%	47.87%	51.66%	-
ORLM	-	-	-	-	38.39%
Step-Opt-Mistral-7B	-	-	-	-	52.61%
Step-Opt-LLaMA-3-8B	-	-	-	-	**61.61%**

Table 6: Hyper-parameters for Training Step-Opts.

Backbone	BatchSize Per GPU	Gradient Accumulation	Learning rate	Epochs
Mistral-7B	4	8	1.25×10^{-4}	10
LLaMA-3-8B	4	8	1.25×10^{-4}	12

Table 7: Hyper-parameters for **ablation experiments**.

BatchSize Per GPU	Gradient Accumulation	Learning rate	Epochs
4	8	1.25×10^{-4}	10

In contrast, the Stepwise Validation Mechanism emphasizes real-time validation and correction during the modeling process, , avoiding the additional complexity of reflection. This streamlined approach proves particularly effective for LLMs, as demonstrated by its superior performance with GPT-4. Although CoE excels in certain cases, our method offers a robust and efficient alternative.

Additionally, It is important to consider the inherent difficulty of solving tasks directly during testing, as all methods must generate solutions from scratch. However, when used for data generation, the Stepwise Validation Mechanism can reference the solution of the original problem to generate solutions for new problems. By focusing only on the newly added or modified components, the mechanism significantly reduces the modeling difficulty. This advantage is not available during testing, where tasks must be solved entirely independently, but it underscores the potential of Stepwise Validation Mechanism for facilitating high-quality data generation.

Table 8: Comparison of Stepwise Validation Mechanism and Other Prompt Engineering Methods on MAMO ComplexLP

Method\Model	**GPT-3.5**	**GPT-4**	**GPT-4o**
Standard	10.90%	24.64%	46.92%
CoT	13.27%	29.86%	49.29%
Reflexion	14.22%	36.02%	48.34%
CoE	**17.06%**	40.28%	**54.03%**
Stepwise Validation Mechanism	16.59%	**42.18%**	**50.71%**

A.10 Limitations

The proposed method faces difficulties in dealing with the wide variety of modeling techniques, which limits its ability to handle the full range of possible scenarios. Moreover, the performance of the approach has not been fully tested across all types of OR problems. Finally, its broader application still needs to be tested in other fields to validate its applicability and adaptability.