Large Language Models are Good Multi-lingual Learners : When LLMs Meet Cross-lingual Prompts

Teng Wang[1], **Zhenqi He**[1], **Wing-Yin Yu**[2*], **Xiaojin Fu**[2], **Xiongwei Han**[3]
[1]Department of Mathematics, The University of Hong Kong, Hong Kong SAR, China
[2]Noah's Ark Lab, Huawei, Hong Kong SAR, China
[3]Noah's Ark Lab, Huawei, Shenzhen, China
Correspondence: {wt0318, zhenqi_he}@connect.hku.hk, {rocket.YuWingYin, fuxiaojin, hanxiongwei}@huawei.com,

Abstract

With the advent of Large Language Models (LLMs), generating rule-based data for real-world applications has become more accessible. Due to the inherent ambiguity of natural language and the complexity of rule sets, especially in long contexts, LLMs often struggle to follow all specified rules, frequently omitting at least one. To enhance the reasoning and understanding of LLMs on long and complex contexts, we propose a novel prompting strategy **M**ulti-**L**ingual **Prompt**, namely **MLPrompt**, which automatically translates the error-prone rule that an LLM struggles to follow into another language, thus drawing greater attention to it. Experimental results on public datasets across various tasks have shown MLPrompt can outperform state-of-the-art prompting methods such as Chain of Thought, Tree of Thought, and Self-Consistency. Additionally, we introduce a framework integrating MLPrompt with an auto-checking mechanism for structured data generation, with a specific case study in text-to-MIP instances. Further, we extend the proposed framework for text-to-SQL to demonstrate its generation ability towards structured data synthesis.

1 Introduction

Mixed Integer Programming (MIP) is a significant part of Operations Research (OR), which aims to solve optimization problems where some decision variables are constrained to be integers. It has been widely applied in industrial fields including logistics (Hulagu and Celikoglu, 2020), scheduling (Keha et al., 2009), and supply chain management (Sawik, 2011). Nowadays, benefiting from the development of Large Language Models (LLMs), automatically modeling complex and practical OR problems in plain text description to mathematical optimization formulas is no longer an impossible mission (Xiao et al., 2024; Wei et al., 2022; Yao et al., 2023a). CoE (Xiao et al., 2024) proposes a multi-agent system leveraging LLMs to model and program complex operations research problems and constructs a new dataset named ComplexOR involving intricate constraints, domain-specific terminology, and multi-step reasoning for various domains *e. g.*supply chain, scheduling, and logistics. Although the ComplexOR dataset provides foundational model metadata including *'set'*, *'parameter'*, *'hyper-parameter'*, *'variable'*, *'objective function'*, and *'constraint'*, concrete values of *'set'* and *'parameter'* are missing to generate MIP instances, causing limited applicability in developing autonomous MIP solvers for real-world optimization tasks.

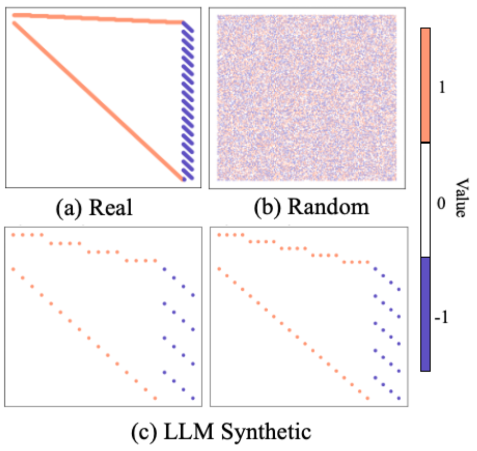


Figure 1: Semantic illustrations of the potential of LLMs in generating data following real distributions. (a) Distribution of Constraint Coefficients in Real-World Factory Location Problems (Cornuejols et al., 1977). (b) Distribution of Constraint Coefficients generated by random simulation. (c) Distribution of Constraint Coefficients generated by GPT-4 (Achiam et al., 2023) with different hyper-parameters. For each case: (demand points, candidate locations) = (4, 4), (5, 4).

With the success of data synthesis in various domains (Yang et al., 2024b; Zhang et al., 2023b,a; Li et al., 2023; He et al., 2023; Zhang et al., 2024), LLMs have shown great capabilities in synthesiz-

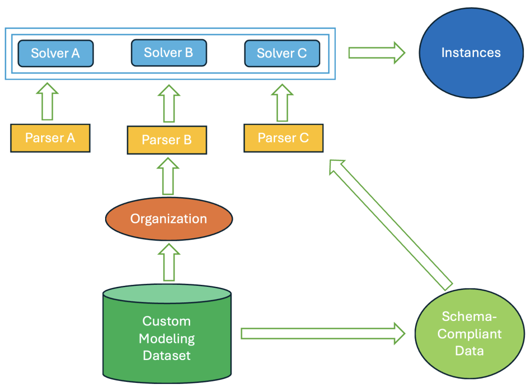


Figure 2: Illustration of the MIP instance generation process using key modeling components, including sets, parameters, variables, constraints, and objective functions. Only when compliant with the parser rules can the parser generate the modeling code, and the modeling tools produce instances.

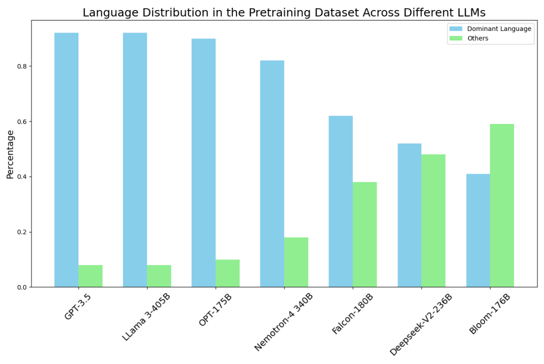


Figure 3: The dominant language refers to the majority language in the pretraining dataset, while "others" refer to all remaining languages. This figure demonstrates the language imbalance phenomenon in the pretraining data of GPT-3.5 (Brown et al., 2020), Llama-3 (Dubey et al., 2024), Deepseek-V2 (Liu et al., 2024), Bloom (Le Scao et al., 2023), Nemotron-4 (Adler et al., 2024), OPT (Zhang et al., 2022), and Falcon (Almazrouei et al., 2023).

ing realistic data in multiple modalities while the potential of LLMs in generating MIP instances data has not been fully explored. By comparing the synthetic distributions generated by GPT-4 (Achiam et al., 2023) with the real distribution of Constraint Coefficients of MIP instances and randomly generated distributions for Factory Location Problem (Cornuejols et al., 1977), shown in Fig. 1, LLMs have demonstrated its superiority in MIP instances generating compared with random simulation.

Fig. 2 demonstrates the autonomous MIP instance generation pipeline, incorporating modeling information such as sets, parameters, variables, constraints, and objective functions across multiple solvers. Commercial MIP solvers such as Gurobi (Achterberg, 2019), OptVerse (Li et al., 2024), and CPLEX (Bliek1ú et al., 2014) usually have its unique data representation with different parser rules for MIP instances, requiring various intricate rules in natural language to condition the generation of data that can be imported by the solvers for successful modeling. While direct handcrafted rules may lead to long and ambiguous contexts, LLMs have a high risk of neglecting certain rules to generate unsatisfactory instances given the complicated contexts.

To facilitate the autonomous generation of MIP instances within industrial pipelines, we propose a novel **M**ulti-**L**ingual **P**rompt algorithm, namely **MLPrompt**, specifically designed for structural data synthesis and adaptability for various solvers with different formats. Additionally, we propose a framework incorporating MLPrompt with an auto-checking mechanism to enable iterative prompt updates to ensure compliance with input constraints.

The design of MLPrompt is motivated by the following observations. The trilingual parallel language processing experiments (Pathak et al., 2024) have shown that more dominant languages receive greater cognitive focus, which facilitates faster response times and reduces the mental load when processing in non-dominant languages for polyglots. These findings are derived from mouse-tracking experiments (Pathak et al., 2024) that observed participants' language processing while listening to words in different languages and selecting corresponding images. Similarly, LLMs, such as ChatGPT (Achiam et al., 2023), function as polyglots, supporting over 80 languages. The scale of pretraining data varies across languages, as depicted in Fig. 3, where dominant languages frequently appear in the pretraining data, while non-dominant languages are comparatively underrepresented (Achiam et al., 2023). Hence, deriving from phenomena in human polyglots where the existence of non-dominant languages helps the process of dominant languages, MLPrompt is proposed to leverage the non-dominant languages of LLMs to strengthen the understanding on dominant languages.

Existing LLM-based reasoning approaches, such as Chain-of-Thought (CoT) (Wei et al., 2022), Tree-of-Thoughts (ToT) (Yao et al., 2023a), Self-

Consistency (SC) (Wang et al., 2023), and ReAct (Yao et al., 2023b), primarily focus on enhancing reasoning through iterative steps. These approaches aim to improve the quality of generated outputs by verifying the correctness of intermediate steps and selecting the best path forward. However, when generating structured outputs like JSON that must adhere to given constraints, it poses challenges to break the task into smaller and independent parts due to the interconnections within each part. In contrast, MLPrompt tackles structured data synthesis with introducing non-dominant languages to raise LLMs' attention in error-prone constraints reduce inference time without requiring multiple inference steps, all while preserving the interconnected relevance of the constraints.

In this paper, we propose a general MIP instances synthesis pipeline with a novel prompting method. Our contributions are three-fold: (1) We introduce **MLPrompt**, a simple yet effective multiple-lingual prompting strategy, for enhancing LLM reasoning, inspired by the capability of cross-lingual understanding in polyglots. (2) We make the first attempt at a LLM for MIP instance generation which serves as a bridge to connect existing research-based datasets with industrial needs, and it is easy to be extended into a general pipeline for structured data synthesis. (3) Extensive experiments on the ComplexOR dataset demonstrate the superiority of our prompt strategy compared to existing prompting strategies for MIP instance generation. Moreover, additional demonstration on the text2SQL(Yu et al., 2018; Cao et al., 2024) task highlights the broader applicability of our framework in other structured data generation tasks.

2 Related Work

2.1 MIP Instance Generation

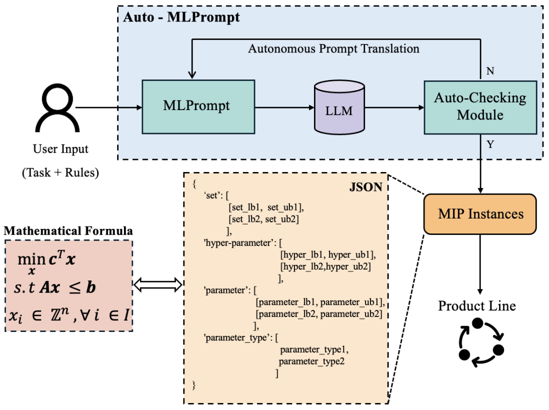
MIP instance generation plays a crucial role in the development of commercial MILP solvers (Rimmi Anand and Kumar, 2017). Traditional mathematical-formulation-based methods for MIP instance generation include TSP (Wiel and Sahinidis, 1995), structure-based instance generation (Bixby et al.; Applegate et al., 2006), mixed-integer knapsack (Atamtürk, 2003), and set covering (Balas and Ho, 1980; Anureet Saxena and Lejeune, 2010). G2MILP (Geng et al., 2023) introduces the first learning-based MILP instance generation framework, representing MILP problems as bipartite graphs and using a masked variational autoencoder (VAE) (Kingma and Welling, 2019) to iteratively generate new instances. ACM-MILP (Guo et al., 2024) presents an adaptive and structure-preserving approach by using a community detection algorithm to group strongly related constraints for collective modification. MILPGen (Yang et al., 2024c) simplifies bipartite graph representations of MILP instances into tree-like structures and uses graph convolutional networks (GCNs) to calculate node embeddings, allowing for optimal node pair merging. This enables the creation of larger, more complex instances while maintaining the original structural characteristics. Building upon these approaches, our method generates general MILP instances for a variety of applications, such as scheduling, logistics, and product management, through the use of input text descriptions. Specifically, we leverage the modeling information from the ComplexOR dataset to produce MIP instances, aligning with the broader goal of addressing real-world industrial needs.

2.2 Prompt Engineering for LLMs

Prompts are gradient-free strategy to strengthen LLMs' reasoning for complex tasks. The Chain-of-Thought (CoT) (Wei et al., 2022) decouples complex reasoning tasks into intermediate reasoning steps. Auto-CoT (Zhang et al., 2023c) automate CoT by by encouraging LLMs to *think step by step* to reduce manual operations in prompting. Self-Consistency (SC) (Wang et al., 2023) introduces a novel decoding strategy to enhance the reasoning capabilities of LLMs when using CoT prompting. Instead of relying on a single reasoning path derived from greedy decoding, it samples multiple diverse reasoning paths and determines the final answer by marginalizing over these paths to select the most consistent one. Tree-of-thought (ToT) (Yao et al., 2023a) and Graph-of-thought (GoT) (Besta et al., 2024) enhance the reasoning capabilities of LLMs through structured prompting schemes that go beyond traditional linear approaches like CoT.

2.3 Multilingual LLMs

With the success of English-center LLMs across various NLP tasks such as Question Answering (Kočiský et al., 2018) and Summarization (Hermann et al., 2015), increasing attention has been drawn to multilingual LLMs due to globalization. A multilingual LLM possesses the ability to process and produce content in multiple languages


Auto - MLPrompt
Autonomous Prompt Translation
N
MLPrompt
LLM
Auto-Checking Module
Y
User Input
(Task + Rules)
JSON
MIP Instances
Mathematical Formula
$\min_{\boldsymbol{x}} \boldsymbol{c}^T\boldsymbol{x}$
$s.t\ \boldsymbol{Ax} \leq \boldsymbol{b}$
$x_i \in \mathbb{Z}^n, \forall i \in I$
Product Line

```
{
    'set': [
            [set_lb1,  set_ub1],
            [set_lb2, set_ub2]
           ],
    'hyper-parameter': [
                        [hyper_lb1, hyper_ub1],
                        [hyper_lb2,hyper_ub2]
                       ],
    'parameter': [
                  [parameter_lb1, parameter_ub1],
                  [parameter_lb2, parameter_ub2]
                 ],
    'parameter_type': [
                       parameter_type1,
                       parameter_type2
                      ]
}
```

Figure 4: The workflow of the proposed framework for structured data generation. The process includes prompt construction with predefined rules, data generation by the LLM, evaluation of compliance with the rules, and iterative refinement of the prompt by translating rules into other languages if necessary.

simultaneously. Existing approaches for Multilingual LLMs are mainly split into two ways - (1) Parameter-tuning the LLMs with multilingual data (Du et al., 2022; Mendonca et al., 2023) (2) Parameter-frozen with prompting (Hoang et al., 2024). In this paper, we focus on the analysis of increasing the understanding and reasoning capabilities of LLMs trained with multilingual data such as GPT (Achiam et al., 2023) with proposed MLPrompt.

3 Methodology

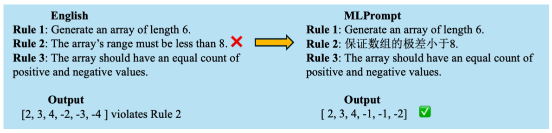
3.1 Problem Statement

An Mixed-Linear-Integer-Programming (MLIP) problem can be formulated as follows:

$$\begin{aligned} &\min_{\boldsymbol{x}} \boldsymbol{c}^T\boldsymbol{x} \\ &\text{s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ &\boldsymbol{x}_i \in \mathbb{Z}^n \text{ for } \forall i \in I, \end{aligned} \tag{1}$$

where $\boldsymbol{c} \in \mathbb{R}^n$ is the vector of objective coefficients, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is the matrix of constraint coefficients, $\boldsymbol{b} \in \mathbb{R}^m$ is the vector of constraint bounds, and $\boldsymbol{x} \in \mathbb{R}^n$ represents the decision variable. The index set I indicates the decision variables $\boldsymbol{x}_i$ constrained to be integers.

Here, $\boldsymbol{A}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are parameters to be generated, and their dimensions n, m will be defined in the given modeling problem. To simplify the generation and avoid unexpected long outputs, we formulate the MLIP instances into JSON data to record the data types, along with the lower and upper bounds for sets, parameters, and hyper-parameters as shown in Fig. 4. Then a random process generates values within the generated bounds to be fed into the solver's parser to construct the MLIP instance. By that, we successfully convert the MLIP instance generation problem into rule-based structured data generation, and the core is to employ the LLM to produce the corresponding JSON.



English
Rule 1: Generate an array of length 6.
Rule 2: The array's range must be less than 8. ✗
Rule 3: The array should have an equal count of positive and negative values.

Output
[2, 3, 4, -2, -3, -4] violates Rule 2

MLPrompt
Rule 1: Generate an array of length 6.
Rule 2: 保证数组的极差小于8.
Rule 3: The array should have an equal count of positive and negative values.

Output
[2, 3, 4, -1, -1, -2] ✅

Figure 5: A demo of how MLPrompt builds prompts.

3.2 MLPrompt

Generally, user-specific parsers are used in industries and academic teams for mathematical modeling and MIP instance generation (Xing et al., 2024; Wang et al., 2024b), leading to complex and interdependent rules to condition the generation of desired structured data. To address the complex constraints in natural language, we propose **MLPrompt** to leverage the multilingual capabilities of LLMs to draw LLMs attention to error-prone rules by translating any rule that is not followed into a non-dominant language of LLMs and hence to improve the quality of the generated data.

MLPrompt Inspired by the phenomenon in human polyglots where the existence of non-dominant languages helps the process of dominant languages (Pathak et al., 2024), we propose the **MLPrompt** to translate the error-prone rule into a non-dominant language of LLM to strengthen the understanding and reasoning of LLM in given complex contexts. A demonstration of this approach is presented in Fig. 5.

Here, we define the dominant language as the most frequently occurring language within the pre-training dataset, while all other languages, except the dominant one, are referred to as "others.". Prior work on Falcon (Almazrouei et al., 2023) highlights the challenge that arises when incorporating a substantial amount of multilingual data (e.g., exceeding 10%) in language models—leading to a decline in performance on tasks that are more aligned with the dominant language. Consequently, many state-of-the-art LLMs prioritize training on large volumes of dominant language data, further exacerbating the imbalance across languages. Fig. 3 demonstrates that while LLMs are designed with multilingual capabilities in mind, there is a prevalent tendency to disproportionately prioritize dominant language exposure, primarily to optimize performance on tasks aligned with the dominant language.

Auto-MLPrompt Strategy As introduced in Sec. 3.1, we formulate the MIP instance generation as a conditioned structured data synthesis constrained by natural language rules. It is relatively straightforward to validate whether JSON data adheres to the given constraints and to identify the specific rule or rules that have been violated. With such property in the rule-based JSON generation, we design an autonomous MIP generation pipeline with the proposed MLPrompt strategy to automatically detect the violated rule and translate it into a non-dominant language. The generation flow can be summarized as follows: Initially, we incorporate the predefined rules and input modeling information directly to generate the initial prompts. Then, the LLM generates structured data, like JSON, based on the initial prompt, and an evaluation function assesses whether the generated data complies with the given rules. If the error or misgeneration is detected, the corresponding rule will be translated into a non-dominant language to update the prompt, and the data generation process will be repeated.

4 Experiments

4.1 Dataset

ComplexOR (Xiao et al., 2024) contains 60 complex operations research problems with natural language description and corresponding mathematical formula. For each problem, it contains comprehensive information (an example is shown in Appendix A.1) for constructing models, including problem backgrounds, sets, parameters, hyper-parameters, variables, objective functions, and constraint functions. To generate the data following given conditions, we convert this extensive context into a set of rules for the LLMs to follow, requiring the output JSON to include the data type, lower bound, and upper bound for each set, parameter, and hyper-parameter. Subsequently, a random process is simulated to generate values for these sets and parameters. The solver's parser then reads the model-building information, loads the generated random data, and constructs the MIP instance accordingly.

4.2 Experimental Settings

As described in Section 4.1, we use the ComplexOR (Xiao et al., 2024) dataset (an example of binpacking problem is shown in Appendix A.1) for our experiments and utilize an LLM to generate a JSON file constrained by input rules. The rules are outlined in Appendix A.2, and we evaluate the model's performance based on rules 4, 7, and 8. A detailed analysis of the model's compliance with these rules is provided in Appendix A.4. We also implement an evaluation function to check whether the generated JSON complies with each rule, compute the accuracy for each rule, and calculate the final score by averaging the accuracies of the three rules.

We evaluate our proposed prompting strategy, MLPrompt, on various LLMs of different sizes, categorizing them into three groups based on the number of parameters. (1) **Small-scale LLMs**: LLMs with fewer than 10 billion parameters are classified as small. We use open-source models such as Mistral-7B (Jiang et al., 2023), Llama-3-8B (Dubey et al., 2024), Gemma-2-9B (Team et al., 2024), Qwen2-7B (Yang et al., 2024a), Llama-3.1-8B (Dubey et al., 2024), Qwen1.5-7B (Bai et al., 2023) and Openchat-3.5-7B (Wang et al., 2024a). (2) **Medium-scale LLMs**: LLMs with parameter sizes between 50 billion and 200 billion are considered medium-sized. These include GPT-3.5 (Brown et al., 2020), Mixtral-8$\times$7B (Jiang et al., 2024), Llama-3.1-70B (Dubey et al., 2024), Deepseek-67B (Liu et al., 2024), Llama-3-70B (Dubey et al., 2024), Qwen2-72B (Yang et al., 2024a), WizardLM-8$\times$22B (Xu et al., 2023) and Mixtral-8$\times$22B (Jiang et al., 2024). (3) **Large-scale LLMs**: We define LLMs with more than 200 billion parameters as large models. For this category, we focus on the GPT-4 series and conduct experiments using GPT-4o, GPT-4o mini, and GPT-4-Turbo (Achiam et al., 2023).

For each model scale, we compare MLPrompt with CoT (Wei et al., 2022), ToT (Yao et al., 2023a) and SC (Wang et al., 2023)

4.3 Small-scale LLMs

In our experiments, the input contexts are long and comprehensive, requiring LLMs to have a strong capability of understanding and reasoning of long contexts, while small models often fail to comprehend our intentions and fail to generate the correct JSON format by input constraints. Hence, for

Methods	Mistral-7B	Llama-3-8B	Llama-3.1-8B	Gemma-2-9B	Qwen1.5-7B	Qwen2-7B	Openchat-3.5-7B
Zero-shot	0.211	0.268	0.016	0.302	**0.017**	0.100	0.250
Few-shots	0.178	0.062	0.032	0.326	0.017	0.033	0.100
CoT(Wei et al., 2022)	**0.350**	**0.300**	0.167	**0.717**	0.000	0.133	0.233
ToT(Yao et al., 2023a)	0.300	0.033	**0.350**	0.033	0.000	0.233	**0.433**
SC(Wang et al., 2023)	0.267	0.067	0.333	0.050	0.000	**0.250**	0.100

Table 1: The table presents the success rate of small-scale LLMs generating correctly formatted JSON under various prompting methods, without considering rule compliance.

experiments on the small-scale model, we only consider the success rate of generating the correct JSON format by LLMs, without assessing rule compliance.

Natural language, unlike formal logic, lacks clear evaluative properties, making it difficult to assess the accuracy of generated outputs. This limitation poses a challenge for methods like CoT, ToT, and SC, which struggle to handle this ambiguity effectively. Additionally, the interconnected nature of rules in structured outputs, such as JSON, makes it difficult to verify each step in isolation—correctness in one part does not necessarily guarantee overall correctness when the parts are combined. To effectively utilize SOTA prompting methods, in the following experiments, CoT, ToT, and SC are applied without evaluating intermediate results. Detailed explanations are provided in Appendix A.5.

The performance, as shown in Table 1, indicates that these small LLMs struggle to generate the desired JSON format and improve the quality of the generated data under prompting methods such as CoT, ToT, and SC. Due to their limitations in handling complex tasks, we will not use these small LLMs in further experiments.

4.4 Medium-scale LLMs

Medium LLMs have the capability to comprehend our requirements and generate correctly formatted JSON. We evaluate these medium LLMs to determine whether the generated data adhere to the rules we have defined. The final results, as shown in Table 2, demonstrate that MLPrompt effectively improves the quality of generating complex data by translating a single rule from one language to another. This approach is more efficient compared to methods like ToT and SC, which involve multiple steps to obtain intermediate results, then combine and infer from these results to reach the final output.

The poor performance of CoT, ToT, and SC, as shown in Table 2, can be attributed to our inability to evaluate the intermediate results required by these methods in generating complex data task. As discussed in Section 4.3, these medium LLMs can be considered as weak learners, and their intermediate outputs often contain errors. As a result, combining multiple weak learners not only fails to improve the quality of the final results but may even degrade them. In contrast, similar to bagging, only when combining the results of strong learners can performance improvements be expected (Bhavan et al., 2019).

Furthermore, as shown in Table 2, adding the missing rule in another language alongside the existing one not only fails to outperform the replacement method but, in some cases, performs worse than the baseline. This approach also appears influenced by the re-adding prompt method, which undermines the purpose of the ablation experiment. Therefore, in the subsequent experiments, we will focus solely on the MLPrompt strategy using the replacement approach.

4.5 Large-scale Models

In this section, we evaluate the performance of large-scale models, specifically from the GPT-4 series, across various prompting strategies. GPT-4o served as the base model, and the results, as shown in Table 3, demonstrate several key findings. The baseline performance in a zero-shot setting is reasonable, and improvements are achieved using CoT, ToT, and SC prompting methods, which showcase their ability to refine model performance by structuring reasoning steps.

However, the most significant gains are observed with the MLPrompt strategy, which replaces key rules with alternative languages such as Mandarin, Thai, and Korean. Across all model variations, MLPrompt consistently yielded the highest performance, with Mandarin replacement proving particularly effective. This strategy outperformed traditional multi-step prompting methods like CoT, ToT, and SC, not only improving accuracy but also enhancing the efficiency of the models by reducing

Methods	GPT-3.5	WizardLM-8×22B	Mixtral-8×7B	Mixtral-8×22B	Llama-3.1-70B	Llama-3-70B	Deepseek-67B	Qwen2-72B
Baseline	0.436	0.250	0.133	0.239	0.378	0.461	0.083	0.383
CoT(Wei et al., 2022)	0.156	0.389	0.128	0.261	0.428	0.150	**0.128**	0.583
ToT(Yao et al., 2023a)	0.294	0.339	0.011	0.128	0.167	0.083	0.061	0.606
SC(Wang et al., 2023)	0.131	0.217	0.011	0.078	0.167	0.117	0.000	0.611
Repeated Missed Rule	0.133	0.244	0.067	0.228	0.328	0.183	0.056	0.528
MLPrompt + Mandarin	0.378	0.283 ↑ 3.3%	0.039	0.239	0.472 ↑ 9.4%	0.194	0.094 ↑ 1.1%	**0.633 ↑ 25.0%**
MLPrompt + Thai	0.383	0.272 ↑ 2.2%	0.011	**0.306 ↑ 6.7%**	**0.500 ↑ 12.2%**	0.372	0.044	0.500 ↑ 11.7%
MLPrompt + Korean	0.372	0.350 ↑ 10.0%	0.044	0.256 ↑ 1.7%	0.483 ↑ 10.5%	0.233	0.050	0.572 ↑ 18.9%
MLPrompt ↔ Mandarin	0.414	0.383 ↑ 13.3%	**0.211 ↑ 7.8%**	0.289 ↑ 5.0%	0.411 ↑ 3.3%	0.483 ↑ 2.2%	0.117 ↑ 3.4%	0.422 ↑ 3.9%
MLPrompt ↔ Thai	0.454 ↑ 1.8%	0.294 ↑ 4.4%	0.150 ↑ 1.7%	0.294 ↑ 5.5%	0.394 ↑ 1.6%	0.456	**0.128 ↑ 4.5%**	0.428 ↑ 4.5%
MLPrompt ↔ Korean	**0.591 ↑ 15.5%**	**0.406 ↑ 15.6%**	0.089	0.267 ↑ 2.8%	0.444 ↑ 6.6%	**0.533 ↑ 7.2%**	0.106 ↑ 2.3%	0.339

Table 2: The table presents the accuracy of each medium-scale model across various settings. MLPrompt consistently enhances data quality. The "+" symbol denotes the addition of a new language rule to complement an unmet rule, while "↔" signifies the replacement of the original rule with an existing one.

Methods	GPT-4*o*	GPT-4*o* mini	GPT-4-Turbo
Baseline	0.472	0.625	0.753
CoT(Wei et al., 2022)	0.492	-	-
ToT(Yao et al., 2023a)	0.530	-	-
SC(Wang et al., 2023)	0.619	-	-
Repeated Missed Rule	0.536	-	-
MLPrompt ↔ Mandarin	**0.844 ↑37.2%**	**0.888 ↑26.3%**	0.874 ↑12.1%
MLPrompt ↔ Thai	0.813 ↑34.1%	0.816 ↑19.1%	0.777 ↑2.4%
MLPrompt ↔ Korea	0.796 ↑32.4%	0.613	**0.892 ↑13.9%**

Table 3: Performance evaluation of large-scale LLMs, comparing GPT-4 series under different prompting strategies, including CoT, ToT, SC, and Repeated-missed-rule.

inference time and improving the overall quality of the generated data. As such, MLPrompt demonstrates its superiority in handling complex data generation tasks in large-scale models.

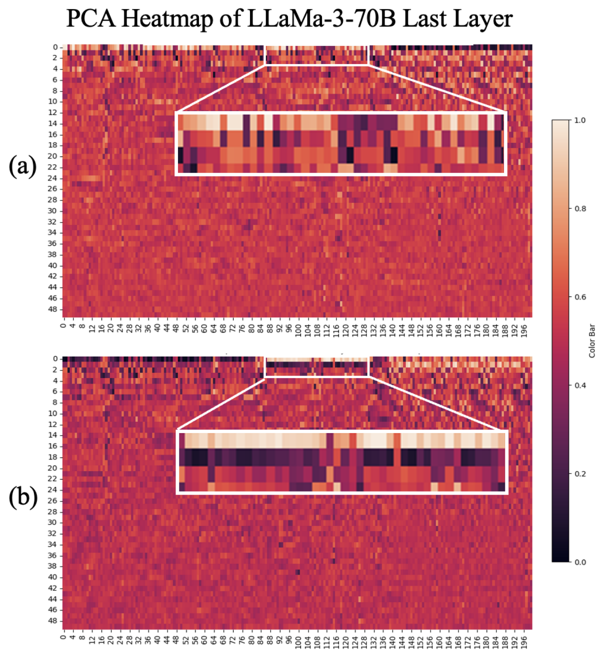


Figure 6: Heatmap of PCA-transformed final layer of LLaMA-3-70B with different prompts. (a) With original single-lingual Prompt. (b) Multi-lingual Prompts with the rule 8 in maindarin (Ours). The X-axis represents input tokens, and the Y-axis shows the 50 PCA components.

4.6 Attention Verification

To verify whether MLPrompt increases the attention of LLMs of error-prone rules, we visualize the attention map of the final layer of the open-source LLaMA-3-70B (Dubey et al., 2024) when inputting the English prompt and our MLPrompt with rule 8 being translated into Mandarin. The attention map is firstly downsampled from 8192 to 50 for better representation using PCA. Fig. 6 shows the downsampled attention map, and the corresponding error-prone rule 8 positioned between 84 to 128 has been zoomed in for clear representations. The first few principal components of our proposed prompts predominantly focus on the translated Rule 8 while the original English prompts do not exhibit a similar concentration on any specific rule. The attention map visualization from the final layer of the LLM illustrates the efficacy of our proposed MLPrompt, which effectively directs the LLM's attention towards error-prone rules by incorporating non-dominant languages.

4.7 Text-to-SQL

We further expand our MLPrompt in text-to-SQL tasks to validate the generalization ability. Text-to-SQL is more challenging than text-to-JSON due to the difficulty in detecting rule violations in SQL, which in turn complicates the identification of error-prone rules. However, it is straightforward to manually identify rule violations from the SQL results. To demonstrate the effectiveness of our approach, we present an example where MLPrompt outperforms other methods in text-to-SQL, and the rules for LLMs and the process by which these rules are derived are detailed in Appendix A.6.

The task is: "Find the first name and age of students who have a pet," using the pets_1 database from the Spider V1.0 dataset (Yu et al., 2018). We generate the prompt for GPT-4 by combining the SQL schema with predefined rules. Manual anal-

Prompt combination	Error Rate (%)
English	0.35
English w Repetitive Rule 4	0.50
Rule 4 in Mandarin (Ours)	0.10
Rule 4 in Japanese (Ours)	**0.00**
Rule 4 in Korean (Ours)	**0.00**

Table 4: Error rates for different prompt configurations in the text-to-SQL task, based on 20 runs of the given sample.

ysis reveals that GPT-4 frequently violates rule 4 in this task. To address it, we translate rule 4 into Korean, Japanese, and Mandarin respectively using an online translation. We also present ablation studies examining repetitive error-prone rules. The performance, evaluated by the error rate for SQL execution based on 20 runs of the provided example, is shown in Tab. 4, which shows that GPT-4 would fail to follow long and comprehensive rules. Even when the error-prone rules are repeated for emphasis, GPT-4 still fails. A potential strategy to mitigate this challenge is to split the rules within the prompt. However, isolating them is not feasible due to their interleaved nature, which could compromise the integrity of the prompt structure. Our proposed MLPrompt, utilizing various non-dominant languages, consistently exhibits low error rates, demonstrating the superiority of MLPrompt in text-to-SQL tasks.

5 Conclusion

In this work, we tackle the challenge of generating structured data using LLMs in real-world applications, where complex rules and natural language ambiguity often hinder the effectiveness of traditional methods. To address this, we introduce MLPrompt, a novel method that improves LLM reasoning to generate structured data by translating error-prone rules into another language, enhancing attention from LLM, and overall data quality. In comparison with state-of-the-art prompting strategies like CoT, ToT, and SC, MLPrompt demonstrates faster inference times and lower error rates. Additionally, we utilize MLPrompt to bridge the gap between LLM and autonomous industrial MIP generation, conducting extensive experiments on Text-to-MIP to prove MLPrompt's effectiveness. Finally, we explore the possibility of applying MLPrompt to structured data generation tasks, such as Text-to-SQL.

6 Limitations

Difficulty in Identifying Rule Violations in Natural Language Prompts: While MLPrompt can enhance the quality of generated data after localizing the rule LLM would fail to follow, the identification of omitting or violated rules remains a significant challenge. One limitation of our approach arises from the abstract nature of natural language, which often leads to rule inter-dependencies. Unlike programming languages, which can be translated into executable code to verify rule compliance, natural language is an abstract representation that cannot be executed directly. In structured data generation tasks like text-to-SQL (see Appendix A.6, we define rules for LLMs to follow and validate the generated SQL by comparing it to the expected output. However, when discrepancies occur, pinpointing the specific rule violation is difficult, often requiring manual analysis.

However, enforcing rules with mathematical constraints, such as checking the data values, is relatively straightforward. The mechanism or model that can pinpoint which rule is violated is crucial for the effectiveness of MLPrompt.

Which non-dominant language should we use: In our experiment, the dominant language for LLMs is English, and we implement MLPrompt by translating the error-prone rule in English to another non-dominant language, such as German, French, Mandarin, Thai, Japanese, and Korean. The performance of German and French is notably lower, while Mandarin, Thai, and Korean show better results. Since our method relies on the GPT-4 series, and OpenAI has not disclosed the language distribution in GPT-4's training data, we reference the language distribution from GPT-3 (Brown et al., 2020). According to this, English, German, and French are among the top three languages in the training dataset, while Mandarin, Thai, and Korean rank much lower, around the 20s.

We hypothesize that MLPrompt is most effective when the distribution difference between dominant and non-dominant languages in the training data is neither too large nor too small. However, this hypothesis remains unproven, and selecting the appropriate language combination in the MLPrompt generation remains a key challenge.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tobias Achterberg. 2019. What's new in gurobi 9.0. *Webinar Talk url: https://www. gurobi. com/wp-content/uploads/2019/12/Gurobi-90-Overview-Webinar-Slides-1. pdf*, 5(9):97–113.

Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. 2024. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. 2023. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*.

Vineet Goyal Anureet Saxena and Miguel A. Lejeune. 2010. Mip reformulations of the probabilistic set covering problem. *Mathematical Programming*.

David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. 2006. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Alper Atamtürk. 2003. On the facets of the mixed–integer knapsack polyhedron. *Mathematical Programming*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Egon Balas and Andrew Ho. 1980. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. *Combinatorial Optimization*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*.

Anjali Bhavan, Pankaj Chauhan, Rajiv Ratn Shah, et al. 2019. Bagged support vector machines for emotion recognition from speech. *Knowledge-Based Systems*.

Robert E Bixby, Mark Fenelon, Zonghao Gu, Edward Rothberg, and Roland Wunderling. Mip: Theory and practice—closing the gap. In *System Modelling and Optimization*, pages 19–49.

Christian Bliek1ú, Pierre Bonami, and Andrea Lodi. 2014. Solving mixed-integer quadratic programming problems with ibm-cplex: a progress report. In *Proceedings of the twenty-sixth RAMP symposium*, pages 16–17.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*, volume 33, pages 1877–1901. Curran Associates, Inc.

Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, et al. 2024. Spider2-v: How far are multimodal agents from automating data science and engineering workflows? *arXiv preprint arXiv:2407.10956*.

Gerard Cornuejols, Marshall L Fisher, and George L Nemhauser. 1977. Exceptional paper—location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management science*, 23(8):789–810.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

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

Zijie Geng, Xijun Li, Jie Wang, Xiao Li, Yongdong Zhang, and Feng Wu. 2023. A deep instance generative framework for milp solvers under limited data availability. In *NeurIPS*.

Ziao Guo, Yang Li, Chang Liu, Wenli Ouyang, and Junchi Yan. 2024. ACM-MILP: Adaptive constraint modification via grouping and selection for hardness-preserving MILP instance generation. In *ICML*.

Zhenqi He, Junjun He, Jin Ye, and Yiqing Shen. 2023. Artifact restoration in histology images with diffusion probabilistic models. In *Medical Image Computing and Computer Assisted Intervention – MICCAI*, pages 518–527.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NeurIPS*.

Hieu Hoang, Huda Khayrallah, and Marcin Junczys-Dowmunt. 2024. On-the-fly fusion of large language models and machine translation. In *Findings of the Association for Computational Linguistics: NAACL 2024*.

Selin Hulagu and Hilmi Berk Celikoglu. 2020. A mixed integer linear programming formulation for green vehicle routing problem: Case for shuttle services. In *Computer Aided Systems Theory–EUROCAST 2019: 17th International Conference, Las Palmas de Gran Canaria, Spain, February 17–22, 2019, Revised Selected Papers, Part II 17*, pages 153–160. Springer.

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

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Ahmet B. Keha, Ketan Khowala, and John W. Fowler. 2009. Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*.

Diederik P. Kingma and Max Welling. 2019.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The NarrativeQA Reading Comprehension Challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2023. Bloom: A 176b-parameter open-access multilingual language model.

Xijun Li, Fangzhou Zhu, Hui-Ling Zhen, Weilin Luo, Meng Lu, Yimin Huang, Zhenan Fan, Zirui Zhou, Yufei Kuang, Zhihai Wang, et al. 2024. Machine learning insides optverse ai solver: Design principles and applications. *arXiv preprint arXiv:2401.05960*.

Zhuoyan Li, Hangxiao Zhu, Zhuoran Lu, and Ming Yin. 2023. Synthetic data generation with large language models for text classification: Potential and limitations. In *EMNLP*.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

John Mendonca, Alon Lavie, and Isabel Trancoso. 2023. Towards multilingual automatic open-domain dialogue evaluation. In *Proceedings of the 24th Annual Meeting of the Special Interest Group on Discourse and Dialogue*.

Lekhnath Sharma Pathak, Mila Vulchanova, Poshak Pathak, and Ramesh Kumar Mishra. 2024. Trilingual parallel processing: Do the dominant languages grab all the attention? *Bilingualism: Language and Cognition*, page 1–18.

Divya Aggarwal Rimmi Anand and Vijay Kumar. 2017. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems*.

Tadeusz Sawik. 2011. *Scheduling in Supply Chains Using Mixed Integer Programming*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Guan Wang, Sijie Cheng, Xianyuan Zhan, Xiangang Li, Sen Song, and Yang Liu. 2024a. Openchat: Advancing open-source language models with mixed-quality data. In *ICLR*.

Teng Wang, Wing-Yin Yu, Ruifeng She, Wenhan Yang, Taijie Chen, and Jianping Zhang. 2024b. Leveraging large language models for solving rare mip challenges. *arXiv preprint arXiv:2409.04464*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *NeurIPS*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Russ J Vander Wiel and Nikolaos V Sahinidis. 1995. Heuristic bounds and test problem generation for the time-dependent traveling salesman problem. *Transportation Science*.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. 2024. Chain-of-experts: When LLMs meet complex operations research problems. In *ICLR*.

Linzi Xing, Xinglu Wang, Yuxi Feng, Zhenan Fan, Jing Xiong, Zhijiang Guo, Xiaojin Fu, Rindra Ramamonjison, Mahdi Mostajabdaveh, Xiongwei Han, et al. 2024. Towards human-aligned evaluation for linear programming word problems. In *LREC-COLING*, pages 16550–16556.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024b. Synthesizing text-to-sql data from weak and strong llms. In *ACL*.

Tianxing Yang, Huigen Ye, and Hua Xu. 2024c. Learning to generate scalable milp instances. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *ICLR*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *ACL*.

Jianping Zhang, Yizhan Huang, Weibin Wu, and Michael R Lyu. 2023a. Transferable adversarial attacks on vision transformers with token gradient regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16415–16424.

Jianping Zhang, Yizhan Huang, Zhuoer Xu, Weibin Wu, and Michael R Lyu. 2024. Improving the adversarial transferability of vision transformers with virtual dense connection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7133–7141.

Jianping Zhang, Yung-Chieh Huang, Weibin Wu, and Michael R Lyu. 2023b. Towards semantics-and domain-aware adversarial attacks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 536–544.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023c. Automatic chain of thought prompting in large language models. In *ICLR*.

A Appendix

A.1 An Example for ComplexOR Dataset

Binpacking modeling problem in the ComplexOR dataset is shown blew:

```
{
  "id": 3,
  "title": "Binpacking Problem",
  "description": "The bin packing
      problem involves assigning items
      of known weights to bins with
      uniform capacity. The objective is
       to minimize the total number of
      bins utilized while ensuring that
      all items are allocated and each
      bin's total weight does not exceed
       the bin capacity.",
  "category": ["Binpacking Problem"],
  "model": {
    "set": [
      {
        "name": "I",
        "description": "Set of items"
      }
    ],
    "parameter": [
      {
        "name": "s",
        "description": "weight of item ‘
            i‘",
        "domain": "{i <in> I}"
      },
      {
        "name": "c",
        "description": "Capacity of a
            bin"
      }
    ],
    "variable": [
      {
        "name": "y",
        "description": "Binary variable,
             1 if we use bin ‘j‘",
        "domain": "{j <in> I}",
        "type": "binary"
      },
      {
        "name": "x",
        "description": "Binary variable,
             1 if we assign item ‘i‘ to
            bin ‘j‘",
        "domain": "{i <in> I, j <in> I}",

        "type": "binary"
      }
    ],
    "objective": [
      {
        "name": "MinBins",
        "description": "Minimize the
            total number of used bins",
        "sense": "min",
        "function": "<sum>_{j <in> I} y_{
            j}"
      }
    ],
    "constraint": [
      {
        "name": "CapConstraint",
        "description": "The total weight
             of assigned items to a bin
            should not exceed the bin
            capacity",
        "domain": "{j <in> I}",
        "function": "<sum>_{i <in> I} s_{
            i} * x_{i,j} <= c * y_{j}"
      },
      {
        "name": "AssignConstraint",
        "description": "Every items
            should be assigned to a bin",
        "domain": "{i <in> I}",
        "function": "<sum>_{j <in> I} x_{
            i,j} = 1"
      }
    ]
  }
}
```

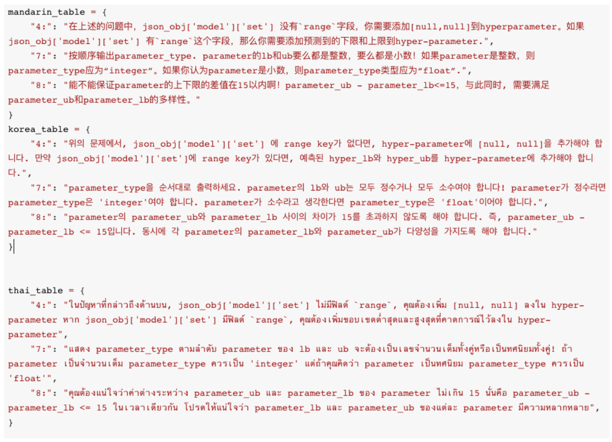
A.2 Rule for Text-to-MIP

The following rules are to be obeyed by the LLM, combined with modeling information (shown in Appendix A.1 from the ComplexOR dataset), to generate the prompt.

```
You must follow the following rules:

1: If the set in the model.json file has
    the 'range' key, this set has a
    hyper-parameter. For example, {range
    : [1,T]}, where T is the hyper-
    parameter. If the set in the model.
```

```
   json file doesn't have the 'range'
   key, the set doesn't have a hyper-
   parameter.
2: The number of set equals to the
   number of hyper-parameter.
3: In the set key, you need to append
   each set's lower bound and upper
   bound to the set value in order. The
    value of bound is generated by you
   after reading the model json file.
4: In the above case, if json_obj['model
   ']['set'] does not have the range
   field, you need to add [null, null]
   to the hyper-parameter. If json_obj
   ['model']['set']} has the range
   field, you need to add the predicted
    lower and upper bounds to the hyper
   -parameter.
5: The lower bound and upper bound of
   the set must be numbers, not null!
   The lb of set can start from non-one
    number.
6: When you provide the lower and upper
   bounds of the parameter, the value
   includes the lower bound but does
   not include the upper bound.
7: You should specify the data type of
   each parameter in order. The lb and
   ub of a parameter must either both
   be integers or both be float. If you
    think the type of this parameter is
    int, then you should append '
   integer'. If you think the type of
   this parameter is float, then you
   should append 'float'.
8: You should ensure that the gap
   between ub and lb of the parameter
   should be less than or equal to 15.
   ub-lb <= 15! At the same time,
   diversify the upper and lower bound
   of parameter.
9: You must follow the json data format
   {'set': [[lb1, ub1], [lb2,ub2]...],
   'hyper-parameter': [[lb1, ub1], [lb2
   ,ub2]...], 'parameter': [[lb1, ub1
   ],[lb2, ub2]...], 'parameter_types
   ':[integer, integer, float, ...]}.
```



```
mandarin_table = {
    "4:": "在上述的问题中，json_obj['model']['set'] 没有`range`字段，你需要添加[null,null]到hyperparameter。如果
json_obj['model']['set'] 有`range`这个字段，那么你需要添加预测到的下限和上限到hyper-parameter.",
    "7:": "按顺序输出parameter_type. parameter的lb和ub要么都是整数，要么都是小数！如果parameter是整数，则
parameter_type应为“integer”。如果你认为parameter是小数，则parameter_type类型应为“float”.",
    "8:": "能不能保证parameter的上下限的差值在15以内啊！parameter_ub - parameter_lb<=15，与此同时，需要满足
parameter_ub和parameter_lb的多样性。"
}
korea_table = {
    "4:": "위의 문제에서, json_obj['model']['set'] 에 range key가 없다면, hyper-parameter에 [null, null]을 추가해야 합
니다. 만약 json_obj['model']['set']에 range key가 있다면, 예측된 hyper_lb와 hyper_ub를 hyper-parameter에 추가해야 합니
다.",
    "7:": "parameter_type을 순서대로 출력하세요. parameter의 lb와 ub는 모두 정수거나 모두 소수여야 합니다! parameter가 정수라면
parameter_type은 'integer'여야 합니다. parameter가 소수라고 생각한다면 parameter_type은 'float'이어야 합니다.",
    "8:": "parameter의 parameter_ub와 parameter_lb 사이의 차이가 15를 초과하지 않도록 해야 합니다. 즉, parameter_ub -
parameter_lb <= 15입니다. 동시에 각 parameter의 parameter_lb와 parameter_ub가 다양성을 가지도록 해야 합니다."
}


thai_table = {
    "4:": "ในปัญหาที่กล่าวถึงด้านบน, json_obj['model']['set'] ไม่มีฟิลด์ `range`, คุณต้องเพิ่ม [null, null] ลงใน hyper-
parameter หาก json_obj['model']['set'] มีฟิลด์ `range`, คุณต้องเพิ่มขอบเขตต่ำสุดและสูงสุดที่คาดการณ์ไว้ลงใน hyper-
parameter",
    "7:": "แสดง parameter_type ตามลำดับ parameter ของ lb และ ub จะต้องเป็นเลขจำนวนเต็มทั้งคู่หรือเป็นทศนิยมทั้งคู่! ถ้า
parameter เป็นจำนวนเต็ม parameter_type ควรเป็น 'integer' แต่ถ้าคุณคิดว่า parameter เป็นทศนิยม parameter_type ควรเป็น
'float'",
    "8:": "คุณต้องแน่ใจว่าค่าต่างระหว่าง parameter_ub และ parameter_lb ของ parameter ไม่เกิน 15 นั่นคือ parameter_ub -
parameter_lb <= 15 ในเวลาเดียวกัน โปรดให้แน่ใจว่า parameter_lb และ parameter_ub ของแต่ละ parameter มีความหลากหลาย",
}
```

Figure 7: The figure shows the translations of rules 4, 7, and 8 into Mandarin, Korean, and Thai, illustrating the language-specific versions of these rules.

Since our experiment primarily focuses on generating data according to rules 4, 7, and 8, we also present the corresponding dictionary in Fig. 7, which illustrates how these rules are translated into Mandarin, Korean, and Thai.

A.3 Prompt Template for Text-to-MIP

The MLPrompt approach involves simply replacing a specific rule with its equivalent in another language. The dominant language in the prompt is English, and during application, only the target rule needs to be swapped with its counterpart in the desired language. The prompt template is as follows:

Prompt Template

You are required to return a feasible solution distribution under the given constraints.
Please read the following mixed integer programming (MIP) model and return a JSON object containing the lower and upper bounds for each set, hyper-parameter, and parameter.
Since the model does not include any data, your task is to provide the lower and upper bounds for every set, parameter, and hyper-parameter to construct instances for this model.
The required JSON format is as follows:
{'set': [[lb1, ub1], [lb2,ub2]...], 'hyper-parameter': [[lb1, ub1], [lb2,ub2]...], 'parameter': [[lb1, ub1],[lb2, ub2]...], 'parameter_types':[integer, integer, float, ...]}.
Rules is shown in Appendix A.2.

Modeling Information from ComplexOR Dataset. (An example is shown in Appendix A.1)

A.4 Analysis rules for Text to MIP

Rule 4 requires the LLM to read through the detailed modeling information, which consists of a long text, analyze the modeling process, and determine whether each set contains a corresponding hyper-parameter after reading the rule. Rule 4 evaluates the LLM's ability to comprehend the lengthy text and reason the result based on both the given question and the context.

Rule 7 requires consistency between the generated parameter values and their respective data types. Since the LLM must first output all parameter values and then generate their corresponding data types, it often forgets previous outputs, leading to struggles in maintaining coherence across the generated values. Rule 7 assesses the LLM's ability to ensure coherence throughout the process.

Rule 8 requires the LLM to diversify its output while adhering to specific mathematical constraints. The LLM often fails by generating common or repeated data values and not complying with the necessary constraints. Rule 8 evaluates the LLM's ability to generate diverse numbers, perform calculations, and adhere to given constraints.

A.5 Details of CoT, ToT, and SC

Due to the inherent ambiguity in natural language, evaluating intermediate steps during the reasoning process becomes problematic for methods like CoT, ToT, and SC. These methods rely on breaking down complex tasks into smaller steps and verifying them, which works well in structured environments like logic expressions. However, in structured data generation tasks, it is challenging to clearly define whether an intermediate result is accurate, as language is often subject to interpretation. Additionally, the interconnected nature of rules in structured outputs, such as JSON, makes it difficult to verify each step in isolation—correctness in one part does not necessarily guarantee overall correctness when the parts are combined. In our experiment, we made slight modifications to CoT, ToT, and SC to adapt them to our specific task. The details of these modifications are as follows:

CoT We implement CoT using zero-shot prompting by simply adding the phrase, "Let's think it step by step," to guide the LLM in reasoning through the task.

ToT We implement ToT by having the LLM generate each part of the desired JSON, similar to the structure of a tree, one by one. The previous output is combined into the prompt at each step until the entire requirement is fulfilled.

SC We implement SC by generating the full JSON multiple times, evaluating each result as either correct or incorrect. These labeled outputs are then combined into a new prompt, which is provided to the LLM to generate a more consistent and accurate final output.

A.6 Text to SQL

To demonstrate the generalization and robustness of MLPrompt, we apply our method to the Text-to-SQL task using the Spider V1.0 dataset (Yu et al., 2018). Given that each entity—such as companies, individuals, and datasets—follows its own coding style, we filter questions that GPT-4 answered incorrectly by having it generate SQL queries directly based on the SQL schema used in database construction, without any specific rules. We manually analyzed these incorrect cases and summarized a set of rules for GPT-4 to follow. Given the large size of the dataset, we used only 20% of it and identified the types of questions GPT-4 struggles to handle effectively. By combining these rules with the SQL schema, we employed zero-shot learning with GPT-4 to test whether it could generate the correct queries. The rules are as follows:

Rules for Text-to-SQL

1. When handling queries involving counting-related issues, avoid using "LEFT JOIN" or "RIGHT JOIN" to generate non-existent records. Instead, use "INNER JOIN".
2. Pay attention to the order of values requested in the question! Make sure the "SELECT" clause provides the appropriate field order. If the question does not ask for a count, do not include it. If the requirement is "xxx1 for each xxx2," the first field in "SELECT" should contain the data (xxx1) and the last field should return the category (xxx2). If the target is a primary key and the question asks how many xxx each primary key has, put the primary key last and the count first. There's no need to include

non-key fields like names.
3. If the logic is complex, use subqueries instead of joining tables. Avoid using "DISTINCT" unless necessary. When considering relationships between tables based on the query and the table's information, determine whether it's a "has-a" or "is-a" relationship. If the "SELECT" fields include a "has-a" scenario, use "DISTINCT" to avoid duplicates. If the query asks to list all xxx, do not use "DISTINCT".
4. If the user wants to find the maximum/minimum/average value in a table, consider using subqueries instead of grouping by different categories. If the query requires finding the maximum/minimum/average value within categories, use aggregation functions and "GROUP BY".
5. If the query requires listing all information, use "SELECT *".