Evaluation and Fine-Tuning of LLMs for Optimization Problem Transformation

Zhengxuan Wei
ShanghaiTech University
2022533077
weizhx2022@shanghaitech.edu.cn

Shouchen Zhou
ShanghaiTech University
2021533042
zhoushch@shanghaitech.edu.cn

Abstract

Transforming natural language descriptions of linear programming (LP) problems into standardized mathematical formulations is a critical yet challenging task. In this project, we leverage the NL4Opt dataset [1] to benchmark the capabilities of state-of-the-art large language models (LLMs) in this domain-specific transformation. We evaluate both proprietary and open-source models, highlighting common issues such as hallucinations and limited interpretability. To address these challenges, we fine-tune the Llama model[2, 3] using the NL4Opt dataset, demonstrating significant improvements in accuracy and reliability. Our results indicate that task-specific fine-tuning enhances the performance of open-source models, making them more suitable for practical applications. This work provides valuable insights into the strengths and limitations of current LLMs for optimization problem transformation and paves the way for developing more effective, domain-tailored language models.

1 Introduction

LLMs have demonstrated remarkable capabilities across a variety of natural language processing (NLP) tasks. However, their ability to address domain-specific challenges, such as transforming natural language descriptions of optimization problems into standard mathematical formulations, remains underexplored. Optimization problems are ubiquitous in science, engineering, and business, yet their natural language descriptions are often ambiguous and require expert knowledge to translate into formal representations. Automating this process has significant practical value, reducing manual effort and enabling broader accessibility to optimization methodologies.

The NL4Opt dataset has recently emerged as a benchmark for evaluating the performance of LLMs on optimization problem transformation tasks. Despite the growing interest in LLMs, existing studies have largely focused on general-purpose tasks, leaving a critical gap in understanding their performance on this specialized problem domain. Furthermore, current LLMs face challenges such as hallucination, lack of interpretability, and limited adaptability to task-specific requirements, making rigorous evaluation and fine-tuning essential for practical deployment.

In this work, we aim to evaluate the performance of state-of-the-art LLMs on the NL4Opt benchmark and identify the key challenges they face in transforming natural language descriptions into standard optimization formulations. Our study includes both proprietary models and open-source models. Additionally, we explore the potential of task-specific fine-tuning to improve the performance of open-source models, leveraging their flexibility for deployment and debugging in real-world scenarios.

By systematically analyzing the strengths and weaknesses of existing LLMs, our work seeks to advance the understanding of their capabilities in optimization problem transformation. We hope

SI151A Convex Optimization and Its Applications in Information Science Final Project.

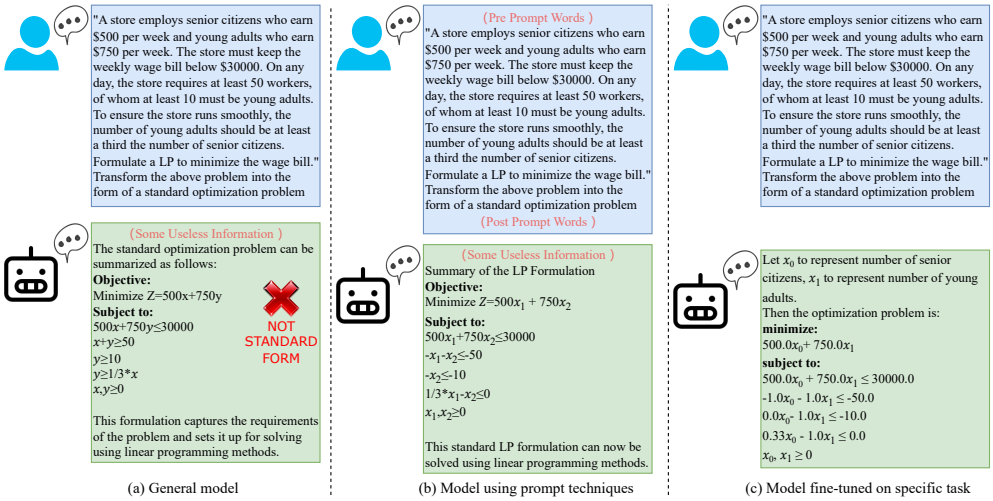

Figure 1: Overview of performance comparison among different models for this task. (a) Results from directly querying a general-purpose large language model. (b) Results from querying the large language model with added prompts. (c) Results after fine-tuning the model on this specific task.

our findings will provide valuable insights and foster further research into developing more effective and interpretable language models for solving domain-specific challenges.

2 Related Work

LLMs for optimization

Recent efforts have explored leveraging large language models (LLMs) to simplify combinatorial optimization, enabling non-experts to interact with optimization tools more effectively. The NL4Opt competition [3] emphasized translating natural language into optimization models, with Task 1 involving identification of objectives, variables, and constraints from text. Traditional Named Entity Recognition (NER) methods were used alongside fine-tuned LLMs like BERT and GPT on optimization-specific datasets [4]. Task 2 focused on constructing mathematical representations, where researchers applied sequence-to-sequence models to translate text into optimization formulations, often requiring separate models for extraction and formulation.

To address the limitations of multi-step approaches, Tsouro proposed a unified LLM-based framework [5] that directly generates optimization models from prompts, presenting promising results on NL4Opt datasets. Building on this, AhmadiTeshnizi introduced OptiMUS [6], a framework that uses GPT-based models to parse natural language into Mixed Integer Linear Programming (MILP) formulations. OptiMUS integrates end-to-end capabilities, including model generation, Gurobi solver integration, and debugging, and is benchmarked on the NLP4LP dataset, achieving significant improvements over baseline LLM approaches.

Yang presented OPRO [7], a prompt-based framework for iterative optimization without relying on traditional solvers. By generating multiple solutions and refining them through feedback, OPRO demonstrated encouraging preliminary results on datasets like GSM8K[8] and BBH [9]. However, its scalability to more complex problems remains under evaluation.

In applied contexts, Li developed OptiGuide [10], a system integrating LLMs with combinatorial optimization for decision-making in supply chain management. OptiGuide translates user queries into actionable optimization tasks, generating and validating solver code while providing user-friendly explanations. Deployed in real-world scenarios, such as Microsoft Azures supply chain, it achieved high accuracy (93% with GPT-4), demonstrating its practical effectiveness. These advancements collectively aim to democratize optimization tools, making them accessible to broader audiences.

3 Methods

3.1 Overview

In this section, we introduce our proposed methodology, which consists of three key components. First, we describe the specific techniques employed to preprocess and adapt the data to meet the requirements of our experiments. Second, we propose a prompting strategy designed to effectively enhance the accuracy of language models in solving optimization problems. Finally, we explain the principles and procedures behind fine-tuning the LLaMA model to address the downstream task better. This overview provides a comprehensive outline of our approach and sets the stage for the detailed explanations in the following sections. Figure 1 shows an example of this overview.

3.2 Data Pre-processing

The given data is combined with an instruction and a series of annotations. What we have done in this part is to transform the annotations into a canonical form, then a natural language-expressed optimization problem's standard form.

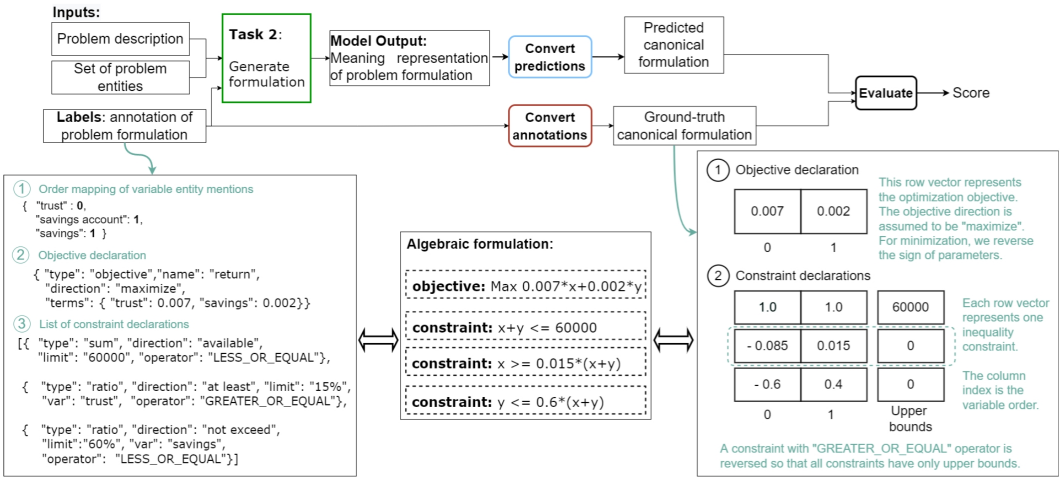

Figure 2: Details of how the ground-truth data transfer instructions into optimization problem's canonical form

3.2.1 Instruction to canonical form

The data pre-processing step uses the data from the competition NL4OPT [1], which details can be seen at https://github.com/nl4opt/nl4opt-competition. The data are annotated with extracted entities, objective function direction(i.e. a minimization or maximization problem), constraints from the instructions. The open-source code https://github.com/nl4opt/nl4opt-subtask2-baseline has already transformed these annotations into a canonical form of an LP problem. The pipeline for the annotation transformation can be seen in Figure 2.

The transformed canonical form is separated into the objective and constraint functions.

For the objective function, it could be written as $\mathbf{c}^\top \mathbf{x}$.

Constraints are typically formulated as a set of linear inequalities. These can be written in the general form:

$$A\mathbf{x} \leq \mathbf{b}$$

Where:

- $\mathbf{c}$ is the objective vector.
- A is an $m \times n$ matrix containing the coefficients a_{ij} of the decision variables.
- $\mathbf{x}$ is the n-dimensional vector of decision variables.

- $\mathbf{b}$ is the m-dimensional vector representing the right-hand side constants of the inequalities.

The canonical form consolidates all constraints into a single matrix equation:

$$[A \mid \mathbf{b}] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{bmatrix}$$

3.2.2 Canonical form to Nature Language

The canonical form is suitable for optimization, but not for human reading, so we additionally transform them into natural language.

As a brief overview. The canonical representation in our case describes the objectives and constraints as a vector and matrix respectively.

1. The annotated data also has information in the constraint called 'direction', which means that based on the annotated direction of the objective function, the optimization problem is categorized as either a minimization or a maximization task. The objective function is then articulated as a linear combination of the decision variables

2. Utilizing the annotated entities. The annotated data also has information in the constraint called 'entities', which extract the variables of the optimization problem from the instruction, and give each entity a unique id. Then each variable is systematically defined as x_i, where i ranges from 0 to n. This establishes a clear correspondence between the annotated entities and their respective decision variables in the LP formulation.

3. As mentioned in section 3.2.1, we could obtain the canonical form $\mathbf{c}$, and $[A \mid \mathbf{b}]$.

The constraints extracted from the annotations are expressed as a series of linear inequalities. Each constraint is formulated in the general form in natural language as follows:

Let x_0 to represent object_0, ..., x_0 to represent object_n. Then the optimization problem is:

$$\begin{aligned} \text{minimize / maximize} \quad & c_0 x_0 + c_1 x_1 + \cdots + c_n x_n \\ \text{s.t.} \quad & a_{i0} x_0 + a_{i1} x_1 + \ldots + a_{in} x_n \leq b_i \quad \text{for} \quad i = 1, 2, \ldots, m \end{aligned}$$

In conclusion, this methodological approach ensures a structured and accurate translation of natural language instructions into their corresponding LP formulations. By leveraging the detailed annotations provided by NL4OPT, the process maintains consistency and facilitates the subsequent manual conversion into specific optimization problems involving variables $x_0, x_1, \ldots, x_n$.

3.3 Prompt Techniques

We observe that directly querying general large language models with raw problems often results in errors during the transformation into standard optimization forms. While these models are capable of understanding the semantic meaning of the text, they frequently fail to address detailed aspects, particularly in converting problems into the specific standard format we require, even when explicitly instructed about the desired format.

To address these issues, we design an effective prompting strategy to significantly enhance the accuracy of model outputs. The prompting process consists of three stages. First, before posing the question, we explicitly state the requirements of the task, provide a concrete example, and emphasize the specific format expected in the output. Second, following the question, we instruct the model to adopt a chain-of-thought reasoning approach to derive the solution step-by-step. Finally, after the model provides its answer, we prompt it to verify its results for potential errors or issues in formatting, and we use the corrected response as the final output.

3.4 Fine-Tuning LLaMA

3.4.1 Fine-Tuning Objectives and Optimization Strategies

LLaMA (Large Language Model Meta AI) is a decoder-only Transformer model designed for autoregressive language modeling. Given a sequence of tokens $x = \{x_1, x_2, \ldots, x_T\}$, the model predicts the next token by modeling the conditional probability $p(x_t \mid x_{<t})$, where $x_{<t} = \{x_1, x_2, \ldots, x_{t-1}\}$ represents the context of the token x_t. The training objective maximizes the likelihood of the sequence x, which can be factorized as:

$$p(x) = \prod_{t=1}^{T} p(x_t \mid x_{<t}), \tag{1}$$

and equivalently minimizes the negative log-likelihood (NLL) loss:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}), \tag{2}$$

where θ represents the model parameters. The model consists of token embeddings, positional encodings, and a stack of Transformer decoder layers with self-attention and feedforward networks. Each Transformer block computes a contextualized representation $\mathbf{h}_t$ for each token, which is projected to logits over the vocabulary and normalized with a softmax function to produce $p_\theta(x_t \mid x_{<t})$.

For fine-tuning, we adapt the pre-trained parameters θ using a task-specific labeled dataset $\mathcal{D}$. Each sample in $\mathcal{D}$ consists of an input prompt x and a target sequence $y = \{y_1, y_2, \ldots, y_T\}$. The fine-tuning objective is:

$$\mathcal{L}(\theta) = -\sum_{(x,y)\in\mathcal{D}} \sum_{t=1}^{T} \log p_\theta(y_t \mid y_{<t}, x), \tag{3}$$

where x provides the task-specific context, and y is the desired output. Fine-tuning is performed using the AdamW optimizer with weight decay and a linear learning rate schedule with warm-up. Regularization techniques such as dropout and gradient clipping are applied to improve generalization and stabilize training.

After fine-tuning, the model generates text autoregressively, predicting one token at a time by sampling from $p_\theta(x_t \mid x_{<t})$ until an end-of-sequence token is generated or a predefined maximum sequence length is reached.

3.4.2 QLoRA: Quantized Low-Rank Adaptation

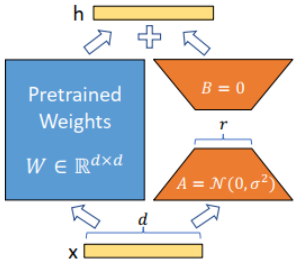

Figure 3: LoRA(Low-Rank Adaptation)

Research has demonstrated that pre-trained models possess an exceptionally low intrinsic dimension, indicating the existence of a highly reduced parameter subspace where fine-tuning achieves

performance comparable to that obtained by fine-tuning across the entire parameter space. Furthermore, empirical studies reveal that larger models exhibit even smaller intrinsic dimensions post-pre-training, thereby elucidating the superior few-shot learning capabilities inherent to large-scale models.

Inspired by the concept of intrinsic dimension, Low-Rank Adaptation (LoRA) operates under the premise that parameter updates during fine-tuning reside within an intrinsic rank. It could be understood as in Figure 3. Specifically, for a pre-trained weight matrix $W \in \mathbb{R}^{d\times k}$, the parameter update ΔW is represented through a low-rank decomposition:

$$W_0 + \Delta W = W_0 + BA$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$, with $r \ll \min(d, k)$. During the training process, the original pre-trained parameters W_0 are frozen, and only the parameters within matrices B and A are updated. Consequently, for an input x, the forward propagation is modified as follows:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

This approach significantly reduces the number of trainable parameters, enhancing both the efficiency and scalability of fine-tuning large language models. By leveraging low-rank matrices B and A, LoRA effectively captures the essential parameter updates within a compact subspace, aligning with the intrinsic dimensionality observed in pre-trained models. This methodology not only conserves computational resources but also preserves the integrity of the pre-trained weights, facilitating robust task-specific adaptations.

Building upon the Low-Rank Adaptation LoRA framework, QLoRA incorporates quantization techniques to further optimize memory efficiency during the fine-tuning of large language models. Specifically, QLoRA employs 4-bit quantization for the pre-trained weight matrices, significantly reducing the memory footprint required to store and process model parameters. By combining low-rank parameter updates with quantized weights, QLoRA enables the fine-tuning of substantially larger models on hardware with limited memory resources without compromising performance.

4 Experiments

4.1 Dataset and metirc

We use NL4OPT [1] as the dataset for our experiments. During the testing of the general large model, we use question 1 from the test set as a prompt example as Figure 1, and questions 2-11 as the ten test samples. The accuracy of the final ten questions is used as the evaluation metric. Considering the randomness of large language models, we repeat the test three times for each sample, and if more than half of the answers are correct, it is considered correct.

For the testing of LLaMA, we also use questions 2-11 from the test set as the ten test samples. When fine-tuning Llama, we train it using all the training set data.

4.2 Comparison of General LLMs

We tested several mainstream general large models, and the experimental results are shown in Table 1. Evidently, for all models, performance significantly improved after using prompts compared to their previous performance. Under the same settings, DeepSeek performed the best on this task, which we attribute to its use of chain-of-thought techniques, leading to a notable improvement in handling math-related problems. Other models that performed well include Gemini and Qwen. We can also see that GPT-4 significantly improved over GPT -3.5, highlighting the effectiveness of recent model iterations. It is worth noting that, due to testing costs, Claude was tested using the Haiku version of the model; other versions of Claude may have better performance.

4.3 Performance of Fine-tuned LLaMA

We have also tested the same questions, but without prompt on the open-source LLM: LLaMA3. Restricted by computing resources, we used the LLaMA3 with 8B parameters. And we fine-tuned the model with the QLoRA strategy mentioned in 3.4. The fine-tuning code is referencing https://github.com/taishan1994/Llama3.1-Finetuning. And the results could be seen in Table 2.

	Model	1	2	3	4	5	6	7	8	9	10	Acc
wo prompt	GPT-3.5	✓	×	×	×	×	×	×	×	×	✓	20%
	GPT-4	×	×	×	✓	✓	×	✓	✓	×	✓	50%
	Gemini	×	✓	✓	×	×	✓	✓	✓	×	✓	60%
	Claude	×	×	×	×	×	×	✓	✓	×	×	20%
	ERNIE	×	×	×	×	×	×	×	×	×	✓	10%
	Qwen	×	✓	✓	×	×	×	✓	✓	✓	✓	60%
	GLM	×	×	✓	×	×	×	×	×	✓	×	20%
	DeepSeek	×	✓	✓	✓	×	×	✓	✓	✓	✓	70%
after prompt	GPT-3.5	✓	×	✓	×	×	×	✓	×	×	✓	40%
	GPT-4	✓	×	✓	✓	✓	×	✓	✓	×	✓	70%
	Gemini	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	90%
	Claude	×	×	✓	×	✓	×	✓	✓	×	✓	50%
	ERNIE	✓	✓	×	×	×	✓	×	×	×	✓	40%
	Qwen	✓	✓	✓	×	✓	×	✓	✓	✓	✓	80%
	GLM	✓	✓	✓	×	×	×	×	×	✓	×	40%
	DeepSeek	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	90%

Table 1: Performance Comparison of General Large Language Models

Model	1	2	3	4	5	6	7	8	9	10	Acc
LLaMA	✓	×	×	✓	✓	✓	×	×	×	✓	40%
Finetuned LLaMA	✓	✓	×	✓	✓	✓	✓	✓	×	✓	80%

Table 2: Performance Comparison of LLaMA with and without finetuning

We put the outputs for the LLaMa with or without fine-tuning in the appendix folder. Surprisingly, the fine-tuned model could correctly determine the variables in the optimization problem and their representation in the instruction. It has quite similar results to ground truth. The format is exactly the same as ground truth, but it is slightly different in details. For example, the third question in the testing dataset has one requirement: 'The number of senior accountants should be at least a third of the number of junior accountants.' Let x_0 represent senior accountants, and x_1 represent junior accountants. The output of the fine-tuned model is $3.0x_0$ - $1.0x_1$ <= 0.0, while the ground-truth is that -$1.0x_0$ + $0.33x_1$ <= 0.0. Where they have different outputs not the same, but with similar correct meanings.

We can see that fine-tuning the LLMs could significantly improve the behaviors. The original LLaMa without prompt has 40% accuracy, which is moderate. But after fine-tuning, it reaches 80% accuracy, ranking first among our testing without prompting.

We are convinced that the LLMs have large potential, due to the computation limitation, we only fine-tuned a small model. If we enlarge the model, such as changing the 8B model into the 70B model, using more data for fine-tuning, with a bigger batch size and training time, after the prompt, it could almost certainly have much better results!

5 Conclusion

In conclusion, our evaluation of state-of-the-art LLMs on the NL4Opt benchmark reveals both promising results and significant challenges. While LLMs demonstrate a degree of success in transforming natural language descriptions of optimization problems into mathematical formulations, they still exhibit limitations such as hallucinations, lack of interpretability, and difficulty in adapting to specific task requirements. Our findings highlight the critical need for further research and development in this area, including the exploration of task-specific fine-tuning strategies to enhance the performance and reliability of LLMs for solving domain-specific challenges like optimization problem transformation.

References

[1] . Ramamonjison et al. Augmenting operations research with auto-formulation of optimization models from problem descriptions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 29–62, Abu Dhabi, UAE, December 2022. Association for Computational Linguistics.

[2] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[3] Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 189–203. PMLR, 28 Nov–09 Dec 2022.

[4] Parag Pravin Dakle, Serdar Kadolu, Karthik Uppuluri, Regina Politi, Preethi Raghavan, SaiKrishna Rallabandi, and Ravisutha Srinivasamurthy. Ner4opt: Named entity recognition for optimization modelling from natural language. 2023.

[5] Dimos Tsouros, Hélène Verhaeghe, Serdar Kadıoğlu, and Tias Guns. Holy grail 2.0: From natural language to constraint models. *arXiv preprint arXiv:2308.01589*, 2023.

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

[7] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[9] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

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