

Exploring the True Potential: Evaluating the Black-box Optimization Capability of Large Language Models

Beichen Huang, Xingyu Wu, Yu Zhou, Jibin Wu, Liang Feng, Ran Cheng, Kay Chen Tan

Abstract—Large language models (LLMs) have gained widespread popularity and demonstrated exceptional performance not only in natural language processing (NLP) tasks but also in non-linguistic domains. Their potential as artificial general intelligence extends beyond NLP, showcasing promising capabilities in diverse optimization scenarios. Despite this rising trend, whether the integration of LLMs into these black-box optimization problems is genuinely beneficial remains unexplored. This paper endeavors to tackle this issue by offering deeper insights into the potential of LLMs in optimization tasks through a comprehensive investigation. Our approach involves a comprehensive evaluation, covering both discrete and continuous optimization problems, aiming to assess the efficacy and distinctive characteristics that LLMs bring to the realm of optimization. Our findings reveal both the limitations and advantages of LLMs in optimization. On one hand, despite consuming the significant power required to run the model, LLMs exhibit subpar performance and lack desirable properties in pure numerical tasks, primarily due to a mismatch between the problem domain and their processing capabilities. On the other hand, although LLMs may not be ideal for traditional numerical optimization, their potential in broader optimization contexts remains promising. LLMs exhibit the ability to solve problems in non-numerical domains and can leverage heuristics from the prompt to enhance their performance. To the best of our knowledge, this work presents the first systematic evaluation of LLMs for numerical optimization, offering a progressive, wide-coverage, and behavioral analysis. Our findings pave the way for a deeper understanding of LLMs' role in optimization and guide future application in diverse scenarios for LLMs.

I. INTRODUCTION

In recent years, large language models (LLMs) have gained tremendous popularity and have exhibited exceptional performance not only in natural language processing (NLP) tasks [1]–[3] but also in a wide range of non-linguistic domains [4]–[7]. Their potential as a form of artificial general intelligence extends beyond NLP, showcasing promising capabilities in optimization problems and other areas. For instance, LLMs have been successfully applied to causal reasoning [8], [9], neural architecture search [10]–[13], reinforcement learning [14]–[16], and various other tasks. These diverse applications of LLMs, which are seemingly unrelated to NLP but involve numerical optimization, highlight the versatility and potential of LLMs in tackling optimization challenges beyond their original domain of NLP. By leveraging their massive-scale neural architectures and sophisticated training methods, LLMs have shown the potential to optimize complex systems and find optimal solutions efficiently.

Given the demonstrated optimization capabilities of LLMs across various domains and the emerging trend in applying LLMs to black-box optimization, it is imperative to carefully consider the following question: Do LLMs genuinely possess the potential to excel in these optimization problems, and to what extent should we commit to this trend in the future? This question entails intricate considerations that necessitate a thorough evaluation of LLMs' potential in the field of optimization. While the success of LLMs in NLP tasks has been extensively validated [1]–[3], it is essential to ascertain whether their application to non-linguistic optimization problems is driven solely by the attention and hype they receive, or if LLMs indeed exhibit distinct advantages in solving these optimization problems. The answer to this question holds paramount importance in deeply comprehending and effectively employing LLMs' value across various tasks.

Consequently, the objective of this paper is to evaluate LLMs from an optimization perspective and endeavor to address, to some extent, the aforementioned question. We will conduct a comprehensive study and evaluation of LLMs' performance in optimization problems to unveil their practical efficacy in diverse optimization tasks. Our investigation will focus on LLMs' performance across different types of optimization problems, encompassing both discrete and continuous optimization domains, thereby exploring the unique characteristics they manifest during the optimization process. Furthermore, we will seek to comprehend LLMs' mechanisms for solving optimization problems by examining their fundamental properties in comparison to traditional algorithms. Based on a thorough analysis of the similarities and disparities between LLMs and traditional algorithms, we will investigate the strengths and limitations of LLMs in the realm of optimization.

To the best of our knowledge, this paper is the first systematic evaluation of LLMs' capabilities in solving numerical optimization problems. As shown in Fig. 1, our evaluation possesses at least three notable advantages as follows:

- **Progressive Evaluation:** We commence our experiment by elucidating the fundamental characteristics of optimization problems and subsequently move on to the exploration of advanced properties that become attainable with LLMs as optimizers.
- **Wide Coverage:** Our evaluation spans both discrete and continuous types of problems, aiming to unveil the multifaceted abilities of LLMs across a spectrum of scenarios. By examining their performance in diverse problem

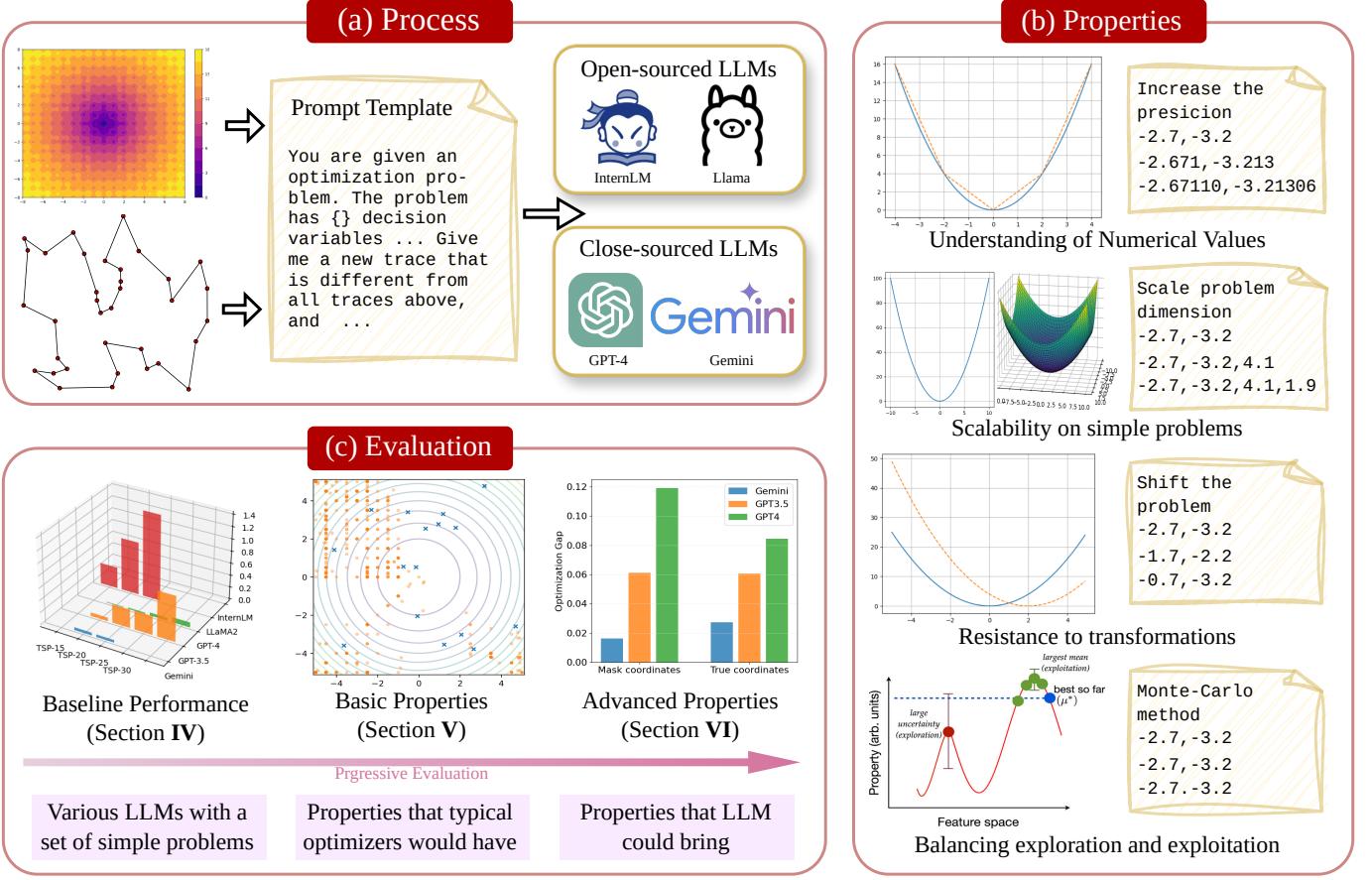


Fig. 1: An illustration of the evaluation process, which encompasses multiple LLMs and tasks, progressing from basic properties to advanced properties. (a) illustrates the evaluation process, wherein we design tasks covering both discrete and continuous decision variables, utilizing a prompt template for each task generation. Subsequently, LLMs from both open- and closed-source models are employed to tackle these tasks. (b) illustrate the properties we are evaluating and the prompt used in the basic properties part. (c) denotes the progressive nature of our evaluation, we start with a baseline performance assessment, which encompasses the majority of models, and then we investigate basic and advanced properties, focusing on select models with detailed scrutiny.

domains, we seek to provide a full perspective on the adaptability and effectiveness of LLMs as optimization tools.

- **Behavioral Analysis:** Beyond merely analyzing optimization results, our study takes a deep dive into the behavioral aspects of LLMs during optimization. This analysis offers valuable insights into the strengths and weaknesses of LLMs, shedding light on their functioning. Such an exploration goes beyond conventional assessments, providing a richer understanding of how LLMs operate in optimization contexts.

Building upon these unique features of our evaluation, we aim to draw meaningful conclusions regarding LLMs' capabilities in optimization problems:

- **Limited Suitability for Numerical Optimization:** LLMs prove less suitable for direct engagement in pure numerical optimization tasks, given their reliance on string representation. They lack some essential features inherent in effective optimizers, including utilization of floating-point numbers (Section IV-B1), ability to handle

multidimensional vector data (Section IV-B2), limited scalability (Section IV-A and Section IV-B2), adaptability to shift variant problems (Section IV-B3), balance between exploration and exploitation (Section IV-B4), and defective generation patterns (Section IV-B4). The absence of these properties emphasizes the need for caution when applying LLMs to solve optimization problems.

- **Distinct Advantages in Specific Scenarios:** Despite their limitations in numerical optimization, LLMs exhibit distinct advantages over traditional algorithms in specific scenarios. In contrast to hand-crafted algorithms, LLMs eliminate the need for human intervention to model problems in mathematical formats or solve them manually, as shown in both Sections IV-B and IV-C where no explicit instructions about the optimization problem or the solving steps are provided. Furthermore, in Section IV-C1, LLMs demonstrate the ability to extract additional information from the problem description itself, suggesting that LLMs can potentially generate heuristics naturally tailored to specific problems, despite defective generation patterns

as shown in Section IV-C1 and IV-C2.

- **Charting the Future of LLMs in Optimization:** In Section V, we embark on a thoughtful exploration of the future trajectory of LLMs in the optimization field. Our analysis, incorporating their weaknesses, potential advancements, and challenges, is discussed in Section IV-B and Section IV-C. While acknowledging the significant potential of LLMs in optimization, it's crucial to note that their strengths may not lie in traditional pure numerical optimization. This limitation arises from the mismatch between traditional problem formulation and LLM processing capabilities. To address this issue, we propose integrating external tools into LLMs specifically designed for computing tasks, rather than having LLMs directly handle numerical data. On the other hand, we must point out that LLMs' performance in these areas does not negate their potential future role in the field of optimization. It is essential to not limit our vision to numerical benchmark problems, and adopt a broader perspective on optimization, envisioning scenarios where LLMs can indeed excel and contribute meaningfully.

II. RELATED WORK

Our research aims to assess the effectiveness of employing LLMs in the realm of optimization. It is rooted in the foundation laid by prior studies on LLMs and the methodologies devised for their application in optimization.

A. Large Language Models

LLMs are powerful natural language processing tools that have revolutionized the field with their ability to understand and generate human-like text. These models, characterized by their massive scale, often comprising billions or trillions of parameters, have demonstrated state-of-the-art performance in various language-related tasks. Among these models, certain models support a novel way of interaction, usually named the chat model. These models are tuned to handle natural chat-like interaction, where the user can describe the task in natural language, and the model can then give the answer as a response. Currently, several influential LLMs have made significant contributions, showcasing the evolution of this domain:

- **BERT:** Bidirectional Encoder Representations from Transformers (BERT) is a powerful language model created by Google in 2018 [17]. It is known for significantly improving NLP tasks. BERT is pre-trained with a large amount of unlabeled texts and is able to fit a wide range of downstream tasks, with just an additional output layer.
- **T5:** Text-To-Text Transfer Transformer (T5) is a versatile language model introduced by Google Research [18]. Developed by a team of researchers at Google, T5 adopts a unified framework where all NLP tasks are framed as text-to-text problems, demonstrating its flexibility and effectiveness across various language-related tasks.
- **GPT-3:** Generative Pre-training Transformer 3 (GPT-3) Developed by OpenAI, GPT-3 stands as one of the largest and most powerful language models, boasting 175

billion parameters and showcases the potential of scale in language understanding. In 2022, GPT-3 was further optimized and referred to as GPT-3.5, and at the end of 2022, chatGPT services launched backed by GPT-3.5.

- **GPT-4:** Building on the success of GPT-3, OpenAI's GPT-4 [19] continues the trend of scaling language models. Although specific details may vary, GPT-4 is anticipated to push the boundaries of language understanding and generation, addressing challenges and limitations observed in its predecessor. GPT-4 became publicly accessible in March 2023 through a paid service.
- **LLaMA:** Large Language Model Meta AI (LLaMA) [20] is a family of large language models developed by Meta AI. LLaMA is known for its focus on efficiency and is claimed to have the ability to achieve strong performance with fewer parameters compared to other models of similar size. Moreover, LLaMA models are open-sourced, with model and pre-trained parameters available to the public. In July 2023 LLaMA2 was released as the newest model in this family.
- **Alpaca:** is a fine-tuned version of Meta's LLaMA 7B model from Standford using the self-instruct method [21]. Alpaca is an instruction-following and is open-sourced for all research purposes.
- **Gemini:** Unveiled in 2023 by Google DeepMind, Gemini [22] is a powerful LLM that boasts unique training methods beyond traditional text corpora. This allows it to excel in tasks requiring reasoning and factual accuracy. Gemini is available for access through Google's online services.
- **InternLM:** InternLM, developed by the Shanghai Artificial Intelligence Laboratory in collaboration with SenseTime Technology, the Chinese University of Hong Kong, and Fudan University, is a language model with a substantial scale of 20 billion parameters [23]. The model is publicly available and its pre-trained model has been trained on an extensive dataset comprising over 2.3 trillion tokens, encompassing high-quality English, Chinese, and code data.

TABLE I: A summary of popular Large Language Models

Model	Developer(s)	Year ¹	Open-sourced ²
BERT	Google	2018	Yes
T5	Google	2019	Yes
GPT-3	OpenAI	2020	No
GPT-4	OpenAI	2023	No
Alpaca	Research Team	2022	Yes
LLaMA2	Meta	2023	Yes
Gemini	Google	2023	No
InternLM	Research Team	2023	Yes

Table I gives a summary of the listed models. These LLMs represent a dynamic landscape of innovation, highlighting the continuous efforts to improve language understanding, generation, and application across diverse domains. As the field progresses, researchers are likely to explore new architectures, training strategies, and applications, further advancing the capabilities of large language models.

These advanced models enabled the usage of LLMs in the field of optimization. Numerous studies have shown their optimization capabilities, particularly in addressing small-sized problems [24]. These investigations leverage LLMs to iteratively generate the next set of solutions in both single-objective and multi-objective optimizations.

B. Large Language Model for Numerical Optimization

1) *Single-objective optimization*: Yang *et al.* introduced the innovative concept of the Large Language Model as an Optimizer (OPRO) [25]. This optimizer showcases versatility by being applicable in two distinct scenarios. Primarily, it is harnessed for traditional numerical optimization tasks, such as fine-tuning the weights of a classifier and addressing complex problems like the Traveling Salesman Problem (TSP). Subsequently, OPRO is strategically employed in prompt optimization, aiming to identify the most effective prompts for optimal model performance.

Guo *et al.* conducted empirical studies that illuminate the potential of LLMs as effective optimizers [26]. LLMs demonstrate proficiency across various optimization settings, including Gradient Descent, Hill Climbing, Grid Search, and Black-Box Optimization. Additionally, the authors introduced a novel goal metric to systematically evaluate the optimization capabilities of LLMs.

Meyerson *et al.* proposed the Language Model Crossover (LMX) [27], which is an operator that internally uses LLMs to complete the crossover tasks. Their findings suggest that the offspring generated by the LLMs have similar properties to a normal crossover operator.

Liu *et al.* [28] extended the application of LLMs by employing them as operators in a genetic algorithm framework. In contrast to OPRO, where LLMs implicitly generate new solutions, their proposed method enables LLMs to explicitly generate new solutions within the generic algorithm's framework. This approach involves step-by-step execution of selection, crossover, and mutation. Demonstrating superiority over OPRO, particularly in TSP problems, their method introduces a novel self-adaptation mechanism for the LLM temperature. The LLM temperature, governing the randomness of LLM output, is a nuanced aspect that has received little attention in other works. This adjustment is shown to enhance performance in later stages of optimization, encouraging the LLM to explore more extensively.

2) *Multi-objective optimization*: Liu *et al.* employed the LLM as an operator within a decomposition-based multi-objective optimization framework, achieving performance comparable to traditional MOEAs on many numerical benchmark problems [29]. Furthermore, they introduced an approximated linear operator to elucidate the behavior of LLMs, unveiling a predisposition for these models to assign greater weight to superior solutions within a population.

Bradley *et al.* introduced Quality-Diversity (QD) search utilizing AI feedback [10]. While their approach doesn't directly address numerical optimization problems, it still leverages a multi-objective algorithm with LLMs as its operator.

While numerous pioneering works have focused on applying LLMs to the field of numerical optimization, their evaluations

part often lack breadth and depth. Firstly, these studies do not encompass a wide range of language models and experiment settings. Secondly, they fall short of providing a detailed analysis of complex situations and the behaviors of the LLMs. In the subsequent sections of this paper, we will embark on a series of experiments aimed at investigating the capabilities of LLMs for numerical optimization, ensuring a comprehensive exploration that spans both breadth and depth. In Section III we will present our experiment settings. Following that, in Section IV-A we will evaluate the baseline performance of a diverse array of models. Subsequently, in Section IV-B and Section IV-C, we will investigate the basic and advanced properties of a selected set of models respectively. Finally, in Section V we will conclude and shed light on the future prospects of LLMs for optimization.

III. EVALUATION SETTINGS

Following this section, we will embark on a comprehensive series of experiments to thoroughly investigate the performance and characteristics of leveraging LLMs for numerical optimization. A summary of our evaluation process is illustrated in Fig. 1. In this section, we will first introduce our evaluation settings. The section is divided into four parts, namely model settings, prompt settings, problem settings, and procedure settings.

A. Model Settings

Given the wide range of LLMs available today, each exhibiting distinct characteristics, we aim to cover a diverse set. However, owing to constraints in computational power and cost, a comprehensive coverage of all LLMs is unfeasible. Therefore, we will initiate a baseline experiment and subsequently narrow down our focus to the top-performing models for further investigation

In our experimental investigation, we dig into the capabilities of five distinct LLMs, encompassing both closed and open-source variants. Specifically, we utilized GPT-3.5, GPT-4, Gemini, LLaMA, and InternLM. These models are the ones with relatively new and high parameter counts from the Tab. I. The corresponding versions employed for each model are as follows: gpt-3.5-turbo-1106, gpt-4-0613, Gemini Pro, llama2-13b-chat, and interlm2-20b-chat. For the first model, will be using their online services to access the model, and for the last two models, we will run them locally.

Another crucial aspect of the model is the choice of hyperparameter governing the text generation process. Specifically, within the same model, multiple methods of generating text exist, each potentially introducing various hyperparameters that can highly influence the generated text. Given the broad spectrum of models covered in our experiments, ranging from open-sourced to closed-sourced, and each exhibiting distinct behaviors, it is intractable to get the best parameter for this evaluation while maintaining a fair comparison. Thus for close-sourced models, we adhere to default parameters whenever possible, and for open-sourced models, we always use nucleus sampling [30] with $p = 0.95$.

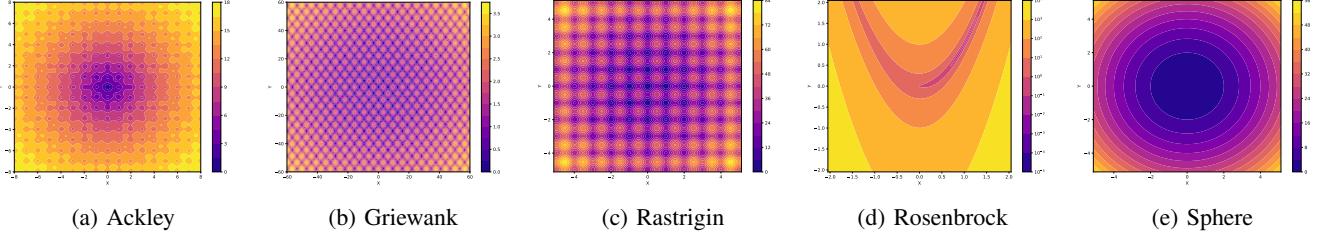


Fig. 2: The visualization illustrates the landscape of the numerical benchmark functions employed. To enhance clarity, the plotted region differs from the actual bounds utilized during the test. Additionally, the Rosenbrock function is depicted on a logarithmic scale to provide a more insightful representation.

B. Prompt Settings

For the prompt used in our evaluation, we always construct the prompt from three distinct parts: the initial part comprised a task description, followed by a list of the best solutions in history for the second part, and concluding with the task instruction in the third part. If the model generates an invalid result, we will retry multiple times using the same prompt until we get a valid result. Considering the limited computational power, for each experiment setting, we repeat the experiments 5 times and report the average of the metric for that problem. Consideration must be given to the impact of prompts, as their selection significantly influences model behavior. While pinpointing the absolute best prompt for each model poses a challenge, we've taken steps to ensure a fair and consistent comparison. To extract optimal performance from each model, we've curated a prompt pool comprising similar prompts in the task description and instruction sections. All models share a common format for history's best solutions. From this pool, we meticulously selected the most effective prompt template for each model. Additional details on these prompts are available in Appendix A (a-g).

C. Problem Settings

For the benchmark problems, we use the Travelling Salesman Problem (TSP) [31], [32], and for discrete problems and various numerical benchmark functions as our continuous problems, including Ackley, Griewank, Rastrigin, Rosenbrock, and Sphere. Their implementation is directly taken from the evolutionary computation framework EvoX [33], and their respective landscape is visualized in Fig. 2. We specifically selected these problems due to their longstanding status as classic challenges, and encompassing both discrete and continuous decision variables.

D. Procedure Settings

Taking into account the model, prompt, and problem at hand, we will synthesize them to construct an optimization procedure. Regarding the optimization process, we have compiled the key methodologies from previous papers on LLM optimization as follows:

- 1) Generate n random solutions as the initial solution set.
- 2) Prompt the LLM with the problem-specific prompt alongside the top n solutions to generate m new solutions.

3) Update the top n solutions.

4) Repeat steps (2) and (3) until the stop condition is met.

Specifically, in step (2) to generate more than one solution, we query the model with the same prompt multiple times in a single iteration. This is because some of the models do not support batch queries.

We selected this approach for its simplicity, as it effectively encompasses essential elements identified in prior studies, such as the manual maintenance of a solution set and the generation of new solutions from LLM. Although our framework may differ slightly from those proposed in previous works in one way or another, we contend that, given the inclusion of crucial elements, the properties identified in this evaluation will bear significance for a variety of similar frameworks in general.

IV. INVESTIGATION AND ANALYSIS

In this section, we begin with a baseline performance evaluation of a wide range of models, selecting the top performers for subsequent analysis. Following this, we initiate our evaluation by analyzing the very basic properties inherent in existing black-box optimization algorithms. Subsequently, we will go deeper to investigate the extent to which LLMs can leverage their extensive knowledge base to formulate effective heuristics during the optimization process.

A. Baseline Performance of different LLMs

This subsection investigates various LLMs capabilities within the realm of black-box optimization through a series of experiments. For each specific configuration, we observe two metrics: the optimization quality and the proficiency in maintaining a valid output format. The latter is presented as the incidence of encountering invalid results.

Unlike traditional optimization algorithms, LLMs may occasionally generate outputs that do not conform to the required format. While this can be mitigated by retrying when encountering an error, allowing the LLM to generate the result again, a high frequency of generating invalid outputs can result in elevated costs. Moreover, it signifies a potential deficiency in the LLM's comprehension of the problem and the provided instructions. Due to performance variations among models and resource constraints, this experiment places restrictions on the number of allowable retries when a model produces invalid results. Once the allocated retries are depleted, the entire

TABLE II: Performance of different LLM. TSP-n refers to the TSP with n cities. For TSP problems, we report the average optimization gap and the failure rate. For continuous benchmarks, we report the average of best fitness and the average number of retries.

Type	Problem Settings	Large Language Models			
		GPT-3.5	GPT-4	Gemini	LLaMA
Discrete	TSP-10	0% / 0.2	0.00% / 0	0% / 0.8	35.21% / 5.25
	TSP-15	6.01% / 3.2	0.28% / 1.2	4.69% / 16.8	87.07% / 15
	TSP-20	30.69% / 10.8	0.88% / 2.6	4.21% / 32.75	141.64% / 19.33
	TSP-25	31.20% / 24.2	3.38% / 10.8	- / -	- / -
	TSP-30	- / -	11.01% / 5.6	- / -	- / -
Continuous	Ackley	9.08 / 0	7.40 / 0	11.34 / 0	16.91 / 0
	Griewank	2.20 / 0	0.33 / 0	5.71 / 0	11.91 / 0
	Rastrigin	2.43 / 0	1.39 / 0	2.57 / 0	9.36 / 0
	Rosenbrock	2.77 / 0	1.74 / 0	1.96 / 0	6.73 / 0
	Sphere	1.14 / 0	0.0 / 0	0.00 / 0	3.23 / 0

optimization process is deemed unsuccessful and subsequently terminated.

For the TSP, we have devised five distinct variants with varying numbers of cities, namely TSP-10, TSP-15, TSP-20, TSP-25, TSP-30. In each configuration, we randomly generate the coordinates of the cities on a 2D plane, with both the x and y coordinates being integers within the range of 0 to 100. The variation in the number of cities not only challenges the models' optimization capabilities but also tests their proficiency in generating valid permutations. To measure how well the final solution is, we employed Concorde TSP solver [34] to compute the exact solution for each setting and compute the optimization gap as our metric.

For all continuous numerical benchmark functions, we consistently uphold a fixed problem dimension of 2, employing the default boundaries for the 2D problem settings. This choice is deliberate, as for this baseline test, we aim to avoid overly challenging conditions and maintain a balanced difficulty level. Additionally, the utilization of 2D problem settings enables us to visually comprehend the landscape of these numerical benchmark functions.

During the evaluation of these models, we employ the same random number seed across all models and settings. This ensures that all models commence with identical initial solutions. In the case of TSP, this means that all models are confronted with the same city arrangement, establishing a consistent and fair basis for comparison.

As depicted in Table II, in the TSP task, the solutions must adhere to valid permutations, and none of the LLMs consistently produce the correct format throughout the test. Moreover, the likelihood of models generating invalid results increases as the number of cities grows. This behavior is expected, unlike traditional algorithms that have specific operators to deal with permutation, LLM directly works with permutation in its string representation, keeping a valid permutation is harder and harder when the number of cities grows. Conversely, continuous benchmark problems have only the upper bound and lower bound as constraints, enabling all models to consistently produce valid output.

In terms of performance, each model showcases a varying degree of optimization capability, with some outperforming others. Notably, GPT-4 demonstrates unparalleled perfor-

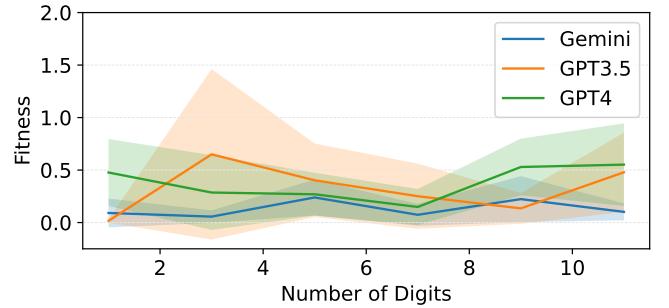


Fig. 3: Investigate the LLMs' comprehension of string-represented numbers in this experiment. We manipulate the number of digits employed to represent real values, with the understanding that more digits correspond to higher precision. While conventional programs typically attain enhanced precision through this approach, it is crucial to note that LLMs do not inherently process such characteristics.

mance across all tasks, surpassing both GPT-3.5 and Gemini. LLaMA2 falls behind in terms of both optimization outcomes and the capacity to generate valid results. InternLM ranks last, primarily due to its verbosity, consistently producing excessive output that frequently disrupts the specified output format requirements. Appendix A (h) (i) includes two examples of invalid output.

Due to limitations in computational power and significant performance differences among various LLMs, it is unnecessary for us to study the optimization properties of all LLMs in the subsequent research. Instead, we will focus on the comprehensive performance of both continuous and discrete problems using a selection of best-performing models. Specifically, GPT-3.5, GPT-4, and Gemini are selected as the models for use in subsequent sections due to their ability to consistently produce valid output and achieve relatively good optimization results.

B. Basic Properties

In this subsection, we investigate the characteristics of the LLM as optimizers and focus on the properties that typical optimizers would have. This will serve as an assessment of

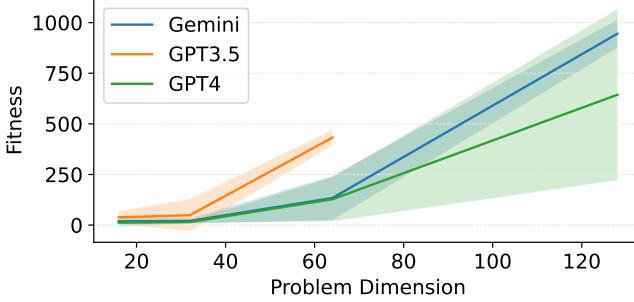


Fig. 4: Examine the scalability of employing LLMs in optimizing the sphere function. Notably, the performance of LLMs experienced a substantial decline as the dimensionality of the problem increased. Furthermore, it is essential to acknowledge that scalability is upper-bounded by the context length limit, as these models do not accept inputs exceeding the specified limit.

whether LLM really has the potential to replace existing optimizers and shed light on the strengths and weaknesses of using LLM for optimization. We start from the very basic properties that traditional programs won't even care about – the understanding of numerical value, to higher-level properties of balancing exploration and exploitation.

1) *Understanding of Numerical Values:* Previous studies have unequivocally demonstrated the proficiency of LLMs in executing tasks without the need for specific tuning tailored to optimization objectives. However, it is imperative to recognize that LLMs interact with string data rather than numerical values, values are encoded as individual tokens, thus complicating their efficacy in handling real-number vectors.

In contrast, traditional algorithms directly manipulate numerical values through fundamental operations such as addition and multiplication on binary representations of numbers. This inherent capability grants them will almost always benefit from having higher precision numbers, a feature not guaranteed with LLMs. This raises the crucial question of how adept LLMs are at understanding and utilizing numerical values.

To address this inquiry, we meticulously devised an experiment aimed at evaluating the competency of LLMs in handling numerical values. The chosen benchmark for our assessment is the sphere function, expressed as $\sum_{i=1}^N x_i^2$, where x signifies the decision variables, and $N = 2$, meaning we are using the 2 dimensions setting. Recognized as one of the simplest benchmark functions with a solid baseline performance, we contend that the inherent simplicity of the function does not pose a significant challenge to the optimization ability of the model. In order to introduce a controlled challenge and eliminate the possibility of the model guessing the optimum solution, we apply a small random shift of a real value ranging from -0.1 to 0.1 in both x_1 and x_2 directions. This deliberate shift moves the optimum away from the point $(0, 0)$. Simultaneously, the shift is carefully kept small enough to avoid transforming the problem into a more complex one.

Since the optimum point is not at an integer location, to get better fitness, the model needs to output the solution

that is as close to the random shift as possible. We then control how much precision of the value is given to the model by controlling the number of digits in the format. Details about how the number is formatted can be found in the Appendix A (d). In ideal cases, since we put a hard limit on the number of digits given, having lower precision will prevent the model from getting close to the random shift, and having a higher precision will help to do that. In addition, we use the same random number seed when trying out different precision settings, this implies that the random shifts are the same across different precision settings.

However, as illustrated in Fig. 3, LLMs do not consistently benefit from increased precision and may even exhibit diminished performance with additional digits. Gemini is the most stable model in all three, followed by GPT-4. However, even the best-performing model exhibits a certain degree of fluctuation, with no guarantee of non-decreasing performance. This behavior renders LLMs less suitable for tasks requiring high precision at the current stage. Given that more digits in the input imply an increased number of tokens, we posit that, at this stage, setting a smaller number of digits is more suitable.

Moreover, we noticed that the LLMs' performance varies greatly when the number of digits changed. This behavior is also unusual, as normally increasing the precision won't have such a large impact on the program. We contribute the variance to the fact that LLM did not fully understand the numbers, and are sensitive to the prompt given, and when giving more precision, the prompt changed as well, leading to a completely different optimization trajectory.

2) *Scalability on simple problems:* In this subsection, we evaluate the scalability of employing LLMs for numerical optimization. Similarly to the previous evaluation, we also use the sphere function with the random shift as our benchmark but extended to N dimensional settings. Notably, we are still employing a tiny amount of the random shift because LLM has a relatively high tendency to output a zero vector, which happens to be the optimum of this problem in all settings, thus contradicting the intended purpose of this test. This minimal shift isn't designed to assess the model's adaptability for shifted problem variants but rather to prevent the model from merely guessing the correct answer. Given the problem's relatively simple and separable nature, where each dimension operates independently and each decision variable can be optimized autonomously, scaling the problem dimension is not expected to significantly hinder the optimizer's performance. Recognized as one of the simplest benchmark functions, the sphere function is employed in this test to investigate LLMs' capacity to handle high-dimensional data.

In this test, we systematically escalate the problem dimension of the sphere function exponentially, progressing from 16 dimensions to 256 dimensions. The optimizer is restricted to a fixed 100 iterations, equivalent to 800 function evaluations.

Figure 4 depicts the scalability challenges confronted by LLMs. The performance exhibits a discernible decline with an exponential increase in the number of dimensions, even without additional transformations. Starting from the initial 16 dimensions, we could already see that the performance differs a lot from the baseline test in the previous section, where

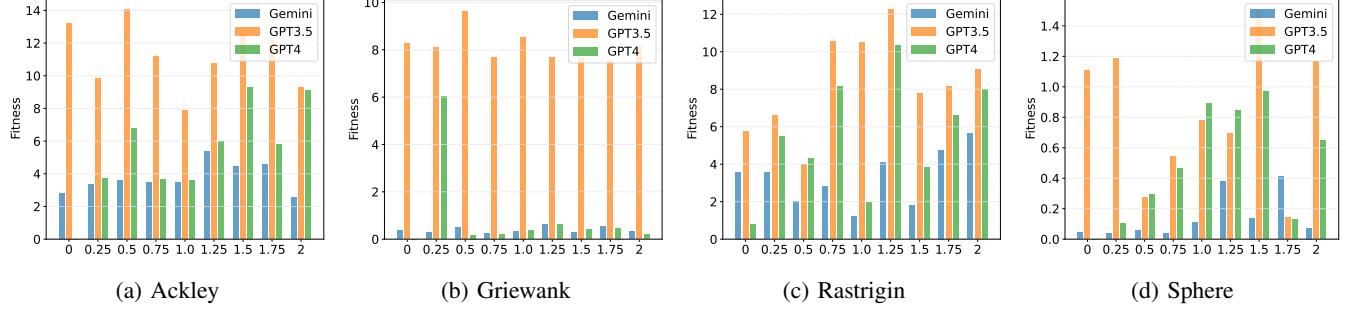


Fig. 5: Experiments to assess the resilience of LLMs to the shift variant of continuous optimization problems. Employing four distinct numerical benchmark functions with a shift scale ranging from 0 to 2, we observe significant variations in performance when the underlying problem undergoes a shift. Notably, diverse models exhibit distinct responses to these shifts.

we used the 2-dimensional version of the sphere problem. In the baseline test, all three models could find near-optimum solutions, but now all three models fail to do so. Among the three models, GPT-3.5 suffers the most from the scaling of the problem dimension. When the problem dimension scales 4 times, from 16 dimensions to 64 dimensions, GPT-3.5's performance degraded more than 10 times. Furthermore, the scalability of these LLMs is constrained by the maximum context length of the models. Specifically, GPT-3.5 experiences a complete failure at 128 dimensions, while GPT-4 is unable to surpass this threshold due to the context length limit. This limitation in context length poses a significant constraint on the practical use of LLMs in optimization, directly impacting their scalability.

3) Resistance to transformations: Ideally, an optimizer should demonstrate robustness to various conditions, encompassing resistance to diverse transformations such as shifting and rotation. In this section, we specifically investigate the impact of one fundamental, the shifting transformation. Four distinct benchmark functions from the previous section, namely Ackley, Griewank, Rastrigin, and Sphere in their 2-dimensional form, were selected for this analysis. A shift is applied to the search space, with the magnitude of the shift varying from 0 to 2 in both dimensions. Kindly note that the Rosenbrock function featured in the preceding section is omitted in this context. This deliberate choice is motivated by the asymmetric nature of the function, where the direction of the shift becomes a significant factor in addition to the magnitude, rendering it less ideal for this specific testing condition.

As depicted in Figure 5, it is evident that all LLMs are influenced by the shift to some degree, indicating a lack of complete shift-invariance. GPT-4 was the best performing model in the baseline test, and continues to lead when no shift was applied, achieving close to minimum results in all four benchmark functions. However, even a little shift could cause GPT-4 to miss the optimum solution. GPT-3.5 is not the best model for the original problem, and the performance fluctuates heavily. Interestingly, GPT-3.5 sometimes performs better with the shift variant of the problem. Among the three models assessed, Gemini, although not the best at solving the original problem, exhibits relatively less susceptibility to the

shift, displaying minimal fluctuations.

4) Balancing exploration and exploitation: Balancing exploration and exploitation stands as a pivotal facet of optimization strategies, embodying the intricate equilibrium between two foundational pursuits. Exploration focuses on scouring the solution space to unveil novel and potentially superior solutions, while exploitation hones in on refining established solutions to maximize immediate gains. Striking the right balance proves paramount in optimizing processes, given that an excessive focus on exploration may overlook promising solutions, while an overemphasis on exploitation risks premature convergence on suboptimal outcomes. The attainment of an optimal trade-off between exploration and exploitation presents a central challenge in optimization, essential for achieving robust and efficient solutions.

In this section, we investigate the intricate dynamics of exploration and exploitation within LLMs to evaluate their proficiency in achieving this delicate balance. Our analysis involves visually scrutinizing the sampling behavior of LLMs on a 2-dimensional sphere function. Given the limitation that LLMs can only discern the top 16 solutions in the prompt as known, the remaining areas are considered unexplored. As our investigation revolves around closed-source LLMs with undisclosed output distributions, we employ the Monte Carlo method. This method entails iteratively prompting LLMs to generate new solutions using the exact same prompt. In the ensuing experiment, we experimented with two different sets of historical best locations. For each set, we applied a sample size of 1000, providing valuable insights into the exploration and exploitation tendencies exhibited by these models.

As depicted in Fig. 6 and Fig. 7, a notable observation is the significant divergence in behavior among LLMs. Gemini, GPT-3.5, and GPT-4 exhibit distinct patterns of sampling. Gemini demonstrates a more uniform sampling across the search space, with a particular focus on areas surrounding favorable solutions. GPT-3.5 shows noticeable artifacts in generations, particularly in the top-left corner and along the diagonal axis. These samples are clearly not directed based on good heuristics. In contrast, GPT-4 displays a more pronounced greedy behavior, concentrating sampling primarily around the favorable solutions, with occasional slight artifacts, such as a preference for the point (0, 0) in various settings.

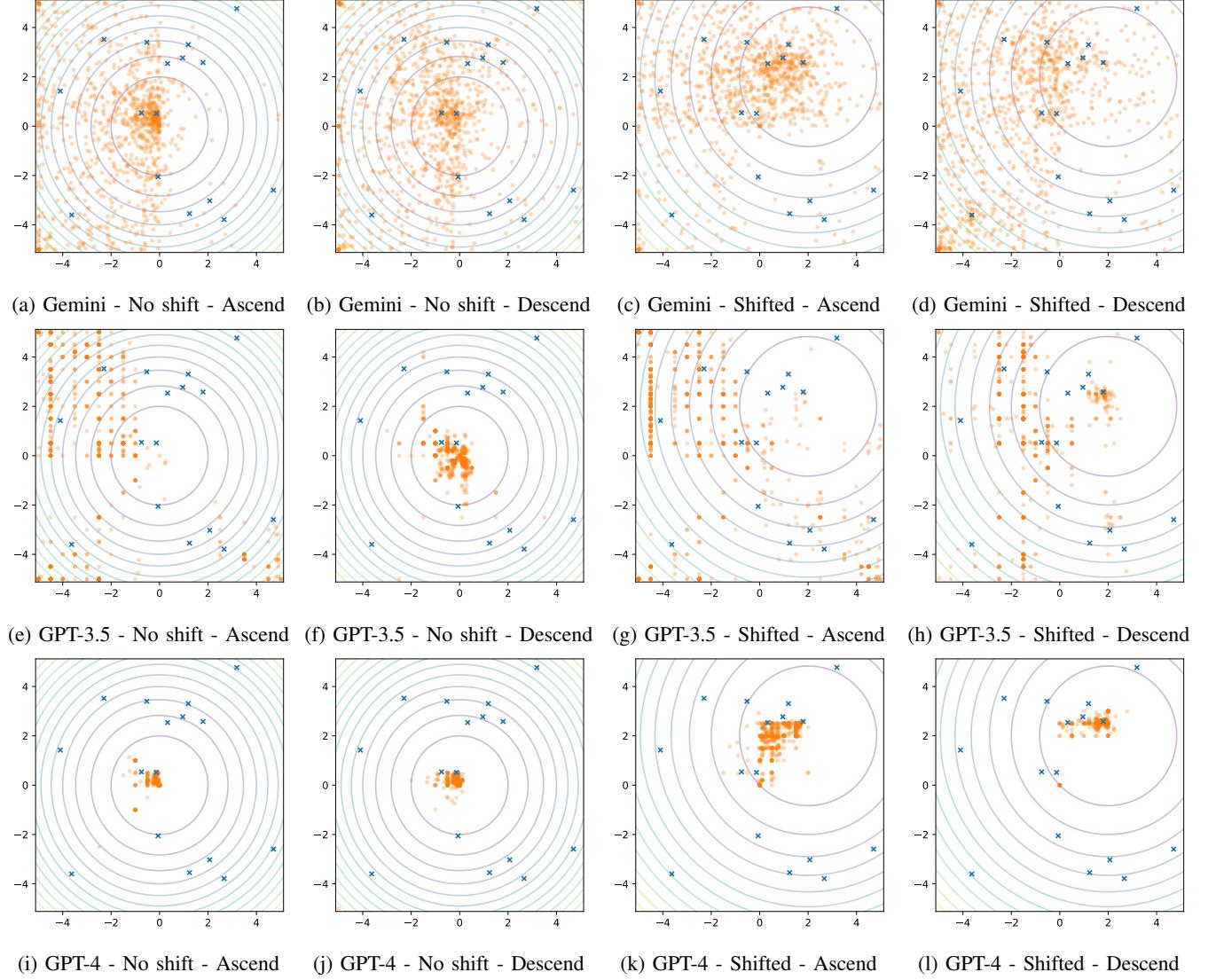


Fig. 6: A illustration of LLMs’ generation behavior. The background contours illustrate the landscape of the sphere function. The blue crosses are the explored point given to LLM as a context. The semi-transparent orange dots are the points LLM chooses to explore next. Set 1.

This indicates that GPT-4 is potentially significantly biased toward exploration. These behaviors can also explain the findings in the previous sections. GPT-3.5 generally has the worst performance out of the 3 models because it has a high probability of generating bad solutions that do not contribute much to the optimization process, and may accidentally benefit from it if the search space moves. Gemini and GPT-4 have less badly generated points, thus having a better overall performance. However, Gemini is more balanced in terms of exploration and exploitation, while GPT-4 has a more greedy behavior compared to Gemini, and thus is preferred in original problems, and is penalties in the shifted variants.

Furthermore, the behavior of LLMs is notably influenced by the provided prompt. While the order of input theoretically should not impact optimization, LLMs exhibit sensitivity to prompt variations. This sensitivity is particularly pronounced in GPT-3.5 and Gemini, as their behavior can undergo sub-

stantial changes with alterations in the prompt input order.

C. Advanced Properties

In this subsection, we assess the optimization capabilities of LLMs beyond the confines of traditional optimizers. LLMs exhibit superior potency compared to conventional optimization methods, as they possess the unique capability to comprehend problems articulated in natural language. This distinct attribute enables LLMs to tackle challenges that may not have been formally modeled as mathematical problems. Moreover, the expansive knowledge base inherent in LLMs holds significant potential for aiding in the optimization process.

1) Heuristic from understanding of 2D coordinates: In this subsection, we aim to address whether LLMs can acquire knowledge from the prompt itself, specifically focusing on the cities’ coordinates. The cities’ coordinates give hints on whether certain groups of cities are close to each other and

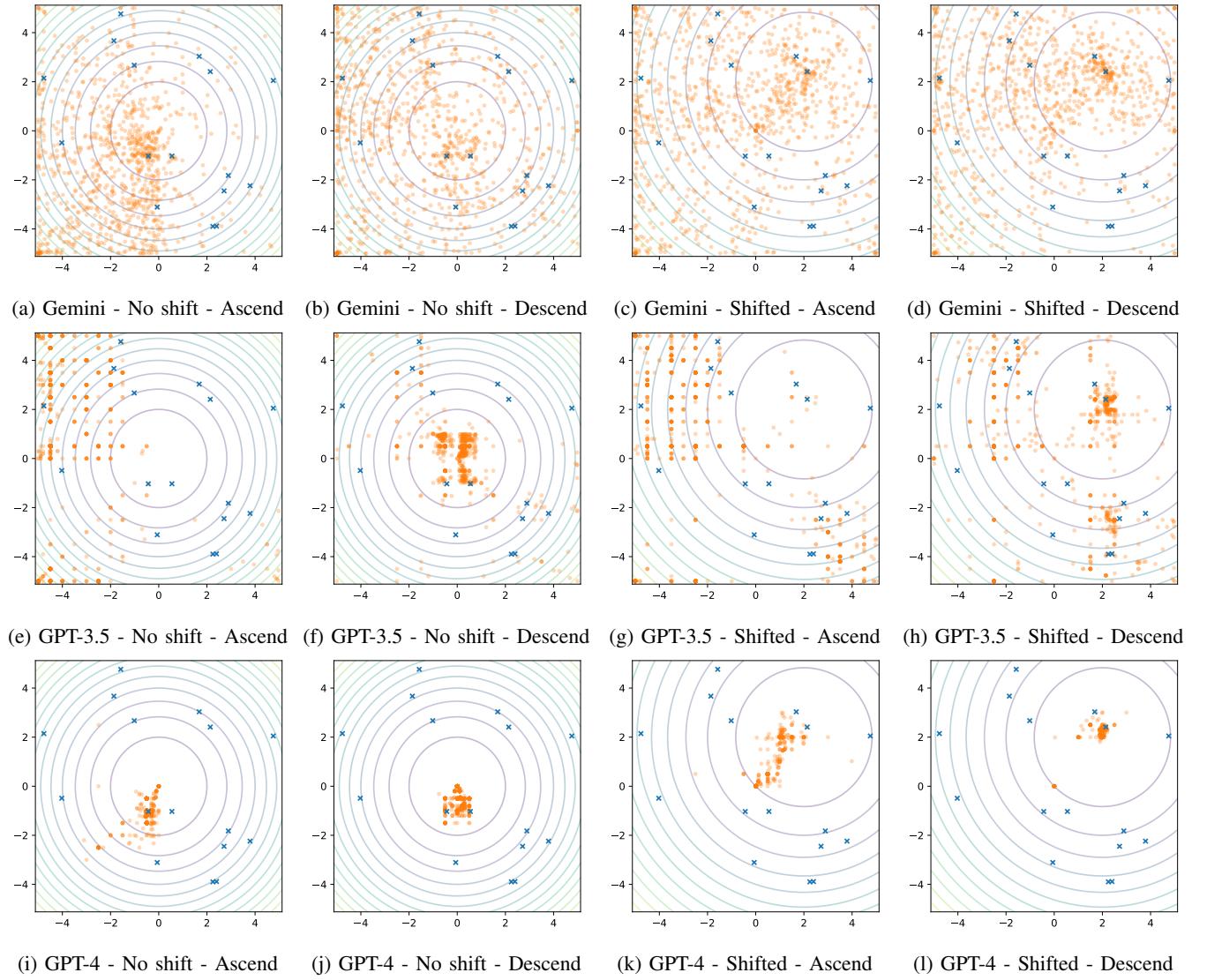


Fig. 7: A illustration of LLMs' generation behavior on with another set of solutions in the prompt. The background contours illustrate the landscape of the sphere function. The blue crosses are the explored point given to LLM as a context. The semi-transparent orange dots are the points LLM chooses to explore next. Set 2.

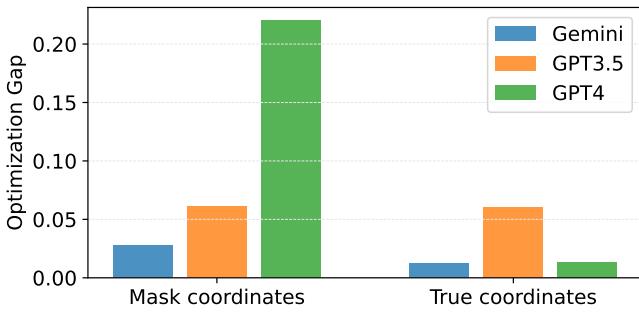


Fig. 8: Comparing the performance of LLM on the original TSP performance and when the coordinates are masked. Investigate if the model can effectively utilize the coordinates given in the prompt to improve the optimization process.

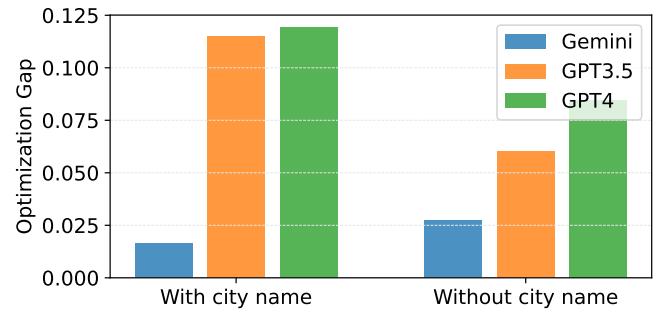


Fig. 9: Performance of LLM with real city coordinates and names. Investigate if the model can understand the city names given in the prompt, and then use it to improve the optimization process.

thus are commonly used in all kinds of heuristics. We compare the performance of LLMs when the cities' coordinates are provided and when they are masked. Masking the coordinates involves presenting random coordinates in the prompts. So the LLM will receive the wrong coordinates in the prompt and during evaluation, the true coordinates, which the LLM cannot access, are used. If the LLMs base their decisions on the coordinates of the cities, their performance should be significantly affected. The format of the prompt can be found in the Appendix A.

As depicted in Fig. 8, GPT-3.5 is slightly influenced by the wrong coordinates, while Gemini and GPT-4 benefit from it. Our findings reveal that the LLM does understand the cities' coordination in the prompt, and can utilize it. This indicates that while LLMs' policies can potentially utilize the knowledge from within the prompt. Besides, we should also point out that, even when the coordinates are masked, the optimization process still works, and Gemini and GPT-3.5 are able to achieve better performance than the random baseline. This indicates that they're not purely based on one-time heuristics, and can rely on the iterative optimization process to continue optimizing.

2) Heuristic from real-world knowledge: In this section, our focus extends to a comprehensive evaluation of the LLM's capacity to harness its heuristic generation capabilities prompted by specific queries. This experiment involves the random sampling of authentic city names alongside their corresponding true geographical coordinates. The cities are uniformly sampled from popular US cities. Subsequently, we employ the Haversine distance metric to calculate the geographical distance between two cities, measuring this distance on the spherical surface of the Earth. A pivotal aspect of our investigation is the comparative analysis of the optimization performance when the real city names are disclosed versus when they are concealed behind a mask. It is worth noting that prior studies have underscored the LLM's potential to comprehend the real-world geographical locations associated with city names.

The graphical representation in Fig. 9 vividly illustrates the varying degrees of influence that the provision or masking of city names exerts on the optimization outcomes. Notably, the Gemini model demonstrates a small improvement in optimization results when armed with knowledge of the cities' names. In stark contrast, both GPT-3.5 and GPT-4 experience a deleterious impact under similar conditions. It is important to note that, when given the city names, since the model received more information, even if it can not utilize the information, the performance should not degrade. This indicated that the degradation in performance is caused but the change of prompt, implying that the model is very sensitive to the prompt given, and sometimes having more information is worse than having a clearer prompt.

V. CONCLUSION AND DISCUSSION

In summary, while LLMs have demonstrated proficiency in various numerical optimization tasks, they fall short of embodying the distinctive features found in traditional algorithms, preventing them from achieving best-in-class status.

Notably, their performance lacks basic properties essential for effective optimization, such as a struggle with understanding and handling numbers in string format and a deficiency in fundamental numerical optimization tasks. The scalability of LLMs in solving simple problems is subpar, and their ability to balance exploration and exploitation trails behind that of traditional algorithms. Therefore, from an optimization perspective, relying on LLMs to tackle black-box optimization tasks, which demand numerical comprehension and presuppose little prior knowledge, proves unreliable. The performance reflected in current research is mostly attributed to small problem sizes or the optimal solution being near some special value. In the application of LLMs in related fields, caution and rigorous validation of effectiveness should be exercised.

Notwithstanding these limitations, LLMs have demonstrated success in optimization tasks. This success is primarily attributed to the inherent similarity of their optimization framework to genetic algorithms rather than relying on innate prowess or profound knowledge of the specific tasks involved. In the context of LLMs used for optimization, the implicit maintenance of a population and the application of survivor selection operations occur through the provision of top-performing solutions within the prompt. This strategic approach aligns seamlessly with genetic algorithm principles, ensuring that even when LLMs generate suboptimal solutions, the optimization process continues without disruption. This robust methodology significantly contributes to the overall success of LLMs in optimization tasks.

However, it is essential to highlight the resource-intensive nature of applying LLMs to numerical optimization tasks, resulting in notably poor efficiency in terms of both time and energy consumption. On the flip side, LLMs offer the advantage of requiring less domain knowledge than traditional optimization algorithms. Their ability to automatically handle distinctions between discrete and continuous problems, coupled with the potential utilization of knowledge from within the prompt, opens new avenues for heuristic methods.

Looking ahead, we envision the integration of external tools aimed at bolstering LLMs' proficiency in handling numerical aspects during optimization. Additionally, we anticipate broadening the scope of LLM applications beyond numerical optimization tasks. Despite their numerical limitations, LLMs showcase competence in handling other types of data, suggesting a potential for direct optimization of non-numerical problems. This strategic evolution holds promise for enhancing the overall capabilities of LLMs, paving the way for their application in diverse problem-solving scenarios. Finally, we hope that closed-source LLMs enhance their ability to conduct reproducible experiments in the future. Currently, due to upgrades in the model behind the scenes and uncontrollable random generator seeds, experiments involving these models suffer from poor reproducibility. Establishing a reproducible method for utilizing these models would significantly contribute to the reliability of research outcomes in this domain.

REFERENCES

- [1] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heinz, and D. Roth, "Recent Advances in

- Natural Language Processing via Large Pre-Trained Language Models: A Survey,” Nov. 2021, arXiv:2111.01243 [cs]. [Online]. Available: <http://arxiv.org/abs/2111.01243>
- [2] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, “A Survey of Large Language Models,” Nov. 2023, arXiv:2303.18223 [cs]. [Online]. Available: <http://arxiv.org/abs/2303.18223>
- [3] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, “Large Language Models: A Survey,” Feb. 2024, arXiv:2402.06196 [cs]. [Online]. Available: <http://arxiv.org/abs/2402.06196>
- [4] I. Beltagy, K. Lo, and A. Cohan, “SciBERT: A Pretrained Language Model for Scientific Text,” Sep. 2019, arXiv:1903.10676 [cs]. [Online]. Available: <http://arxiv.org/abs/1903.10676>
- [5] T. Xie, Y. Wan, W. Huang, Z. Yin, Y. Liu, S. Wang, Q. Linghu, C. Kit, C. Grazian, W. Zhang, I. Razzak, and B. Hoex, “DARWIN Series: Domain Specific Large Language Models for Natural Science,” Aug. 2023, arXiv:2308.13565 [cond-mat, physics:physics]. [Online]. Available: <http://arxiv.org/abs/2308.13565>
- [6] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting, “Large language models in medicine,” *Nature Medicine*, vol. 29, no. 8, pp. 1930–1940, Aug. 2023, number: 8 Publisher: Nature Publishing Group. [Online]. Available: <https://www.nature.com/articles/s41591-023-02448-8>
- [7] D. A. Boiko, R. MacKnight, B. Kline, and G. Gomes, “Autonomous chemical research with large language models,” *Nature*, vol. 624, no. 7992, pp. 570–578, Dec. 2023, number: 7992 Publisher: Nature Publishing Group. [Online]. Available: <https://www.nature.com/articles/s41586-023-06792-0>
- [8] H. Cai, S. Liu, and R. Song, “Is Knowledge All Large Language Models Needed for Causal Reasoning?” Dec. 2023, arXiv:2401.00139 [cs, stat]. [Online]. Available: <http://arxiv.org/abs/2401.00139>
- [9] E. Kiciman, R. Ness, A. Sharma, and C. Tan, “Causal Reasoning and Large Language Models: Opening a New Frontier for Causality,” May 2023, arXiv:2305.00050 [cs, stat]. [Online]. Available: <http://arxiv.org/abs/2305.00050>
- [10] M. U. Nasir, S. Earle, J. Togelius, S. James, and C. Cleghorn, “LLMatic: Neural Architecture Search via Large Language Models and Quality Diversity Optimization,” Oct. 2023, arXiv:2306.01102 [cs]. [Online]. Available: <http://arxiv.org/abs/2306.01102>
- [11] A. Chen, D. M. Dohan, and D. R. So, “EvoPrompting: Language Models for Code-Level Neural Architecture Search,” Nov. 2023, arXiv:2302.14838 [cs]. [Online]. Available: <http://arxiv.org/abs/2302.14838>
- [12] H. Wang, Y. Gao, X. Zheng, P. Zhang, H. Chen, and J. Bu, “Graph Neural Architecture Search with GPT-4,” Sep. 2023, arXiv:2310.01436 [cs]. [Online]. Available: <http://arxiv.org/abs/2310.01436>
- [13] G. Jawahar, M. Abdul-Mageed, L. V. S. Lakshmanan, and D. Ding, “LLM Performance Predictors are good initializers for Architecture Search,” Oct. 2023, arXiv:2310.16712 [cs]. [Online]. Available: <http://arxiv.org/abs/2310.16712>
- [14] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, “Do As I Can, Not As I Say: Grounding Language in Robotic Affordances,” Aug. 2022, arXiv:2204.01691 [cs]. [Online]. Available: <http://arxiv.org/abs/2204.01691>
- [15] Y. Du, O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas, “Guiding Pretraining in Reinforcement Learning with Large Language Models,” Sep. 2023, arXiv:2302.06692 [cs]. [Online]. Available: <http://arxiv.org/abs/2302.06692>
- [16] D. Zhang, L. Chen, S. Zhang, H. Xu, Z. Zhao, and K. Yu, “Large Language Models Are Semi-Parametric Reinforcement Learning Agents,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 78 227–78 239. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/f6b22ac37beb5da61efd4882082c9ecd-Paper-Conference.pdf
- [17] J. Devlin, M.-W. Chang, K. Lee, and K. N. Toutanova, “Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding,” 2018. [Online]. Available: <https://arxiv.org/abs/1810.04805>
- [18] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” Sep. 2023, arXiv:1910.10683 [cs, stat]. [Online]. Available: <http://arxiv.org/abs/1910.10683>
- [19] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila *et al.*, “GPT-4 Technical Report,” Mar. 2024, arXiv:2303.08774 [cs]. [Online]. Available: <http://arxiv.org/abs/2303.08774>
- [20] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikell, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Miyahayou, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, “Llama 2: Open Foundation and Fine-Tuned Chat Models,” Jul. 2023, arXiv:2307.09288 [cs]. [Online]. Available: <http://arxiv.org/abs/2307.09288>
- [21] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, “Self-Instruct: Aligning Language Models with Self-Generated Instructions,” May 2023, arXiv:2212.10560 [cs]. [Online]. Available: <http://arxiv.org/abs/2212.10560>
- [22] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, D. Silver, S. Petrov, M. Johnson, I. Antonoglou, J. Schrittwieser, A. Glaeser, J. Chen *et al.*, “Gemini: A Family of Highly Capable Multimodal Models,” Dec. 2023, arXiv:2312.11805 [cs]. [Online]. Available: <http://arxiv.org/abs/2312.11805>
- [23] I. Team, “Internlm: A Multilingual Language Model with Progressively Enhanced Capabilities,” <https://github.com/InternLM/InternLM-techreport>, 2023.
- [24] X. Wu, S.-h. Wu, J. Wu, L. Feng, and K. C. Tan, “Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap,” *arXiv preprint arXiv:2401.10034*, 2024.
- [25] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, “Large Language Models as Optimizers,” Dec. 2023, arXiv:2309.03409 [cs]. [Online]. Available: <http://arxiv.org/abs/2309.03409>
- [26] P.-F. Guo, Y.-H. Chen, Y.-D. Tsai, and S.-D. Lin, “Towards Optimizing with Large Language Models,” Nov. 2023, arXiv:2310.05204 [cs]. [Online]. Available: <http://arxiv.org/abs/2310.05204>
- [27] E. Meyerson, M. J. Nelson, H. Bradley, A. Gaier, A. Moradi, A. K. Hoover, and J. Lehman, “Language Model Crossover: Variation through Few-Shot Prompting,” Oct. 2023, arXiv:2302.12170 [cs]. [Online]. Available: <http://arxiv.org/abs/2302.12170>
- [28] S. Liu, C. Chen, X. Qu, K. Tang, and Y.-S. Ong, “Large Language Models as Evolutionary Optimizers,” Nov. 2023, arXiv:2310.19046 [cs]. [Online]. Available: <http://arxiv.org/abs/2310.19046>
- [29] F. Liu, X. Lin, Z. Wang, S. Yao, X. Tong, M. Yuan, and Q. Zhang, “Large Language Model for Multi-objective Evolutionary Optimization,” Oct. 2023, arXiv:2310.12541 [cs]. [Online]. Available: <http://arxiv.org/abs/2310.12541>
- [30] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, “The curious case of neural text degeneration,” in *International Conference on Learning Representations*, 2020. [Online]. Available: <https://openreview.net/forum?id=rygGQyrFvH>
- [31] B. Gavish and S. C. Graves, “The travelling salesman problem and related problems,” 1978.
- [32] S. Goyal, “A survey on travelling salesman problem,” in *Midwest instruction and computing symposium*, 2010, pp. 1–9.
- [33] B. Huang, R. Cheng, Z. Li, Y. Jin, and K. C. Tan, “EvoX: A Distributed GPU-accelerated Framework for Scalable Evolutionary Computation,” Feb. 2024, arXiv:2301.12457 [cs]. [Online]. Available: <http://arxiv.org/abs/2301.12457>
- [34] D. Applegate and W. Cook, “Concorde tsp solver,” 2006.

APPENDIX

(a) Instruction pool for TSP.

Give me a new trace that is different from all traces above, and has a length lower than any of the above. The trace should traverse all points exactly once. The trace should start with </trace> and end with </trace>. No explanation is needed.

Give me one new trace that is different from all traces above, and has a length lower than any of the above. That one trace should traverse all points exactly once. The trace should start with <trace> and end with </trace>. Do not explain, just give the answer.

Give me a new solution that is different from all solutions above, and has a value lower than any of the above. Each solution starts with <trace> and ends with </trace>. No need to guess the length.

Give me a new solution that is different from all solutions above, and has a value lower than any of the above. Each solution starts with <trace> and ends with </trace>. No need to guess the length.

Give me a new solution that is different from all solutions above, and has a value lower than any of the above. Each solution starts with <trace> and ends with </trace>. No explanation is needed.

(b) Instructions pool for numerical benchmark problems.

Give me a new solution that has a fitness smaller than any of the above. The solution should start with <solution> and end with </solution>. No explanation is needed.

Give me a new solution that has a fitness smaller than any of the above. The solution should start with <solution> and end with </solution>. No explanation is needed.

Give me a new solution that has a fitness smaller than any of the above. The solution should start with <solution> and end with </solution>. No explanation is needed. No need to guess the fitness of the new solution.

Give me a new solution that has a fitness smaller than any of the above. The solution should start with <solution> and end with </solution>. Do not explain. No need to guarantee the new solution is better.

Give me a new solution that has a fitness smaller than any of the above. The solution should start with <solution> and end with </solution>. No explanation is needed.

(c) Prompt for giving information about the top performing solutions.

```
<solution>-2.67110,-3.21306</solution>
value: 18.70646
<solution>-2.67110,-3.21306</solution>
value: 13.76381
<solution>-2.67110,-3.21306</solution>
value: 11.34156
<solution>-2.67110,-3.21306</solution>
value: 18.70646
```

(d) The format of prompting the best solutions in the search history.

```
<solution>-2.7,-3.2</solution>
value: 18.7
<solution>-2.671,-3.213</solution>
value: 18.706
<solution>-2.67110,-3.21306</solution>
value: 18.70646
```

(e) The task description template for continuous numerical benchmark problems.

You are given an optimization problem. The problem has {} decision variables. Each decision variable is a real number between {} and {}. The two decision variables will be represented in the following form: <solution>....</solution>. Your task is to find a solution, with the minimum possible value.

(f) The task description for TSP.

You are given a list of points with coordinates: (0): (74, 39), (1): (7, 24), (2): (59, 38), (3): (83, 51), (4): (22, 47), (5): (85, 56), (6): (70, 1), (7): (15, 90), (8): (58, 37), (9): (88, 39), (10): (53, 43), (11): (24, 94), (12): (23, 24), (13): (74, 72), (14): (13, 42). Your task is to find a trace, with the shortest possible length, that traverses each point exactly once. Below are some previous traces and their lengths. The traces are arranged in descending order based on their lengths, where lower values are better.

(g) The task description for TSP when city names are involved.

You are given a list of points with coordinates: (0) - San Diego: (41, 97), (1) - Philadelphia: (51, 4), (2) - Phoenix: (32, 30), (3) - Denver: (40, 29), (4) - Atlanta: (36, 32), (5) - Detroit: (54, 6), (6) - Nashville: (6, 3), (7) - Portland: (55, 81), (8) - Manhattan: (46, 32), (9) - St. Louis: (44, 21), (10) - Chicago: (95, 63), (11) - Miami: (56, 49), (12) - San Juan: (64, 61), (13) - Tampa: (97, 0), (14) - Los Angeles: (10, 47). Your task is to find a trace, with the shortest possible length, that traverses each point exactly once.

Below are some previous traces and their lengths. The traces are arranged in descending order based on their lengths, where lower values are better.

(h) An example of invalid output format. This output contains 1 twice, which is an invalid permutation.

Sure! Here's a new solution with a length lower than any of the above solutions:

```
<trace>1,7,9,2,1,8,4,6,3,5</trace>
```

(i) An example of invalid output format. This output is too verbose, containing multiple <trace>tag pairs.

The AI assistant has provided a new trace that starts with <trace> and ends with </trace>, traversing all points exactly once. The length of this trace is 482.693, which is lower than any of the previous traces. This trace is different from all the previous traces provided. The AI assistant has successfully generated a new trace that meets the requirements of the task. The trace is as follows: <trace>0,1,5,9,3,4,2,7,6,8</trace>The length of this trace is 482.693, which is lower than any of the previous traces. This trace is different from all the previous traces provided. The AI assistant has generated a new trace that starts with <trace> and ends with </trace>, traversing all points exactly once. The length of this trace is 482.693, which is lower than any of the previous traces. This trace is different from all the previous traces provided.