meta-instructions as an informal regularization of the generated solutions, such as "the instruction should be concise and generally applicable".

Optimization trajectory. Besides understanding natural language instructions, LLMs are also shown to be able to recognize patterns from in-context demonstrations (Wei et al., 2023; Madaan & Yazdanbakhsh, 2022; Mirchandani et al., 2023). Our meta-prompt makes use of this property and instructs the LLM to leverage the optimization trajectory for generating new solutions. Specifically, the optimization trajectory includes past solutions and their optimization scores, sorted in the ascending order. Including optimization trajectory in the meta-prompt allows the LLM to identify similarities of solutions with high scores, encouraging the LLM to build upon existing good solutions to construct potentially better ones without the need of explicitly defining how the solution should be updated.

2.3 Solution Generation

At the solution generation step, the LLM generates new solutions with the meta-prompt as input. The following are the key optimization challenges we address in this stage.

Optimization stability. In the optimization process, not all solutions achieve high scores and monotonically improve over prior ones. Due to the sensitivity of in-context learning to the prompt, LLM output can be drastically affected by low-quality solutions in the input optimization trajectory, especially at the beginning when the solution space has not been adequately explored. This sometimes results in optimization instability and large variance. To improve stability, we prompt the LLM to generate multiple solutions at each optimization step, allowing the LLM to simultaneously explore multiple possibilities and quickly discover promising directions to move forward.

Exploration-exploitation trade-off. We tune the LLM sampling temperature to balance between exploration and exploitation. A lower temperature encourages the LLM to exploit the solution space around the previously found solutions and make small adaptations, while a high temperature allows the LLM to more aggressively explore solutions that can be notably different.

3 Motivating Example: Mathematical Optimization

We first demonstrate the potential of LLMs in serving as optimizers for mathematical optimization. In particular, we present a case study on linear regression as an example of continuous optimization, and on the Traveling Salesman Problem (TSP) as an example of discrete optimization. On both tasks, we see LLMs properly capture the optimization directions on small-scale problems merely based on the past optimization trajectory provided in the meta-prompt.

3.1 Linear Regression

In linear regression problems, the goal is to find the linear coefficients that probabilistically best explain the response from the input variables. We study the setting in which the independent and dependent variables X and y are both one-dimensional and an intercept b is present, so that there are two one-dimensional variables w, b to optimize over. In a synthetic setting, we sample ground truth values for one-dimensional variables w_{true} and b_{true}, and generate 50 data points by $y = w_{\text{true}}x + b_{\text{true}} + \epsilon$, in which x ranges from 1 to 50 and ϵ is the standard Gaussian noise. Our optimization starts from 5 randomly sampled (w, b) pairs. In each step, we prompt an instruction-tuned LLM with a meta-prompt that includes the best 20 (w, b) pairs in history and their sorted objective values. The meta-prompt then asks for a new (w, b) pair that further decreases the objective value. A sample meta-prompt is shown in Figure 19 of Appendix C.1. We prompt the meta-prompt 8 times to generate at most 8 new (w, b) pairs in each step to improve optimization stability. Then we evaluate the objective value of the proposed pair and add it to history. We do black-box optimization: the analytic form does not appear in the meta-prompt text. This is because the LLM can often calculate the solution directly from the analytic form.

Table 2 summarizes the results with one of the following optimizer LLMs: `text-bison`, `gpt-3.5-turbo`, and `gpt-4`. We study three settings of w_{true} and b_{true}: within the starting region $[10, 20] \times [10, 20]$, "near outside" (each of w_{true} and b_{true} is outside the starting region but the distance is less than 10), and "far outside" (each of w_{true} and b_{true} is outside the starting region and the distance is greater than 10). We see:

Table 2: Linear regression by optimizer LLMs: the mean $\pm$ standard deviation of the number of steps and the number of unique (w, b) pairs explored before reaching the global optima. Both w and b start from 5 random starting points in $[10, 20]$. We use temperature 1.0 for all models. We run each setting 5 times. The starting points are the same across optimizer LLMs but are different across 5 runs, and are grouped by: within the starting region, outside and close to the starting region, and outside and farther from the starting region. Bold numbers indicate the best among three LLMs in each setting.

w_{true}	b_{true}	number of steps			number of unique (w, b) pairs explored		
		`text-bison`	`gpt-3.5-turbo`	`gpt-4`	`text-bison`	`gpt-3.5-turbo`	`gpt-4`
15	14	5.8 $\pm$ 2.6	7.6 $\pm$ 4.5	**4.0** $\pm$ 1.5	40.0 $\pm$ 12.4	36.0 $\pm$ 15.2	**17.2** $\pm$ 5.1
17	17	**4.0** $\pm$ 1.8	12.6 $\pm$ 6.0	6.0 $\pm$ 3.7	33.4 $\pm$ 11.7	53.8 $\pm$ 16.9	**26.0** $\pm$ 10.6
16	10	**3.8** $\pm$ 2.2	10.4 $\pm$ 5.4	6.2 $\pm$ 3.1	30.2 $\pm$ 13.4	42.8 $\pm$ 16.3	**24.2** $\pm$ 8.2
3	5	**9.8** $\pm$ 2.8	10.8 $\pm$ 2.7	12.2 $\pm$ 2.0	55.8 $\pm$ 16.1	39.6 $\pm$ 10.1	**33.0** $\pm$ 4.0
25	23	19.6 $\pm$ 11.4	26.4 $\pm$ 18.3	**12.2** $\pm$ 3.7	104.0 $\pm$ 52.3	78.6 $\pm$ 26.2	**44.2** $\pm$ 8.3
2	30	**31.4** $\pm$ 6.3	42.8 $\pm$ 9.7	38.0 $\pm$ 15.9	126.4 $\pm$ 17.7	125.6 $\pm$ 21.7	**99.0** $\pm$ 24.6
36	-1	**35.8** $\pm$ 6.4	45.4 $\pm$ 16.9	50.4 $\pm$ 18.8	174.0 $\pm$ 28.2	142.2 $\pm$ 31.2	**116.4** $\pm$ 32.7

- The number of unique (w, b) pairs explored by each model is fewer than exhaustive search, indicating these models are able to to do black-box optimization: compare the numbers and propose a descent direction.
- The `text-bison` and `gpt-4` models outperform `gpt-3.5-turbo` in convergence speed: they arrive at the optima with fewer steps. The `gpt-4` model also outperforms in finding the optima with fewer explored unique points. Taking a closer look at the optimization trajectory, we see `gpt-4` is the best at proposing a reasonable next step from the history: for example, when the history shows the objective values of $(w, b) = (8, 7)$, $(w, b) = (8, 6)$, and $(w, b) = (8, 5)$ are decreasing, it has a highest chance to propose $(w, b) = (8, 4)$ for evaluation.
- The problem becomes harder for all models when the ground truth moves farther from the starting region: all models need more explorations and more steps.

3.2 Traveling Salesman Problem (TSP)

Next, we consider the Traveling Salesman Problem (TSP) (Jünger et al., 1995; Gutin & Punnen, 2006), a classical combinatorial optimization problem with numerous algorithms proposed in literature, including heuristic algorithms and solvers (Rosenkrantz et al., 1977; Golden et al., 1980; Optimization et al., 2020; Applegate et al., 2006; Helsgaun, 2017), and approaches based on training deep neural networks (Kool et al., 2019; Deudon et al., 2018; Chen & Tian, 2019; Nazari et al., 2018). Specifically, given a set of n nodes with their coordinates, the TSP task is to find the shortest route that traverses all nodes from the starting node and finally returns to the starting node.

Our optimization process with LLMs starts from 5 randomly generated solutions, and each optimization step produces at most 8 new solutions. We present the meta-prompt in Figure 20 of Appendix C.1. We generate the problem instances by sampling n nodes with both x and y coordinates in $[-100, 100]$. We use the Gurobi solver (Optimization et al., 2020) to construct the oracle solutions and compute the optimality gap for all approaches, where the optimality gap is defined as the difference between the distance in the solution constructed by the evaluated approach and the distance achieved by the oracle solution, divided by the distance of the oracle solution. Besides evaluating OPRO with different LLMs including `text-bison`, `gpt-3.5-turbo` and `gpt-4`, we also compare OPRO to the following heuristics:

- `Nearest Neighbor (NN)`. Starting from an initial node, the solution is constructed with the nearest neighbor heuristic: At each step, among the remaining nodes that are not included in the current partial solution, NN selects the node with the shortest distance to the end node of the partial solution, and adds it as the new end node. The process finishes when all nodes have been added to the solution.
- `Farthest Insertion (FI)`. One caveat of the nearest neighbor heuristic is that it does not take the distance between the start and end node into consideration when constructing partial solutions. To address this issue, FI aims to optimize the cost of inserting new nodes into the partial solution at each step. Define the minimal insertion cost of adding a new node k as

Table 3: Results of the Traveling Salesman Problem (TSP) with different number of nodes n, where each n contains 5 problems. "# steps" calculates the mean ± standard error of optimization steps for successful runs that find the optimal solution. "# successes" counts the number of problems that OPRO results in the optimal solution. When no optimal solution is found for any evaluated problem, the corresponding number of steps is N/A.

n	optimality gap (%)					# steps (# successes)		
	NN	FI	`text-bison`	`gpt-3.5-turbo`	`gpt-4`	`text-bison`	`gpt-3.5-turbo`	`gpt-4`
10	13.0 ± 1.3	3.2 ± 1.4	**0.0** ± 0.0	**0.0** ± 0.0	**0.0** ± 0.0	40.4 ± 5.6 **(5)**	46.8 ± 9.3 **(5)**	**9.6** ± 3.0 **(5)**
15	9.4 ± 3.7	1.2 ± 0.6	4.4 ± 1.3	1.2 ± 1.1	**0.2** ± 0.2	N/A (0)	202.0 ± 41.1 **(4)**	**58.5** ± 29.0 **(4)**
20	16.0 ± 3.9	**0.2** ± 0.1	30.4 ± 10.6	4.4 ± 2.5	1.4 ± 0.6	N/A (0)	438.0 ± 0.0 (1)	**195.5** ± 127.6 **(2)**
50	19.7 ± 3.1	**9.8** ± 1.5	219.8 ± 13.7	133.0 ± 6.8	11.0 ± 2.6	N/A (0)	N/A (0)	N/A (0)

$c(k) = \min_{(i,j)} d(i,k) + d(k,j) - d(i,j)$, where i and j are adjacent nodes in the current tour, and $d(\cdot,\cdot)$ represents the distance between two nodes. At each step, FI adds a new node that maximizes the minimal insertion cost.

We present the results in Table 3. We randomly generate 5 problem instances for each number of nodes n. In addition to measuring the optimality gap, on problems where the LLM finds the optimal solutions, we also show the number of optimization steps taken to reach the global optimum. First, we observe that `gpt-4` significantly outperforms `gpt-3.5-turbo` and `text-bison` across all problem sizes. Specifically, on smaller-scale problems, `gpt-4` reaches the global optimum about $4\times$ faster than other LLMs. On larger-scale problems, especially with $n = 50$, `gpt-4` still finds solutions with a comparable quality to heuristic algorithms, while both `text-bison` and `gpt-3.5-turbo` get stuck at local optima with up to $20\times$ worse optimality gaps.

On the other hand, the performance of OPRO degrades dramatically on problems with larger sizes. When $n = 10$, all LLMs find the optimal solutions for every evaluated problem; as the problem size gets larger, the OPRO optimality gaps increase quickly, and the farthest insertion heuristic starts to outperform all LLMs in the optimality gap.

Limitations. We would like to note that OPRO is designed for neither outperforming the state-of-the-art gradient-based optimization algorithms for continuous mathematical optimization, nor surpassing the performance of specialized solvers for classical combinatorial optimization problems such as TSP. Instead, the goal is to demonstrate that LLMs are able to optimize different kinds of objective functions simply through prompting, and reach the global optimum for some small-scale problems. Our evaluation reveals several limitations of OPRO for mathematical optimization. Specifically, the length limit of the LLM context window makes it hard to fit large-scale optimization problem descriptions in the prompt, e.g., linear regression with high-dimensional data, and traveling salesman problems with a large set of nodes to visit. In addition, the optimization landscape of some objective functions are too bumpy for the LLM to propose a correct descending direction, causing the optimization to get stuck halfway. We further elaborate our observed failure cases in Appendix A.

4 Application: Prompt Optimization

Next, we demonstrate the effectiveness of OPRO on prompt optimization, where the objective is to find the prompt that maximizes task accuracy. We first introduce the problem setup, then illustrate the meta-prompt design.

4.1 Problem Setup

We focus on prompt optimization for natural language tasks, where both the input and output are in the text format. The task is represented as a dataset with training and test splits, where the training set is used to calculate the training accuracy as the objective value during the optimization process, and we compute the test accuracy on the test set after the optimization finishes. While traditional optimization often requires a decently large training set, our experiment shows that a small number or fraction of training samples (e.g., 3.5% of the training set for GSM8K (Cobbe et al., 2021), 20% for Big-Bench Hard (Suzgun et al., 2022)) is sufficient. The objective function evaluator is an LLM