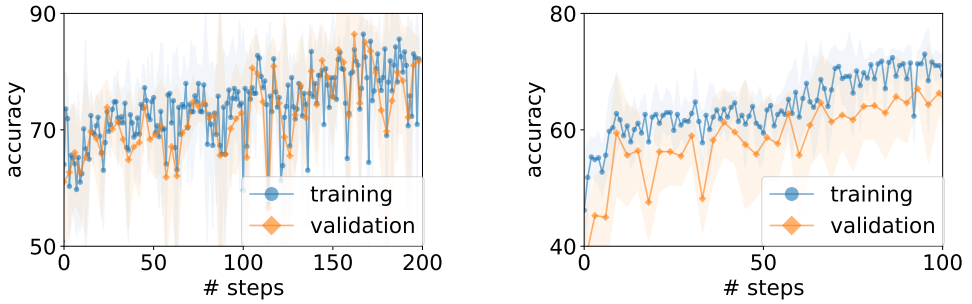


(a) BBH snarks, `PaLM 2-L` as scorer, `PaLM 2-L-IT` as optimizer, starting from "Let's solve the problem."

(b) BBH sports_understanding, `text-bison` as scorer, `gpt-3.5-turbo` as optimizer, starting from the empty string

Figure 11: **Overfitting analysis.** The exemplars are splitted to 1/3 training, 1/3 validation and 1/3 test. We compute the validation accuracy every 3 steps. The training/validation dots are the average training/validation accuracies across 3 optimization repetitions, respectively, and the shaded regions represent standard deviations.

set, the validation accuracy curves trend up and down alongside the training curves in both prompt optimization settings.

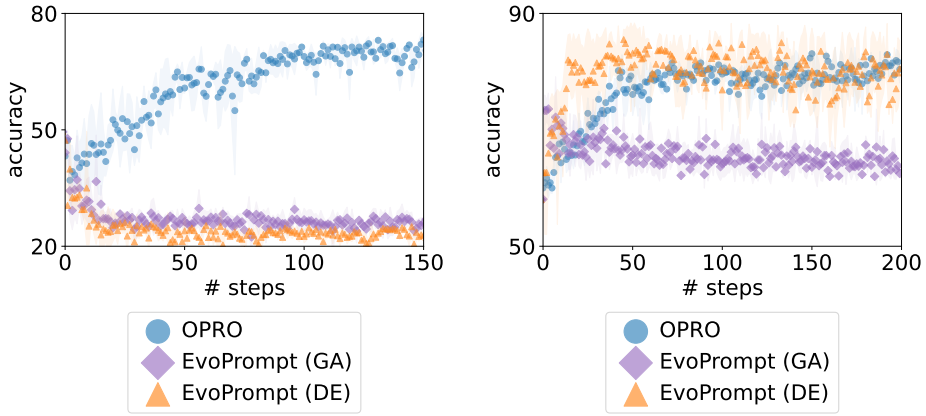
Of course, overfitting still occurs in the instructions found by our prompt optimization: in Table 7 and 10, our training accuracies are often 5%-20% higher than our test accuracies, despite that our test and overall accuracies are still mostly higher than human-written counterparts. Setting aside a larger training set and optimizing for fewer steps (early stopping) may help reduce overfitting.

5.5 Comparison with EvoPrompt

Some concurrent works on prompt optimization propose meta-prompts that explicitly ask the LLM to perform mutation and crossovers of existing prompts (Fernando et al., 2023; Guo et al., 2023). In our evaluation, we compare our approach to the Genetic Algorithm (GA) and Differential Evolution (DE) versions of EvoPrompt (Guo et al., 2023). Specifically, in the GA meta-prompt, given two prompts, the meta-prompt instructs the LLM to cross over the two prompts and generates a new one, then mutates the newly generated prompt to produce the final prompt. DE extends the GA meta-prompt to include more detailed instructions, e.g., asking the LLM to identify different parts between the two given prompts before performing the mutation. This is in contrast with OPRO, which leverages the optimization trajectory including multiple past prompts, instead of only 2 previous prompts. Meanwhile, OPRO also provides the LLM with richer information to facilitate the understanding of the optimization problem, including exemplars and task accuracies of different prompts.

Figure 12 presents the results on GSM8K and BBH sports_understanding benchmarks, where we use `gpt-3.5-turbo` as the optimizer. On GSM8K, the initial instructions of all approaches are "Let's solve the problem." and "Here is the answer.", which are simple and generic. Again, we observe that OPRO performance steadily improves with more optimization steps. On the other hand, both versions of EvoPrompt even degrade the performance on GSM8K. The main reason is because EvoPrompt does not utilize exemplars for prompt optimization, thus it lacks the understanding of the task to optimize for. In this way, EvoPrompt relies on good-quality and task-specific initial prompts to optimize from.

Given this observation, we provide more task-specific initial instructions for experiments on BBH sports_understanding, which are "Solve the sports understanding problem." and "Give me the answer to sports understanding." In this case, EvoPrompt (DE) is able to find better prompts than the initial ones, but the optimization curve is less stable than OPRO. This indicates that leveraging the optimization trajectory helps the LLM to identify promising directions to improve existing prompts.



(a) GSM8K, `PaLM 2-L` scorer, A_begin

(b) BBH sports_understanding, `text-bison` scorer, Q_begin

Figure 12: **Comparison with EvoPrompt in prompt optimization.** We use the `gpt-3.5-turbo` optimizer for both experiments. "EvoPrompt (GA)" uses the meta-prompt from Guo et al. (2023), Figure 1; "EvoPrompt (DE)" uses the meta-prompt from Guo et al. (2023), Figure 2. All optimizations in (a) use the pre-trained `PaLM 2-L` scorer and start from two simple instructions "Let's solve the problem." and "Here is the answer."; all optimizations in (b) use the `text-bison` scorer and start from two richer (task-specific) instructions "Solve the sports understanding problem." and "Give me the answer to sports understanding.". The dots are the average values across 3 optimization repetitions, and the shaded regions represent standard deviations. We use temperature 1.0 for OPRO and temperature 0.5 for EvoPrompt, same as the default settings in respective works.

6 Related Work

Prompt optimization. Prior works have developed soft prompt-tuning methods that optimize the prompt represented as task-specific continuous vectors (Lester et al., 2021; Li & Liang, 2021; Liu et al., 2021; Qin & Eisner, 2021), as well as performing discrete prompt optimization by gradient-guided search (Shin et al., 2020; Wen et al., 2023; Gao et al., 2020; Chen et al., 2023d) and reinforcement learning (Deng et al., 2022; Zhang et al., 2023). These approaches become inapplicable when there is only API access to the LLM. Other works designed edit-based approaches for gradient-free prompt optimization (Xu et al., 2022; Prasad et al., 2022), where the editing can be done with human-defined operations (e.g., swapping two phrases) (Prasad et al., 2022) or language models (e.g., back translation) (Xu et al., 2022). Some recent works investigate LLMs for prompt optimization (Zhou et al., 2022b; Pryzant et al., 2023; Xu et al., 2023). Specifically, APE (Zhou et al., 2022b) first uses the LLM to generate initial instructions. Afterwards, APE selects top instructions with the highest accuracies, then prompts the LLM with each individual instruction to generate a semantically similar variant of the initial instruction. APO (Pryzant et al., 2023) in each step instructs the LLM to produce text feedback on how to update an old instruction. Different from edit-based approaches, the optimizer LLM in our work directly generates new instructions at each optimization step, and the optimizer LLM is merely asked to improve the task accuracy without being required to imitate past instructions. Compared to Zhou et al. (2022b) and Pryzant et al. (2023), our optimization process incorporates the past generated instructions with their scores in the meta-prompt, enabling the optimizer LLM to discover common patterns of high-quality instructions.

Prompting with natural language feedback. A recent line of work investigates approaches to improve the LLM performance by prompting with natural language feedback to revise the model output, which has shown effectiveness in reducing harmful LLM outputs (Bai et al., 2022; Ganguli et al., 2023), improving reasoning (Shinn et al., 2023; Madaan et al., 2023) and code generation performance (Chen et al., 2023e; Olausson et al., 2023; Shinn et al., 2023; Chen et al., 2023b), dialogue applications (Nair et al., 2023; Madaan et al., 2023; Yuan et al., 2023), and so on (Kim et al., 2023; Wang et al., 2023). Specifically, Yuan et al. (2023) develops a human-in-the-loop framework for deriving system-level feedback from a collection of instance-level feedback, which is then used

for refining data. In our work, the optimizer LLM utilizes the optimization trajectory in the prompt, which implicitly requires the LLM to summarize the common characteristics among solutions with similar scores. We consider incorporating explicit natural language feedback on generated solutions for later optimization steps as future work.

Tuning language models for optimization. Some previous works tune or prompt language models to behave as mutation and crossover operators in evolutionary algorithms. Meyerson et al. (2023) utilizes language models with few-shot exemplars to propose evolutionary cross-overs on tasks such as image and code generation. In Lehman et al. (2022), the large language model trained on code diff generation is used as the mutation operator, and they further design a fine-tuning method to improve performance in the Sodarace domain for robot simulation. EvoPrompting (Chen et al., 2023a) uses large language models to evolve neural network architectures, where they combine evolutionary search with soft prompt tuning. With respect to taking the trajectory as the input for optimization, OptFormer (Chen et al., 2022) trains a transformer model on large collections of hyperparameter optimization data. On the other hand, our work performs optimization solely by prompting without additional training.

7 Conclusion

We embark on employing LLMs as optimizers, where the LLM progressively generates new solutions to optimize an objective function. We first motivate OPRO with linear regression and traveling salesman problems, then proceed to prompt optimization as a concrete application. Our evaluation demonstrates that LLMs have the capacity of gradually improving the generated solutions based on the past optimization trajectory. Interestingly, on small-scale traveling salesman problems, OPRO performs on par with some hand-crafted heuristic algorithms. For prompt optimization, optimized prompts outperform human-designed prompts on GSM8K and Big-Bench Hard by a significant margin, sometimes over 50%.

A number of unresolved questions are open for future research on LLMs for optimization. In general, how to reduce the sensitivity to initialization and better balance exploitation with exploration remains a challenge. Specifically, for prompt optimization, one limitation of our current implementation is that the optimizer LLM does not effectively utilize error cases in the training set to infer promising directions to improve the generated instructions. In our experiments, we tried including error cases in the meta-prompt rather than randomly sampling from the training set at each optimization step, but the results are similar, indicating that the error cases alone are not informative enough for the optimizer LLM to grasp the cause of the wrong prediction. Another limitation is that prompt optimization requires a training set to compute the accuracy that guides the optimization process. Currently the training set at least contains tens of samples, so that the optimized prompt does not severely overfit to the training samples. A promising direction is to incorporate richer feedback about the error cases besides the aggregated accuracy, and summarize the key features that distinguish between high-quality and low-quality generated prompts in the optimization trajectory. Such information may inform the optimizer LLM of how to more efficiently improve over the past generated instructions, and potentially further reduce the example set size needed for prompt optimization.

Ethics Statement

This work uses synthetic math problems for linear regression and traveling salesman problems, and uses public datasets like GSM8K and Big-Bench Hard for prompt optimization. These tasks have been commonly used in similar works and should not be regarded controversial. There is a peril that LLMs may generate harmful information that poses safety risks; how to safeguard model behavior remains valuable future work.

Reproducibility Statement

We evaluate on public benchmarks. The `text-bison` API is available at: `https://cloud.google.com/vertex-ai/docs/generative-ai/learn/models`. The GPT models are available here: `http://openai.com/api/`. This work uses `gpt-3.5-turbo-0613` and `gpt-4-0613`.