Large Language Models as Optimizers

Parth Darshan (B22CS040)

Q1. Executive snapshot of the paper

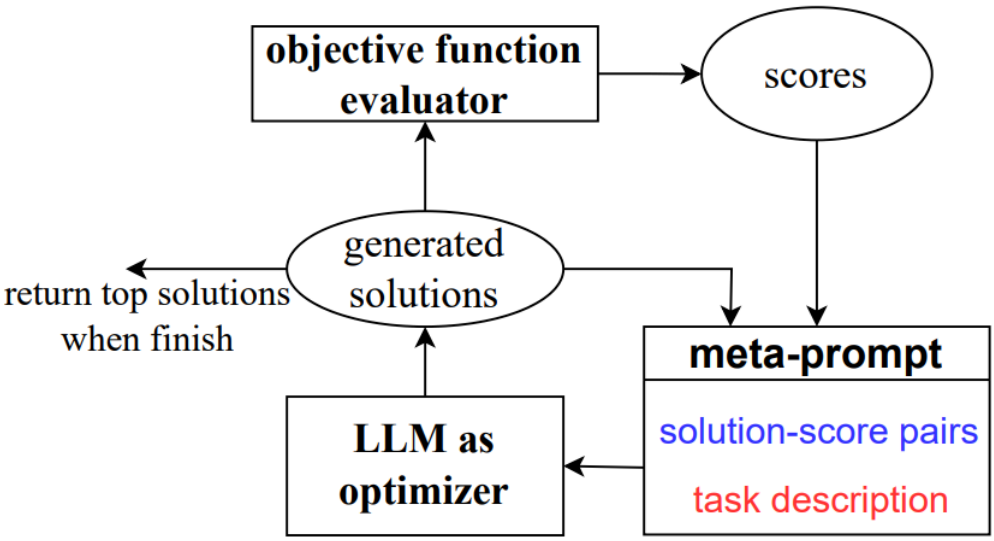
Problem & Motivation

Optimization is central to machine learning, where models iteratively improve through algorithms such as Gradient Descent and ADAM. In contrast, users of Large Language Models (LLMs) typically lack access to model parameters and rely solely on API calls. This creates a challenge: maximizing performance depends on crafting effective prompts, a process that is often ad hoc and labor-intensive.

The paper introduces **Optimization by Prompting (OPRO)**, a framework that treats prompt refinement as an optimization task over natural language itself. By leveraging “textual gradients,” OPRO iteratively adjusts prompts to enhance task performance without requiring access to the model internals.

Experiments across multiple LLMs show that OPRO generates optimized prompts that significantly outperform human-designed ones. Notably, the method yields up to an **8% improvement on GSM8K** and as much as a **50% improvement on Big-Bench Hard**, demonstrating its potential as a scalable, automated approach to prompt optimization.

Core Idea



The core idea is to maintain a **meta-prompt** that tracks task details, solution–score pairs, and top-performing prompts from earlier iterations. An **Optimizer LLM** uses this meta-prompt to generate the next instruction, which is then applied to the **Task LLM** handling the target problem. The Task LLM produces outputs in a structured format (e.g., JSON/XML), evaluated with metrics like Cross-Entropy Loss, Accuracy, or F1-Score. Each instruction–evaluation pair is appended to the meta-prompt, enabling iterative refinement for better performance.

Main Contributions

1. Proposes OPRO framework that treats prompt-optimization process using a meta-prompt to record past solution-score pairs and generate new prompts, enabling iterative improvement without access to model gradients.
2. Demonstrates that LLMs can serve as optimizers in both **continuous** (linear regression) and **discrete** (Traveling Salesman Problem) settings, thus showing broad applicability beyond purely language tasks.
3. Explores design choices like meta-prompt structure (optimization trajectory, examples, meta-instructions), solution sampling vs. greedy evaluation, and trade-offs between exploration/exploitation for stability and performance.
4. Provides evidence of **transferability**: optimized prompts trained on one dataset (e.g. GSM8K) generalize to related tasks, suggesting OPRO captures prompt properties beyond overfitting to a specific dataset.

Takeaway

1. LLMs can iteratively refine prompts using textual feedback, effectively acting as optimizers without direct gradient access. This reframes prompt design as a systematic process rather than manual trial-and-error.
2. Optimized prompts via OPRO significantly outperform human-crafted prompts across reasoning and benchmark tasks, proving that automated prompt optimization can yield scalable, transferable improvements.
3. OPRO liabilities lie on the fact that it is dependent on the LLM heuristic of the Optimizer LLM, i.e. if smaller models perform worse when they are Optimizer LLM.

Q2. Method deep-dive

Mechanics

OPRO frames prompt optimization as an iterative process between two LLMs:

- **Optimizer LLM**: Generates candidate instructions (meta-prompt guided).
- **Scorer LLM**: Evaluates these instructions by computing task accuracy on a training split.

The **objective** is to maximize accuracy =>

$$\max_{\text{prompt}} \ \text{Acc}_{\text{train}}(\text{ScorerLLM}(\text{prompt}))$$

Optimization loop:

(1) meta-prompt stores task examples + prior instructions + scores,
(2) optimizer LLM proposes new instructions,
(3) scorer LLM evaluates,
(4) iteration continues until convergence or number of iteration exceeds pre-set value.

Ablations

An ablation on **training data fraction** showed OPRO works effectively with very small subsets (e.g., 3.5% of GSM8K training set, 20% of BBH). This demonstrates sample efficiency and supports the design choice of lightweight scorer evaluations.

Compute & Data

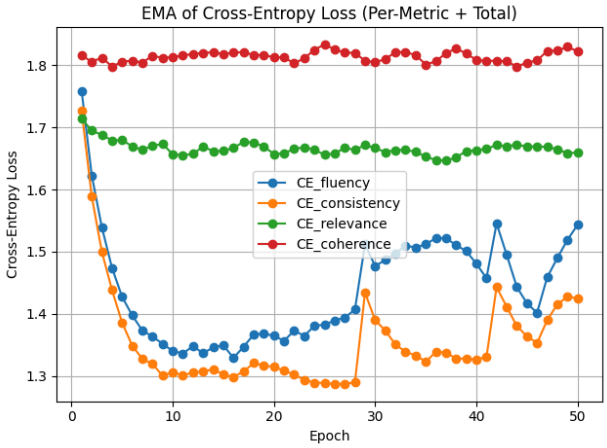
Compute: OPRO itself does not require additional LLM training; it operates entirely through inference (API-style calls). Main cost is multiple scorer LLM evaluations per optimization round.

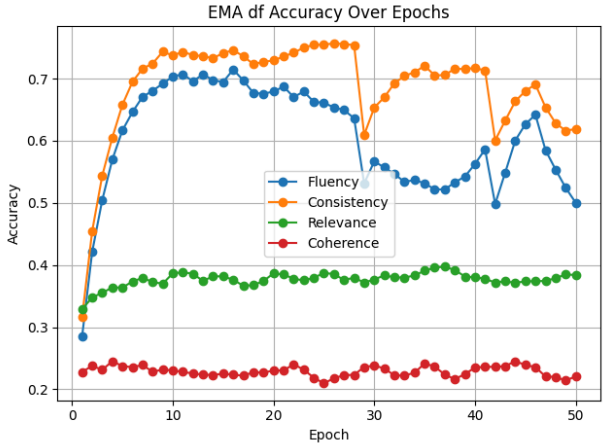
Data sources: Natural language reasoning datasets (GSM8K, Big-Bench Hard). Modalities: purely text-based (input questions + textual instructions, output answers).

Q4. Run on a new dataset

For this part, I've used the SummEval Dataset's subset of samples to test it out on a multi-objective problem. SummEval Dataset consists of a model's summary along with the text and each summary has a human annotated score as the ground truth on Fluency, Relevance, Consistency and Coherence between the range of 1|2|3|4|5. For this experiment, I've used metallama/llama-3.1-8b-instruct model, with the objective to minimize all losses simultaneously, (Sort of Linear Scalarization in statistical models).

Repository: https://github.com/b22cse040/PromptOptimization
Results: Link => See file metric_history_opro_8b.txt for prompts and performance.





Evaluation table => In this case, I am actually evaluating whether Prompt-Optimization in a multi-task setting is feasible than just doing all those tasks separately.

Tasks	**Parameters**	**LLM**	**Test Avg. Accuracy**	**Test Avg. F1**
All	Top-k previous prompts = 10 Metric shown = Cross-Entropy Loss syntax = Granular Epochs = 50	LLama 3.1 8b	Avg: 0.543 **Relevance: 0.352** **Coherence: 0.385** **Fluency: 0.704** Consistency: 0.758	Avg: 0.4895 Relevance: 0.217 Coherence: 0.264 Fluency: 0.716 Consistency: 0.761
Separate tasks	Top-k previous prompts = 10 Metric shown = Cross-Entropy	LLama 3.1 8b	Avg: 0.4890 Relevance: 0.326	Avg: 0.4430 Relevance: 0.166

	Loss syntax = Granular Epochs = 50		Coherence: 0.379 Fluency: 0.394 **Consistency: 0.856**	Coherence: 0.300 Fluency: 0.497 Consistency: 0.81

As evident from the table, the accuracies of Relevance, Coherence and Fluency improved although Consistency decreased. So, Optimization with Prompting (OPRO) is feasible in Prompt Optimization in a multi-task setting.

Setup

There are 480 test set examples and 160 training examples, although there isn't any training.. The purpose of the so-called training is to find the best performing prompt on the training dataset that would be the most optimal on the test dataset.

Starting with the rater LLM => We have the task description, previous top-K best performing prompts (Lowest loss or highest accuracy maintained by a TopKHeap) that form the meta-prompt. The task of this meta-prompt is to generate a new instruction to judge the summaries of the scores.
Once the instruction is retrieved from the rater LLM, it is used to evaluate each of the training samples where we get a JSON Output of the prediction for each example.

Once the results are obtained, we need to evaluate the f1, loss and accuracy score for the generated instruction. Let's name this dictionary as performance.
Recommender LLM => In the recommender LLM, we give (instruction, performance) a task of generating a recommendation for this instruction that would help rater LLM refine the prompt and optimize the performance.

We push this element = (instruction, performance, recommendation) onto the top-K Heap. Which would show up on the meta-prompt to improve the performance for the next iterations.

After all the epochs have been run, we select the instruction at the top of the heap and evaluate on the test set.

Q5. Robustness to Messy Inputs

Errors in the prompts may be unicode / typo / emoji.

Pertubated_Prompt
What is the capital of France?
Who wrote 🎭 Romeo and Juliet
WhaT planet is known as the Red Planet?
How many continents are there on 🌍 Earth
What is the currency 0f Japan
Who painted the Mona Lisa 🎨
What gas do humans need to breate?

It is noted that =>

	Error_Type	Clean_Accuracy	Noisy_Accuracy	Abs_Delta	Rel_Delta_%	Count
0	Emoji	0.800000	0.600000	-0.2000	-25.000000	35
1	Typo	0.700000	0.600000	-0.1000	-14.285714	10
2	Unicode	0.833333	0.645833	-0.1875	-22.500000	48

In the baseline (no error-awareness prompting), performance drops across all error types. Clean inputs yield higher accuracy than noisy ones: Emoji accuracy falls from 0.80 → 0.60, Typo from 0.70 → 0.60, and Unicode from 0.83 → 0.65. The relative drop ranges from –14% to –25%, with Unicode showing the largest data sample (48 cases). Overall, the LLM struggles to maintain robustness against natural input errors when not explicitly guided to handle them. This happens because the model's training distribution is mostly clean text, so unexpected characters or distortions break tokenization. Without explicit error-handling cues, it lacks strategies to normalize or reinterpret noisy inputs.

	Error_Type	Clean_Accuracy	Noisy_Accuracy	Abs_Delta	Rel_Delta_%	Count
0	Emoji	0.800000	0.514286	-0.285714	-35.714286	35
1	Typo	0.700000	0.600000	-0.100000	-14.285714	10
2	Unicode	0.833333	0.583333	-0.250000	-30.000000	48

After awareness, it did in fact perform worse.. ;-;

```
input_prompt = f'''
Your task is to generate a one-word answer to the following question.
Please note that the question may contain unicode/emoji/typo errors as well.
Answer the following question.
Question: {prompt}

Return only the answer and nothing else.
'''
```

Q6.