# LARGE LANGUAGE MODELS AS OPTIMIZERS

**Chengrun Yang*** **Xuezhi Wang** **Yifeng Lu** **Hanxiao Liu**
**Quoc V. Le** **Denny Zhou** **Xinyun Chen***

{chengrun, xuezhiw, yifenglu, hanxiaol}@google.com
{qvl, dennyzhou, xinyunchen}@google.com

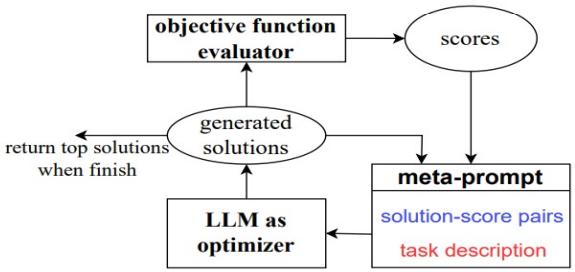
Google DeepMind * Equal contribution

Presenter: Junqi Qu

# Optimization via Natural Language

- **Optimization is a universal problem.** From tuning hyperparameters to designing molecules, we are constantly searching for optimal solutions.
- **The Challenge:** Many real-world problems are "derivative-free," meaning we cannot compute gradients to guide the search. This requires specialized, often hand-designed heuristic algorithms.
- **Idea:** Can we reframe optimization as a natural language task? Instead of writing a formal algorithm, can we simply *describe* the problem to an LLM and ask it to find better solutions iteratively?

# OPRO: Optimization by PROmpting Framework




- **OPRO** is an iterative framework where **Optimizer**.
- **The Loop:**
  - **Generate:** The Optimizer LLM receives a meta-prompt containing the problem description and past solutions. It generates a new batch of candidate solutions.
  - **Evaluate:** An objective function (the "Evaluator" or "Scorer") calculates the score for each new solution.
  - **Update:** The new solutions and their scores are added back into the meta-prompt.

I have some texts along with their corresponding scores. The texts are arranged in ascending order based on their scores, where higher scores indicate better quality.

text:
Let's solve the problem.
score:
63

(. . . more instructions and scores . . . )

The following exemplars show how to apply your text: you replace <INS> in each input with your text, then read the input and give an output. We say your output is wrong if your output is different from the given output, and we say your output is correct if they are the same.

input:
Q: Alannah, Beatrix, and Queen are preparing for the new school year and have been given books by their parents. Alannah has 20 more books than Beatrix. Queen has 1/5 times more books than Alannah. If Beatrix has 30 books, how many books do the three have together?
A: <INS>
output:
140

(. . . more exemplars . . . )

Write your new text that is different from the old ones and has a score as high as possible. Write the text in square brackets.

Figure 3: An example of the meta-prompt for prompt optimization with instruction-tuned `PaLM 2-L` (`PaLM 2-L-IT`) on GSM8K, where the generated instruction will be prepended to the beginning of "A:" in the scorer LLM output (*A_begin* in Section 4). <INS> denotes the position where the generated instruction will be added. The blue text contains solution-score pairs; the purple text describes the optimization task and output format; the orange text are meta-instructions. Appendix E.2 presents the full meta-prompts for other optimizer LLMs.

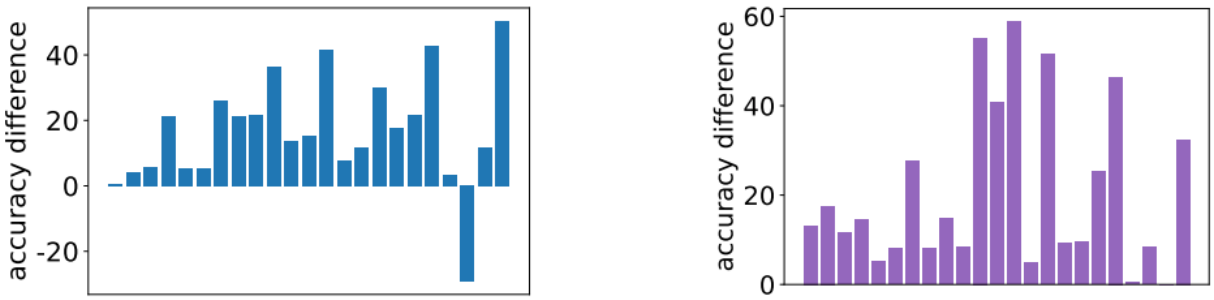
# Primary Application: Prompt Optimization

- **The Problem:** Prompt engineering is a difficult, manual optimization task. The search space is discrete, high-dimensional, and non-intuitive. Semantically similar prompts can yield vastly different performance.
- **OPRO's Approach:**
  - **Goal:** Find a natural language instruction (a prompt) that maximizes a task's accuracy.
  - **Optimizer LLM:** A powerful model (e.g., PaLM 2-L-IT, GPT-4) that generates new instructions.
  - **Scorer LLM:** A model that evaluates the generated instruction's performance on a small training set. This can be the same as or different from the optimizer.

# Key results

Table 1: Top instructions with the highest GSM8K zero-shot test accuracies from prompt optimization with different optimizer LLMs. All results use the pre-trained `PaLM 2-L` as the scorer.

| Source | Instruction | Acc |
|---|---|---|
| *Baselines* | | |
| (Kojima et al., 2022) | Let's think step by step. | 71.8 |
| (Zhou et al., 2022b) | Let's work this out in a step by step way to be sure we have the right answer. | 58.8 |
| | (empty string) | 34.0 |
| *Ours* | | |
| `PaLM 2-L-IT` | Take a deep breath and work on this problem step-by-step. | **80.2** |
| `PaLM 2-L` | Break this down. | 79.9 |
| `gpt-3.5-turbo` | A little bit of arithmetic and a logical approach will help us quickly arrive at the solution to this problem. | 78.5 |
| `gpt-4` | Let's combine our numerical command and clear thinking to quickly and accurately decipher the answer. | 74.5 |

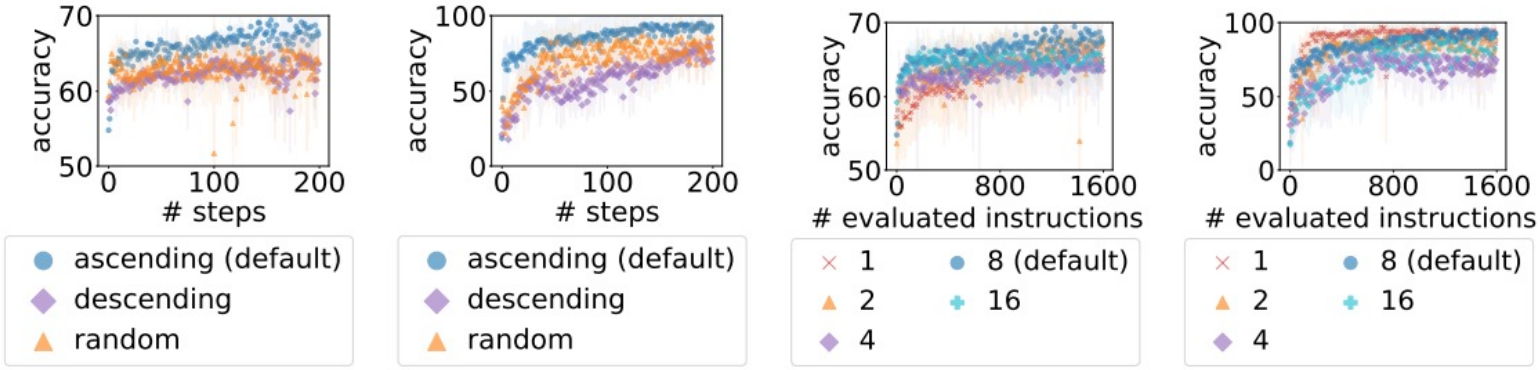



(a) ours minus "Let's think step by step." (b) ours minus empty starting point

Figure 4: On 23 BBH tasks, the accuracy differences among instructions found by prompt optimization (with the `PaLM 2-L` scorer and the `PaLM 2-L-IT` optimizer), "Let's think step by step.", and the empty string (optimization starting point). The bar charts with task names and those with the `text-bison` scorer are deferred to Figure 19 in Appendix J.1.

# Ablation studies

- **Instruction Ordering is Critical:** Sorting the trajectory from lowest to highest score works best. Reversing the order or randomizing it hurts performance significantly. This suggests a strong recency bias in the optimizer LLM.
- **Batching Improves Stability:** Generating a "mini-batch" of 8 instructions per step provides the best balance between exploration and optimization stability, similar to mini-batch SGD.
- **Explicit Scores Matter:** Providing the numerical accuracy scores is more effective than just showing a ranked list of instructions.
- **Task Exemplars are Essential:** Without a few examples of the task, the optimizer doesn't have enough context and fails to optimize effectively.




(a) instruction ordering (GSM8K)

(b) instruction ordering (sports_understanding)

(c) # single-step instructions (GSM8K)

(d) # single-step instructions (sports_understanding)

Figure 5: **Ablation studies.** The dots are the average values across 3 optimization repetitions, and the shaded regions represent standard deviations. In Figure (c) and (d), the x-axis represents the total number of evaluated instructions through the optimization; e.g., we run 200 optimization steps when sampling 8 instructions in each step, run 400 steps when sampling 4 instructions in each step, etc.

# Limitations and Failure Cases

- The paper is transparent about the limitations of using LLMs as optimizers:
  - **Hallucination:** The optimizer can hallucinate function values, highlighting the need for external tools for reliable calculation.
  - **Instruction Following:** Models don't always adhere to negative constraints, like "generate a *new* solution".
  - **Getting Stuck:** Like other optimizers, OPRO can get stuck in local optima, especially on bumpy or deceptive loss landscapes (e.g., Rosenbrock function).
  - **Sensitivity:** The process can be sensitive to the initial starting points.

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. See Appendix B.1 for experiment setup.

| $w_{\text{true}}$ | $b_{\text{true}}$ | 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 |

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 $\pm$ 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. See Appendix B.2 for experiment setup.

| $n$ | 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 $\pm$ 1.3 | 3.2 $\pm$ 1.4 | **0.0** $\pm$ 0.0 | **0.0** $\pm$ 0.0 | **0.0** $\pm$ 0.0 | 40.4 $\pm$ 5.6 **(5)** | 46.8 $\pm$ 9.3 **(5)** | **9.6** $\pm$ 3.0 **(5)** |
| 15 | 9.4 $\pm$ 3.7 | 1.2 $\pm$ 0.6 | 4.4 $\pm$ 1.3 | 1.2 $\pm$ 1.1 | **0.2** $\pm$ 0.2 | N/A (0) | 202.0 $\pm$ 41.1 **(4)** | **58.5** $\pm$ 29.0 **(4)** |
| 20 | 16.0 $\pm$ 3.9 | **0.2** $\pm$ 0.1 | 30.4 $\pm$ 10.6 | 4.4 $\pm$ 2.5 | 1.4 $\pm$ 0.6 | N/A (0) | 438.0 $\pm$ 0.0 (1) | **195.5** $\pm$ 127.6 **(2)** |
| 50 | 19.7 $\pm$ 3.1 | **9.8** $\pm$ 1.5 | 219.8 $\pm$ 13.7 | 133.0 $\pm$ 6.8 | 11.0 $\pm$ 2.6 | N/A (0) | N/A (0) | N/A (0) |