Teach Better or Show Smarter? On Instructions and Exemplars in Automatic Prompt Optimization.

Xingchen Wan, Ruoxi Sun, Hootan Nakhost, Sercan Ö. Arık

NeurIPS 24 (paper)

Prompts and Automatic Prompt Optimization (APO)

Prompts consist of instruction(s) (i.e., to ***teach***) and, if any, exemplars (or demonstrations) (i.e., to ***show***)

$$P(x) = [\underline{I}, \underline{e_1, \ldots, e_k}, x]$$

Automatic prompt optimization (APO) frames prompt engineering as optimization

$$P^*(x) = \arg\max_{P(\cdot)\sim\mathcal{P}} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{val}}}\left[g\Big(f_{\text{LLM}}\big(P(x)\big), y\Big)\right],$$

A labeled validation set is typically required

Exemplar Optimization (EO)

- Targets **exemplars.**
- Arguably how APO started (before instruction-following models)!
- Approaches:
  - *Heuristic-based*: similarity (retrieval), calibration / entropy, diversity...
  - *Optimization-based*: influence function, sensitivity, learning-based (learning a retriever or selection based on validation performance (e.g., DSPy)

Instruction Optimization (IO)

- Targets **instructions.**
- **More popular recently.**
- Typically uses another LLM to rewrite instructions in a human-readable format based on **paraphrasing instructions** and/or the meta-instructions, **reflecting on errors**, or both.
- Approaches:
  - *Paraphrasing-based*: APE, EvoPrompt, InstructZero, PromptBreeder...
  - *Reflection-based*: ProTeGi, PromptAgent
  - *Implicit*: OPRO

*Google papers.

Research Questions

IO and EO address the same overarching problem

but have evolved rather independently:

- Many EO approaches predate instruction tuning, so there are minimal instruction optimization.
- IO approaches require labeled dataset, but **only** use them to evaluate a validation score and then use random exemplars / no exemplars at all
    - Why? Because authors would like to do one thing at a time
- Relative dearth of works targeting **both**.

likelihoods at the time of writing.

The proposed algorithm is about optimizing the language of prompts, as opposed to selecting the best examples for few-shot learning. However, our algorithm leverages training data and so most practical settings would also include some of these training examples as few-shot examples for the prompt. Accordingly, all of the experiments of Section 3.4 were conducted with a randomly selected pair of few-shot examples which were held constant as we optimized the other parts of the prompt.

(Pryzant et al, 2023)

Research Questions

Practically, we **cannot** simply isolate them since they are interdependent.

This study aims to answer:

- What is the relative importance and performance impact of EO and IO, both in isolation and when combined together?
- How do we make the optimal use of the limited data and computational budget under the current APO framework?

Experimental Setup

IO methods

- **No IO:** *Let's think step by step.*
- **APE:** Optimizer LLM iteratively paraphrase the best performing instructions in the prev. Round
- **ProTeGi**: Optimizer LLM critique errors and revise instructions iteratively + beam search.
- **PromptAgent**: Similar to ProTeGi but uses MCTS.
- **OPRO**: Condition optimizer LLM with past trajectory of {instruction, scores} and implicitly ask the LLM to improve.

EO methods

Heuristic-based:

- **No EO:** no exemplars.
- **Random exemplars**
- **Nearest** (embedding distance)
- **Diversity**
- **All exemplars** (Gemini 1.5)

Optimization-based:

- **Random Search** (DSPy)
- **Mutation**

All combinations (outer product)

Models & Data

Models {target model / optimizer model }

- **PaLM 2** (text-bison-002) / **PaLM 2** (text-unicorn-001)
- **Gemini 1.0** Pro / **Gemini 1.0** Ultra
- **Gemini 1.5** Flash / **Gemini 1.5** Pro

Data

- **BIG-Bench Hard** (collection of 26 tasks: numerical reasoning, commonsense problem-solving, logical deduction, linguistic manipulation, machine translation, and tabular reasoning,...
- **MMLU**

Main Results

Table 1: Average BBH accuracy of all ES-IO combinations with **PaLM 2** (`text-bison-002`) target model and **PaLM 2** (`text-unicorn-001`) optimizer model. The last row/column show the max improvement over the *No IO* and/or *No ES* baseline of the respective row/column. The background shades indicate cost in terms of # prompt evaluations on $\mathcal{D}_{\text{val}}$ by the *target model*: gray cells requires no evaluation on $\mathcal{D}_{\text{val}}$ ($m = 0$) ; blue cells perform $m = 32$ evaluations to iteratively optimize instructions *or* exemplars; orange cells iteratively optimize exemplars m times on top of optimized instructions.

		Exemplar Selection (**ES**)						Max Δ
		No ES	Random	Nearest	Diversity	R.S.	Mutation	over *No ES*
Instruction Optimization (**IO**)	*No IO*	60.30	66.91	66.09	66.74	71.16	72.92	***+12.63***
	APE	64.96	69.11	69.01	70.81	75.88	76.25	***+11.28***
	ProTeGi	68.13	70.81	70.01	69.25	75.90	77.29	*+9.16*
	PromptAgent	65.66	67.65	67.82	67.35	72.51	72.77	*+7.11*
	OPRO	63.04	68.50	68.33	67.57	73.02	73.06	***+10.01***
	Max Δ over *No IO*	*+7.83*	*+3.89*	*+3.92*	*+4.07*	*+4.74*	*+4.37*	–

Table 2: Average BBH accuracy of seed instruction (*No IO*) and ProTeGi (best IO strategy from Table 1) with different ES strategies using **Gemini 1.0 Pro** target model and **Gemini 1.0 Ultra** optimizer model. Refer to Table 1 for further explanations.

	Exemplar Selection (**ES**)						Δ
	No ES	Random	Nearest	Diversity	R.S.	Mutation	ES
No IO	63.14	71.12	69.19	67.82	75.77	75.77	***+12.63***
ProTeGi	65.91	72.72	72.13	72.64	78.27	79.01	***+13.10***
Δ IO	*+2.77*	*+1.60*	*+2.94*	*+4.83*	*+2.50*	*+2.52*	–

Table 4: Average BBH accuracy of seed instruction (*No IO*), APE and ProTeGi (top 2 IO strategies from Table 1) with different ES strategies using **Gemini 1.5 Flash** target model and **Gemini 1.5 Pro** optimizer model. Refer to Table 1 for further explanations.

	Exemplar Selection (**ES**)							Δ
	No ES	Random	Nearest	Diversity	All	R.S.	Mutation	ES
No IO	75.07	80.02	81.71	81.52	80.43	83.25	82.42	***+8.18***
APE	77.52	81.20	83.71	81.55	81.20	85.04	84.76	***+7.54***
ProTeGi	80.39	82.40	82.61	82.29	83.52	84.47	84.49	*+4.10*
Δ IO	***+5.32***	*+2.20*	*+2.00*	*+0.77*	*+3.09*	*+1.79*	*+2.34*	–

Insight 1: Free exemplars are no-brainers for performance improvements

- ***Any*** EO improves, *with any IO, or no IO*
- This may not seem surprising but...
  1. Exemplars in this case are self-generated ("reinforced ICL") and come from the validation set -> *No additional data annotation cost*.
  2. Existing works often focus on "zero-shot" (i.e., No EO), but it may neither *reflect* nor *predict* LLM performance with better exemplar selection.

Instruction Optimization (IO)	No ES	Exemplar Selection (ES): Random	Nearest	Diversity	R.S.	Mutation	Max Δ over *No ES*
No IO	60.30	66.91	66.09	66.74	71.16	72.92	*+12.63*
APE	64.96	69.11	69.01	70.81	75.88	76.25	*+11.28*
ProTeGi	68.13	70.81	70.01	69.25	75.90	77.29	*+9.16*
PromptAgent	65.66	67.65	67.82	67.35	72.51	72.77	*+7.11*
OPRO	63.04	68.50	68.33	67.57	73.02	73.06	*+10.01*
Max Δ over *No IO*	*+7.83*	*+3.89*	*+3.92*	*+4.07*	*+4.74*	*+4.37*	–

Second best in "zero-shot"...

Worst with better exemplars...

Insight 1: Free exemplars are no-brainers for performance improvements

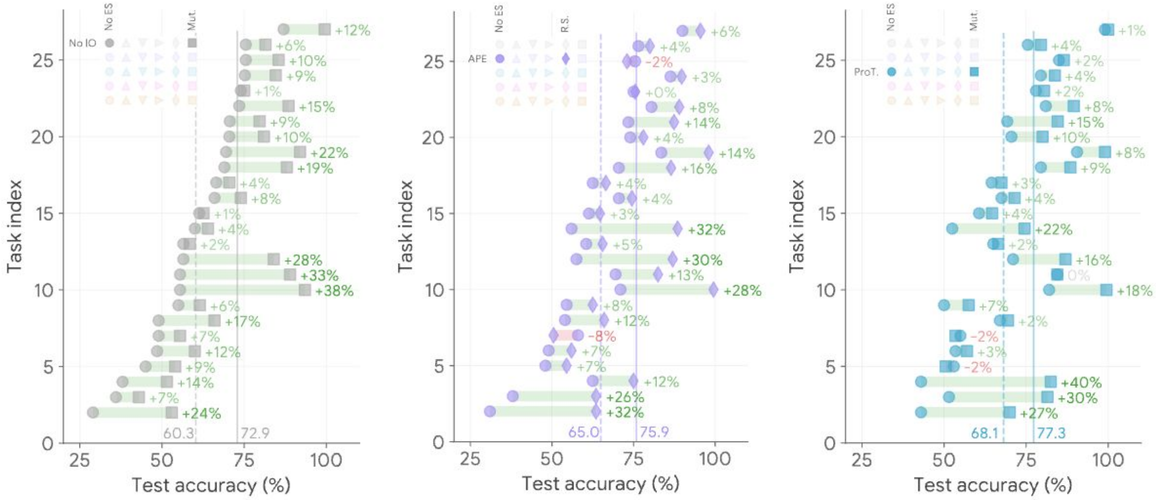


Figure 3: *Appropriate ES improves over any or no IO*: Task-specific BBH performance *with no instruction optimization* (**left**) and *with SoTA IO*: APE (**middle**) and ProTeGi (**right**) before and after applying exemplars found via Mutation (§3.1) on PaLM 2. Dashed and solid lines denote the average performance before and after exemplars, respectively. *Task index* is determined by the ascending order of test accuracy under seed instruction. Refer to additional visualization in App. B.3.

Insight 2: In many cases, EO > IO

In *isolation*: Gain from EO > Gain from IO

In *combination*: improvements stack up, but mostly attributable to better exemplars.

PaLM 2 (text-bison-002)

Instruction Optimization (IO)	Exemplar Selection (ES): *No ES*	Random	Nearest	Diversity	R.S.	Mutation	Max Δ over *No ES*
No IO	60.30	66.91	66.09	66.74	71.16	72.92	*+12.63*
APE	64.96	69.11	69.01	70.81	75.88	76.25	*+11.28*
ProTeGi	68.13	70.81	70.01	69.25	75.90	77.29	*+9.16*
PromptAgent	65.66	67.65	67.82	67.35	72.51	72.77	*+7.11*
OPRO	63.04	68.50	68.33	67.57	73.02	73.06	*+10.01*
Max Δ over *No IO*	*+7.83*	*+3.89*	*+3.92*	*+4.07*	*+4.74*	*+4.37*	–

Gemini 1.5 Flash

	Exemplar Selection (ES): *No ES*	Random	Nearest	Diversity	All	R.S.	Mutation	Δ ES
No IO	75.07	80.02	81.71	81.52	80.43	83.25	82.42	*+8.18*
APE	77.52	81.20	83.71	81.55	81.20	85.04	84.76	*+7.54*
ProTeGi	80.39	82.40	82.61	82.29	83.52	84.47	84.49	*+4.10*
Δ IO	*+5.32*	*+2.20*	*+2.00*	*+0.77*	*+3.09*	*+1.79*	*+2.34*	–

Simply scaling # shots is not necessarily the best

Insight 2: In many cases, EO > IO

"Let's think step by step." + exemplars from 32x random search > SoTA instruction optimization + random exemplars 😲

Aggregated

	Exemplar Selection (ES)					
	No ES	Random	Nearest	Diversity	R.S.	Mutation
No IO	60.30	66.91	66.09	66.74	71.16	72.92
APE	64.96	69.11	69.01	70.81	75.88	76.25
ProTeGi	68.13	70.81	70.01	69.25	75.90	77.29
PromptAgent	65.66	67.65	67.82	67.35	72.51	72.77
OPRO	63.04	68.50	68.33	67.57	73.02	73.06

max(...) = 70.81 < 71.16

Task-wise

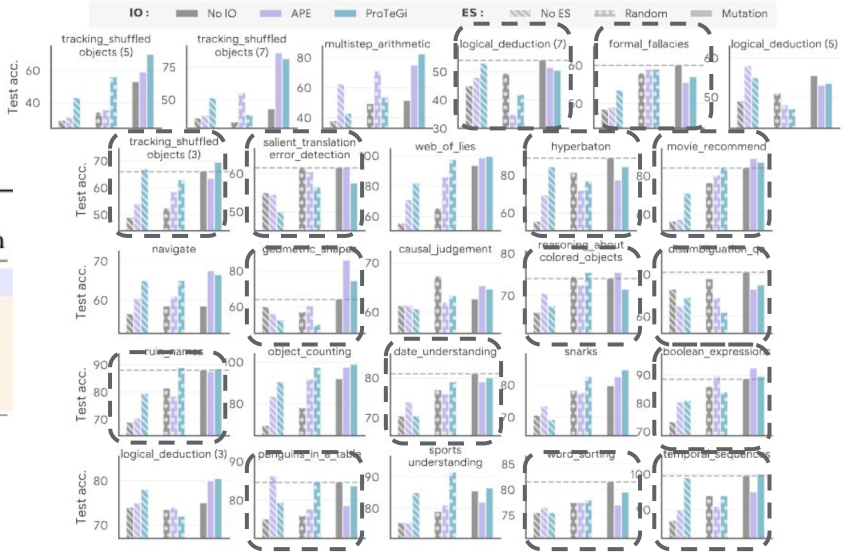
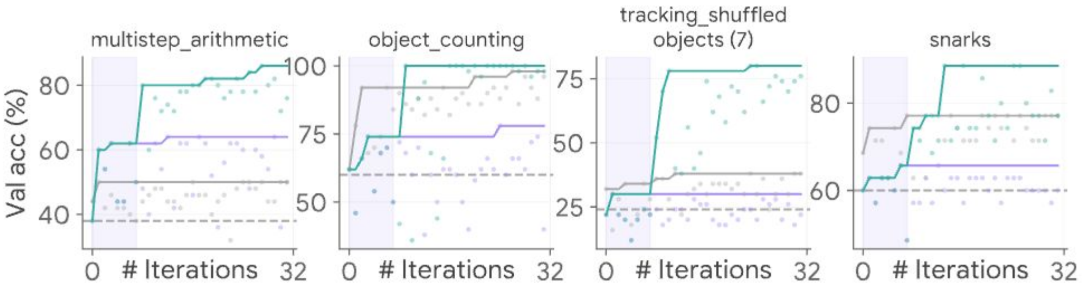


Figure 4: Task-specific BBH performance of selected IO-ES combinations with PaLM 2. Note that **1)** Proper ES almost uniformly improves performance and **2)** With appropriate exemplars, seed instructions **with no optimization** (third bar from the right) can often perform on par or better than SoTA IO but with standard random exemplars or no exemplars commonly used in the literature (first six bars in each figure). Refer to App. B.3 for visualizations with Gemini models.

Insight 3: Combining IO and EO

Combining IO and ES is greater than the sum of its parts *under similar computational budgets*.

Joint instruction and exemplar optimization also powers the Vertex AI Prompt Optimizer!



PaLM 2 (`text-bison-002`)

Eval. budget m	32					64
IO Budget m_{IO}	32	24	16	8	0	32
ES Budget m_{ES}	0	8	16	24	32	32
Avg. test acc. (%) ↑	70.81†	73.26	74.49	**76.14**	72.92	*76.25*
Avg. test rank ↓	4.50	3.63	3.44	**2.88**	3.60	*2.94*

Gemini 1.5 Flash

Eval. budget m	32					64
IO Budget m_{IO}	32	24	16	8	0	32
ES Budget m_{ES}	0	8	16	24	32	32
Avg. test acc. (%) ↑	83.25†	84.90	**85.17**	84.82	83.71	*85.04*
Avg. test rank ↓	4.04	3.65	3.29	**3.00**	3.58	*3.44*

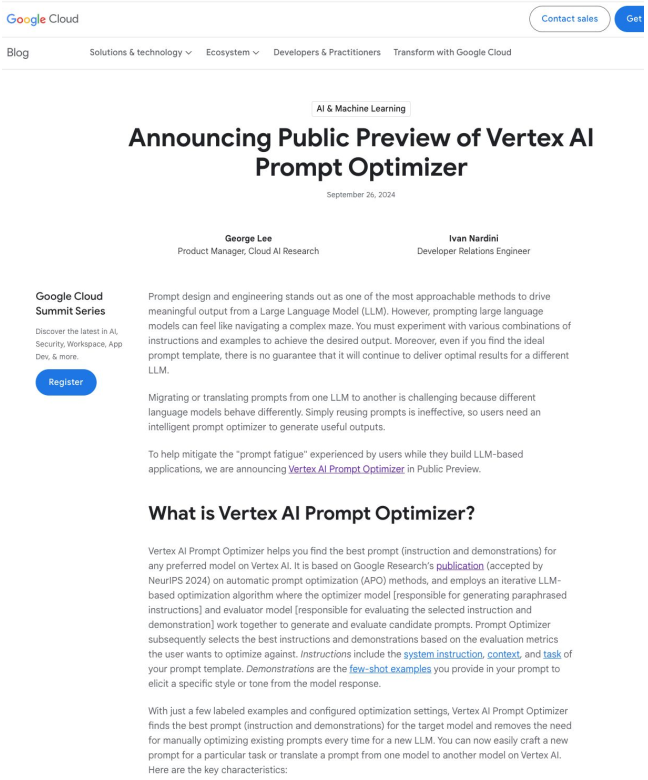
Similar performance, but green boxes are ~2x more expensive than the red boxes (optimal allocation of IO / ES)

Vertex AI Prompt Optimizer: Now Publicly Available

Performs optimization on **instructions** and **demonstrations** of any Vertex AI Model. **Currently available as a Public Preview product.**

Iterative optimization process. Key components:

- *Labeled data*: a small number of labeled data for **validation** and as the source for selection of few-shot **demonstrations**
- *Optimizer model*: An LLM used to propose modified instruction candidates
- *Evaluator model*: An LLM for evaluating the prompts (instructions + demonstrations) on a user-defined **evaluation**



Google Cloud | Contact sales | Get

Blog | Solutions & technology | Ecosystem | Developers & Practitioners | Transform with Google Cloud

AI & Machine Learning

Announcing Public Preview of Vertex AI Prompt Optimizer

September 26, 2024

George Lee
Product Manager, Cloud AI Research

Ivan Nardini
Developer Relations Engineer

Google Cloud Summit Series

Discover the latest in AI, Security, Workspace, App Dev, & more.

Register

Prompt design and engineering stands out as one of the most approachable methods to drive meaningful output from a Large Language Model (LLM). However, prompting large language models can feel like navigating a complex maze. You must experiment with various combinations of instructions and examples to achieve the desired output. Moreover, even if you find the ideal prompt template, there is no guarantee that it will continue to deliver optimal results for a different LLM.

Migrating or translating prompts from one LLM to another is challenging because different language models behave differently. Simply reusing prompts is ineffective, so users need an intelligent prompt optimizer to generate useful outputs.

To help mitigate the "prompt fatigue" experienced by users while they build LLM-based applications, we are announcing Vertex AI Prompt Optimizer in Public Preview.

What is Vertex AI Prompt Optimizer?

Vertex AI Prompt Optimizer helps you find the best prompt (instruction and demonstrations) for any preferred model on Vertex AI. It is based on Google Research's publication (accepted by NeurIPS 2024) on automatic prompt optimization (APO) methods, and employs an iterative LLM-based optimization algorithm where the optimizer model [responsible for generating paraphrased instructions] and evaluator model [responsible for evaluating the selected instruction and demonstration] work together to generate and evaluate candidate prompts. Prompt Optimizer subsequently selects the best instructions and demonstrations based on the evaluation metrics the user wants to optimize against. *Instructions* include the system instruction, context, and task of your prompt template. *Demonstrations* are the few-shot examples you provide in your prompt to elicit a specific style or tone from the model response.

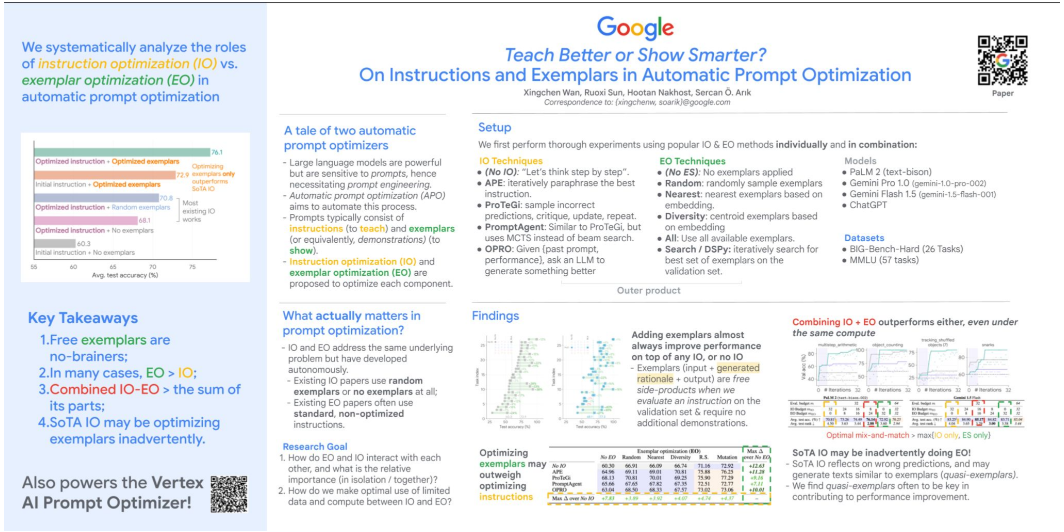
With just a few labeled examples and configured optimization settings, Vertex AI Prompt Optimizer finds the best prompt (instruction and demonstrations) for the target model and removes the need for manually optimizing existing prompts every time for a new LLM. You can now easily craft a new prompt for a particular task or translate a prompt from one model to another model on Vertex AI. Here are the key characteristics:

Link to our Google Cloud Blog

Conclusion

Systematically evaluate instructions and exemplars in APO

1. Intelligently incorporating exemplars generated by the target model itself **significantly** and **consistently** improves performance
2. The performance gains realized by **choosing appropriate exemplars** can eclipse the **improvements brought by SoTA instruction optimization.**
3. Optimally mixing-and-matching IO and ES is greater than the sum of its parts
4. SoTA IO might already be itself implicitly relying on exemplars



Our poster session

Date and Time: Fri 13 Dec (11 a.m. PST - 2 p.m.) PST

Venue: West Ballroom A-D #7000