metaTextGrad: Automatically optimizing language model optimizers

Guowei Xu, Mert Yuksekgonul,

Carlos Guestrin, James Zou

NEURAL INFORMATION PROCESSING SYSTEMS

TL, DR: A **meta-optimizer** — for optimizing **LLM optimizers**

Outline

- Introduction
- Motivation
- Theoretical Insight
- Method
- Experiment
- Summary

Introduction

Intro: Prompt Optimization Engines in LLMs

① TextGrad

[1] Yuksekgonul, M., Bianchi, F., Boen, J. *et al.* Optimizing generative AI by backpropagating language model feedback. *Nature* 639, 609–616 (2025).

❶ Analogy in abstractions

	Math	PyTorch	TextGrad
Input	x	`Tensor(image)`	`tg.Variable(article)`
Model	$\hat{y} = f_\theta(x)$	`ResNet50()`	`tg.BlackboxLLM("You are a summarizer.")`
Loss	$L(y, \hat{y}) = \sum_i y_i \log(\hat{y}_i)$	`CrossEntropyLoss()`	`tg.TextLoss("Rate the summary.")`
Optimizer	$\mathrm{GD}(\theta, \frac{\partial L}{\partial \theta}) = \theta - \frac{\partial L}{\partial \theta}$	`SGD(list(model.parameters()))`	`tg.TGD(list(model.parameters()))`

❷ Automatic differentiation

PyTorch and TextGrad share the same syntax for backpropagation and optimization.

Forward pass	Backward pass	Updating variable
`loss = loss_fn(model(input))`	`loss.backward()`	`optimizer.step()`

② DSPy

DSPy

DSPy: *Programming*—not prompting—Foundation Models

[2] Khattab, O., Singhvi, A., Maheshwari, P., Zhang, et al. "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines." *arXiv* 2310.03714 (2023).

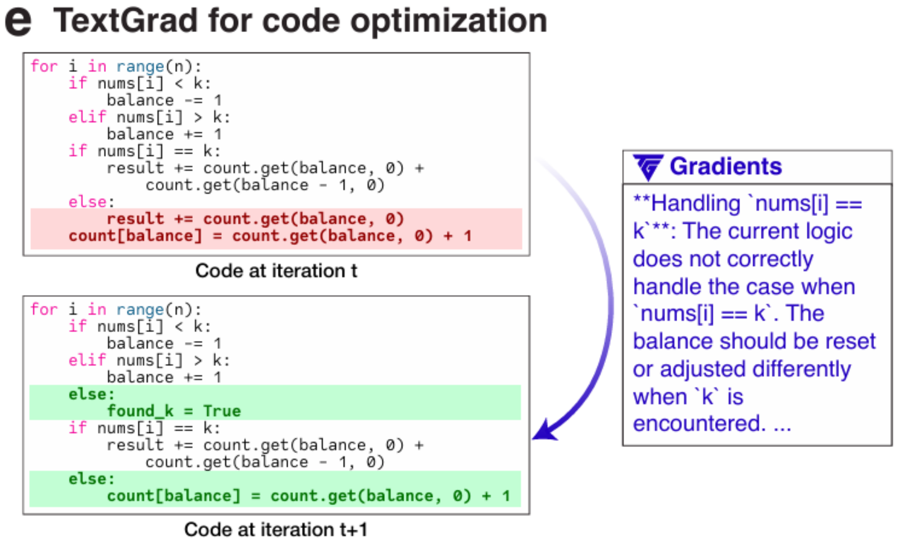
③ LangChain

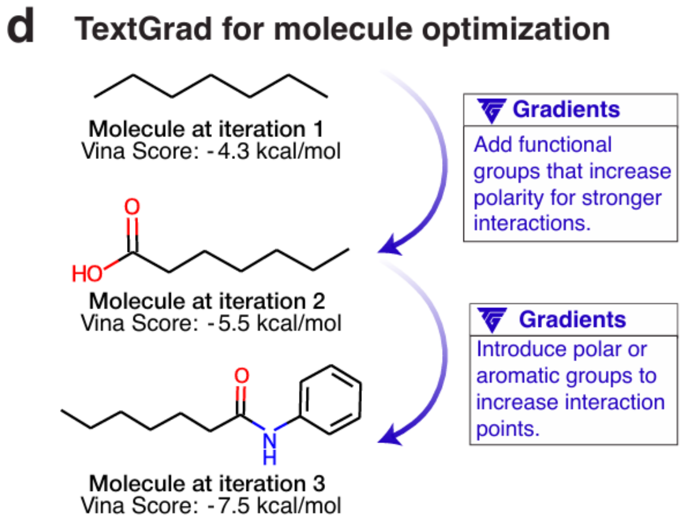
LangChain

release langchain-core==0.3.60 | CI passing | license MIT | downloads 51M/month
stars 108k | open issues 264 | Dev Containers Open | Open in GitHub Codespaces
Follow @LangChainAI | codspeed

[3] LangChain AI. *LangChain: The platform for reliable agents.* GitHub repository. 2025.

Intro: Problems in Prompt Optimization





(1) Optimizers are too broad and inefficient

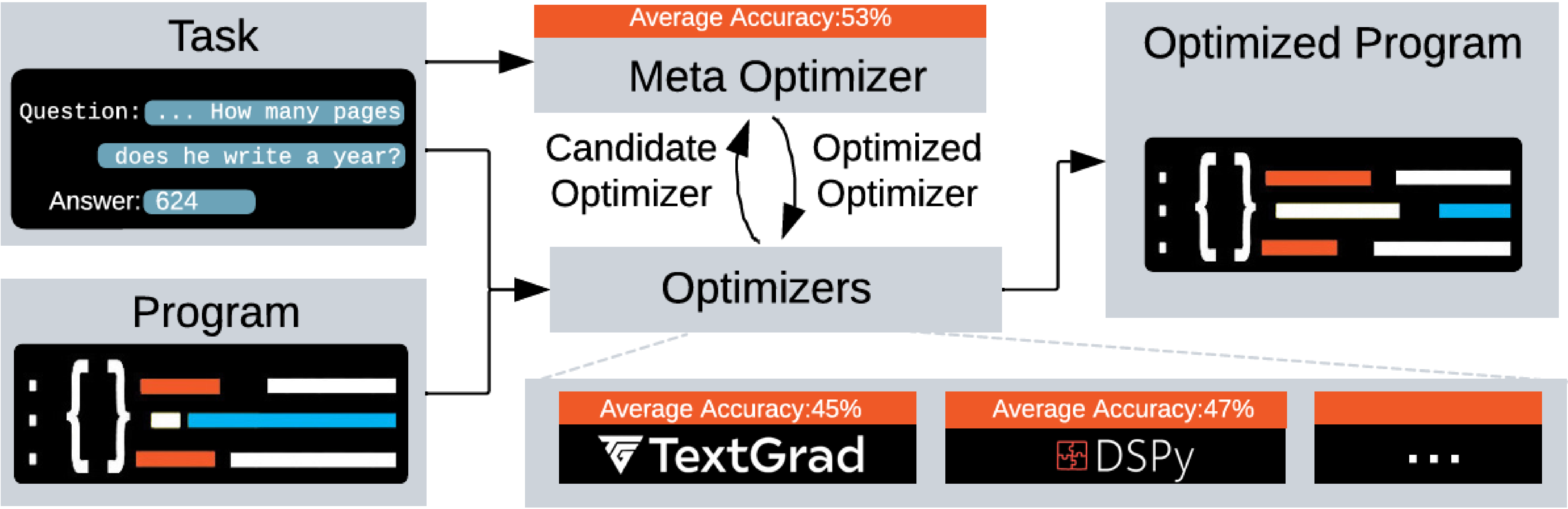
"We need to optimize both the code and the proteins."

(2) Different optimizers each have strengths, but are hard to combine

Is it possible to combine the advantages of different optimizers?

Intro: Our Goal

design a **meta-optimizer** that improves existing large-model optimizers and better **aligns them to specific tasks**



Motivation

Existing LLM optimizers

Existing LLM optimizers like **TextGrad** and **DSPy** are manually designed to perform well across many different tasks. Their prompts are highly general.

TextGrad TGD optimizer Prompt

You are part of an optimization system that improves text (i.e., variable). You will be asked to creatively and critically improve prompts, solutions to problems, code, or any other text-based variable.

Problem: Noisy Feedback

However, this generality can sometimes **limit their effectiveness** on specific tasks. When the optimizer only receives **noisy feedback**, such as overly detailed textual suggestions:

Feedback Received by the TGD optimizer

To improve the prompt for the executer and enhance the objective function, consider the following feedback: 1. **Explicit Criteria Definition**: The prompt should explicitly instruct the executer to define the criteria for optical activity at the beginning of the response. This can prevent ambiguity and ensure that the executer uses the correct scientific principles. For example, the prompt could include a directive to "List the criteria for optical activity before analyzing each compound." 2. **Data Verification Directive**: Incorporate a step in the prompt that requires the executer to verify the input data against reliable sources. This could be phrased as "Cross-check the properties of each compound with a trusted chemical database before proceeding with the analysis." 3. **Structured Logical Reasoning**: Encourage a structured approach to reasoning by breaking down the analysis into distinct steps. The prompt could suggest a format like "For each compound, first identify chiral centers, then assess symmetry, and finally determine optical activity".

It may **incorporate irrelevant information** and fail to improve the model effectively.

Solution: Task-aligned Optimizer

In contrast, a **task-aligned optimizer** can be explicitly aligned with the target task distribution.
For instance, an optimizer optimized for *Dyck Languages (bracket matching)* might include guidance such as focusing on *proper nesting* and *LIFO order*:

A Task-specific Optimizer Prompt

You are part of an optimization system specialized in improving prompts for bracket matching and sequence completion tasks. Your role is to enhance prompts that help solve Dyck language problems, which involve proper nesting and closure of different types of brackets (, <>, ()). When improving prompts, focus on these critical aspects: (1) maintaining accurate bracket pair matching, (2) preserving the LIFO (Last In First Out) order of nested structures, (3) handling multiple bracket types simultaneously, and (4) ensuring complete closure of all open brackets. You should critically analyze how the prompts can better guide the model to track open brackets, maintain proper nesting order, and systematically complete sequences. Consider incorporating pattern recognition strategies and explicit validation rules in the improved prompts. Your improvements should lead to more reliable and accurate bracket sequence completions.

Find a good initial value for the optimization process.

Solution: Task-aligned Optimizer

In contrast, a **task-aligned optimizer** can be explicitly aligned with the target task distribution. **(Find a good initial value for the optimization process.)**

Such alignment helps the optimizer produce better programs even when the feedback remains noisy.

Therefore, rather than designing new optimizers manually for every task, **metaTextGrad** aims to *meta-learn how to adapt optimizers automatically* and create optimizers that are both stronger and task-aligned.

Theoretical Insight

Theoretical Insight

Let R be the loss function, S_1 the training dataset, and S_2 the test dataset (drawn from the same distribution).
Let $\hat{\theta}$ be the optimizer after meta-training, and θ^* be the optimizer that is optimal under a given distribution.

Then we can prove that with probability at least $1 - \delta$:

$$R_{S_2}(\widehat{\theta}) \leq R(\theta^*) + \sqrt{\frac{2\log(6/\delta)}{n}} + \sqrt{\frac{\log(6/\delta)}{2m}}.$$

That is, an optimizer adapted through meta-optimization to a specific task can **effectively optimize data drawn from the same distribution**.

On the other hand, for an optimizer that has not undergone meta-optimization, **there will always exist some tasks for which its optimization performance is poor**.

This demonstrates the **necessity** of conducting meta optimization.

Method

Meta Optimizer: Inner Loop

$$\Phi^* = \arg\max_{\Phi} \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \mu(\boxed{\Phi}(x), y)$$

Algorithm 1 Inner Loop: Optimize Φ with Optimizer M

Input: Optimizer M, Initial Program Φ, Max Iterations I
Input: Training Data $\mathcal{D}$, Validation Data $\mathcal{D}_{val}$, Metric μ
Output: Φ^*, i.e., the optimized version of Φ
M.Initialize($\mathcal{D}, \Phi$)
for $k \leftarrow 1$ **to** I **do**
    $\Phi_k \leftarrow M$.Propose()
    $\sigma \leftarrow \frac{1}{|\mathcal{D}_{val}|} \sum_{(x,y)\in\mathcal{D}_{val}} \mu(\Phi_k(x), y)$
    M.Update(Φ_k, σ)
end for
$(\Phi^*, \sigma^*) \leftarrow M$.ExtractOptimizedProgram()
return (Φ^*, σ^*)

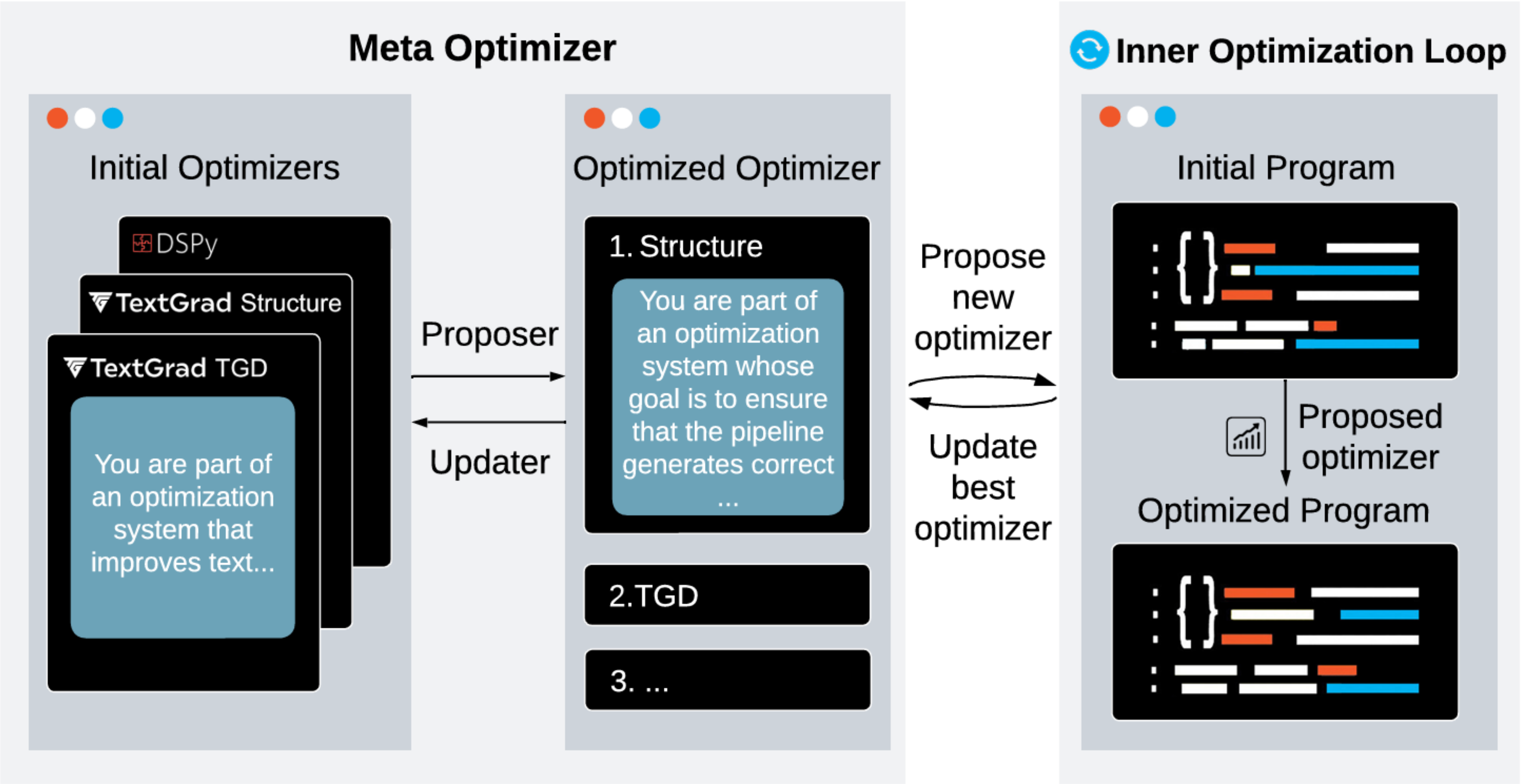
Meta Optimizer: Outer Loop

$$M^* = \arg\max_{M} \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \mu(\boxed{M.}\text{optimize}(\mathcal{D}, \Phi)(x), y).$$

Algorithm 2 Meta-Optimization of Optimizers

Input: Meta-Optimizer $\widehat{M}$
Input: Max Meta-Iterations J, Max Inner-Iterations I
Input: Initial optimizers $\{M^{(1)}, M^{(2)}, \ldots, M^{(r)}\}$
Input: Training Data $\mathcal{D}$, Validation Data $\mathcal{D}_{val}$
Input: Metric μ, Initial Program Φ
Output: Optimized optimizer M^*
$\widehat{M}$.Initialize$(\mathcal{D}, \{M^{(i)}\}_{i=1}^{r})$
for $j \leftarrow 1$ **to** J **do**
    $M_j \leftarrow \widehat{M}$.Propose()
    $(\Phi_j^*, \sigma_j) \leftarrow$ InnerLoop$\big(M_j, \Phi, I, \mathcal{D}, \mathcal{D}_{val}, \mu\big)$
    $\widehat{M}$.Update$\big(M_j, \sigma_j\big)$
end for
$M^* \leftarrow \widehat{M}$.ExtractOptimizedOptimizer()
return M^*

Meta Optimizer: Overview



metaTextGrad consists of a **meta-prompt optimizer** and a **meta-structure optimizer**. Given a set of optimizers, the meta optimizer performs optimization in two steps. First, it optimizes each optimizer individually so that they align better with the specific task. Then, it combines these optimized optimizers into a new optimizer.

Experiment

Overview

Method	Word Sorting		Dyck Languages		GPQA Diamond		Abstract Algebra		Average	
	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
Vanilla prompting methods										
Zero-shot CoT	0.46	0.55	0.06	0.05	0.32	0.34	0.74	0.70	0.40	0.41
8-shot CoT	0.50	0.52	0.14	0.19	0.32	0.35	0.65	0.71	0.40	0.44
Self-consistency (8)	0.47	0.52	0.10	0.12	0.40	**0.42**	0.76	0.70	0.43	0.44
Best of N (8)	0.48	0.52	0.14	0.17	0.37	0.40	0.77	0.74	0.44	0.46
TextGrad optimizers										
TGD Optimizer	0.54	0.55	0.10	0.10	0.34	0.35	0.76	0.71	0.44	0.43
ADAS-TG	0.58	0.58	0.21	0.16	0.36	0.37	0.75	0.70	0.48	0.45
DSPy optimizers										
Zero-shot MIPROv2	0.57	0.55	0.19	0.16	0.43	0.38	0.76	**0.77**	0.49	0.47
8-shot MIPROv2	0.52	0.57	0.33	0.26	0.37	0.34	0.74	0.65	0.49	0.46
Meta-optimized optimizers										
`metaTextGrad`	**0.60**	**0.65**	**0.42**	**0.37**	**0.45**	0.40	**0.78**	0.71	**0.56**	**0.53**

Across various math-reasoning datasets, **metaTextGrad** shows a marked improvement in performance over baselines.

Cost Analysis

To control cost, we want to use a regular model to perform the computationally expensive program execution, a moderately stronger model as the optimizer, and the best model as the meta-optimizer.

The results show that **metaTextGrad** exhibits significant differences in token usage across different reasoning levels, indicating that **this hierarchical design is efficient**.

Level	Tokens
Program level	$\sim$ 400k
Optimizer level	$\sim$ 100k
Meta-optimizer level	$\sim$ 2.5k

Table 2: Token analysis on Abstract Algebra.

Model	Performance	Cost
0-shot CoT (4o-mini)	0.05	0.14$
Ours (4o-mini)	0.37	0.44$
0-shot CoT (4o)	0.18	0.52$

Table 3: Cost analysis on Dyck Languages.

With the help of **metaTextGrad**, GPT-4o-mini even performs **better** than GPT-4o on BBH Dyck Languages.

Generalizability

Method (Claude 3 Haiku)	Dyck Languages	
	Val	Test
Zero-shot CoT	0.07	0.10
TextGrad optimizers		
TGD Optimizer	0.10	0.04
ADAS-TG	0.35	0.34
Optimizers optimized on GPT-4o-mini		
`metaTextGrad`	0.32	0.35

Table 4: Transferability of the optimized optimizer across models.

Method	Abstract Algebra	
	Val	Test
Zero-shot CoT	0.74	0.70
TextGrad optimizers		
TGD Optimizer	0.76	0.71
ADAS-TG	0.75	0.70
Optimizers optimized on GPQA diamond		
`metaTextGrad`	0.78	0.77

Table 5: Transferability of the optimized optimizer across datasets.

The optimizers produced by **metaTextGrad** can effectively generalize across different **models and datasets**.

Open-Source Models & Harder Tasks

Method	Dyck Languages (Qwen models)		ARC-AGI (Challenging Benchmark)	
	Val	Test	Val	Test
Vanilla prompting methods				
Zero-shot CoT	0.27	0.27	0.27	0.23
8-shot CoT	0.37	0.40	0.03	0.00
Self-consistency (8)	0.31	0.32	0.30	0.23
Best of N (8)	0.39	0.41	0.27	0.20
TextGrad optimizers				
TGD Optimizer	0.69	0.68	0.33	0.33
ADAS-TG	0.32	0.34	0.28	0.26
DSPy optimizers				
Zero-shot MIPROv2	0.59	0.50	0.30	0.23
8-shot MIPROv2	0.57	0.51	0.33	0.03
Meta-optimized optimizers				
`metaTextGrad`	**0.82**	**0.77**	**0.37**	**0.40**

metaTextGrad remains effective on **open-source models and complex tasks**.

Ablation Study

Split	0-shot CoT	TGD	ADAS-TG	TGD (O)	ADAS-TG (O)	Struct (O)	metaTextGrad
Val	0.06	0.10	0.21	0.21	0.42	0.24	0.42
Test	0.05	0.10	0.16	0.24	0.37	0.16	0.37

Table 6: Analysis of the effectiveness of each meta optimizer on Dyck Languages. TGD (O), ADAS-TG (O), and Struct (O) respectively denote the TGD and ADAS-TG optimizers enhanced by the meta prompt optimizer, and the optimizers enhanced by the meta structure optimizer.

Each component of metaTextGrad can effectively improve optimization performance.

Summary

Summary of Contribution

- Identify the **necessity of improving existing large-model optimizers** and better aligning them with specific tasks.
- Provide both **theoretical and empirical insights** for performing meta-optimization.
- Design a **meta-prompt optimizer** and a **meta-structure optimizer** to optimize LLM optimizers.
- Demonstrate the **effectiveness** of our approach across different tasks.

metaTextGrad is open-source!

Source Code

https://github.com/zou-group/metatextgrad

Paper

https://arxiv.org/abs/2505.18524

Thank you!