TEXTREG: MITIGATING PROMPT DISTRIBUTIONAL OVERFITTING VIA REGULARIZED TEXT-SPACE OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are highly sensitive to the prompts used to specify task objectives and behavioral constraints. Many recent prompt optimization methods iteratively rewrite prompts using LLM-generated feedback, but the resulting prompts often become longer, accumulate narrow sample-specific rules, and generalize poorly beyond the training distribution. We study this failure mode as *prompt distributional overfitting* and argue that it reflects a lack of representation control in discrete text-space optimization. We formalize this view through *representational inefficiency*, a dual-factor measure that decomposes prompt inefficiency into capacity cost and scope narrowness, attributing distributional prompt overfitting to their coupled growth during optimization. We propose **`TextReg`**, a regularization framework that realizes a soft-penalty objective through regularized textual gradients, combining Dual-Evidence Gradient Purification, Semantic Edit Regularization, and Regularization-Guided Prompt Update. Across multiple reasoning benchmarks, **`TextReg`** substantially improves out-of-distribution (OOD) generalization, with accuracy gains of up to +11.8% over TextGrad and +16.5% over REVOLVE. The code is available at `https://anonymous.4open.science/r/textreg_code`.

1 INTRODUCTION

Large language models (LLMs) have exhibited strong performance on a wide variety of reasoning and generation tasks (Brown et al., 2020; Achiam et al., 2023; Team et al., 2023; Grattafiori et al., 2024), with the input prompt serving as the central interface that specifies task objectives, output formats, and behavioral constraints (Wang et al., 2026b;a). This sensitivity to prompting has motivated the development of *prompt optimization* (Pryzant et al., 2023; Wan et al., 2024), which aims to automatically refine prompts through data-driven structure discovery and feedback, thereby reliably eliciting desired behavior and reducing manual trial-and-error in prompt design. Recent approaches iteratively refine prompts using natural-language feedback from LLM evaluators (Pryzant et al., 2023; Yuksekgonul et al., 2024) and have achieved promising empirical improvements. Despite these gains, optimized prompts often suffer from poor generalization beyond the training data: as optimization proceeds, prompts may become longer, accumulate case-specific instructions, and become more sensitive to input variations (Zhuo et al., 2024), a failure mode known as *prompt distributional overfitting* (Khattak et al., 2023). This failure manifests along two coupled dimensions: prompts not only expand in length as new rules, exceptions, examples, and stylistic constraints are appended, but also narrow in scope as the appended content drifts toward fragmented sample-dependent patches rather than compact general principles. Existing prompt optimizers provide little control over either dimension.

We view a prompt as a structured representation of task knowledge, composed of behavioral rules expressed in natural language; an ideal prompt should encode broadly applicable principles in a compact form, rather than fit training examples through an expanding set of special-case instructions. We formulate this failure mode through the lens of *representational inefficiency*: from this perspective, unconstrained optimization violates this principle in two coupled ways. First, increasing prompt length raises the *capacity cost*—longer prompts consume more context budget and make it harder for the model to reliably locate and leverage relevant instructions, given that LLMs' effective context

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

window is limited Levy et al. (2024); Liu et al. (2024); Jin et al. (2026). Second, prompts can suffer from *rule narrowing*: they accumulate rules that reduce training loss but apply only to a narrow subset of inputs, functioning as ad hoc patches rather than reusable task principles. These two factors jointly lead to representational inefficiency, arising from the interaction between the capacity cost incurred by prompt length and the scope waste among its constituent rules. As they compound, an increasing fraction of the prompt's capacity is wasted on information that contributes little to generalization, suggesting that prompt distributional overfitting (Khattak et al., 2023) is fundamentally a failure of representation efficiency, rather than merely an artifact of optimization dynamics.

In classical machine learning, overfitting is commonly mitigated by regularization that controls model complexity. However, applying this perspective to prompt optimization is non-trivial: the optimization is non-differentiable, with feedback generated by LLM evaluators and updates produced by LLM rewriting.

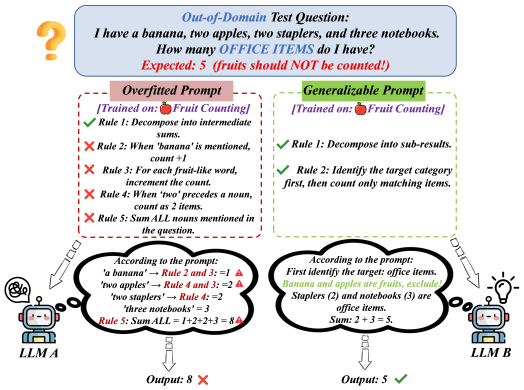


Figure 1: **Problem Illustration.** We illustrate prompt distributional overfitting in prompt optimization: **I)** conventional methods often produce long prompts saturated with narrow rules (left), which **degrade on OOD inputs**. **II)** Our goal is to instead yield compact prompts composed of broadly applicable rules (right), achieving **stronger OOD generalization**.

To address this challenge, we propose **TextReg**, a regularization framework that improves prompt optimization through a regularized textual-gradient view: prompt updates should follow both a task direction that improves empirical performance and a regularization direction that constrains the growth of representational inefficiency. **TextReg** realizes this view through three stages. *Dual-Evidence Gradient Purification* constructs a purified task gradient by combining local batch evidence and RuleBank recurrence evidence to distinguish case-specific patches from broadly applicable rules. *Semantic Edit Regularization* estimates how recent prompt edits change the capacity cost and scope waste, converting the observed degradation into a textual regularization gradient. *Regularization-Guided Prompt Update* then uses this gradient to steer prompt rewriting toward task-consistent edits that avoid unnecessary increases in representational inefficiency. Our contributions are threefold:

❶ **Formalizing Representational Inefficiency.** We formulate prompt distributional overfitting as a failure of representation efficiency in discrete text-space optimization, and introduce *representational inefficiency*—a dual-factor measure capturing the interaction between capacity cost from prompt length growth and scope waste from rule narrowing.

❷ **Regularized Text-Space Prompt Optimization.** We propose **TextReg**, a regularization framework that realizes a soft-penalty objective in discrete text space through three complementary stages—Dual-Evidence Gradient Purification, Semantic Edit Regularization, and Regularization-Guided Prompt Update. **TextReg** explicitly controls the growth of representational inefficiency during optimization, producing prompts that generalize beyond the training distribution.

❸ **Improved Out-of-Distribution Generalization.** Through extensive experiments across multiple reasoning benchmarks, we demonstrate that **TextReg** consistently outperforms existing prompt optimization methods on out-of-distribution (OOD) generalization across both datasets and test engines, validating that controlling representational inefficiency mitigates prompt distributional overfitting.

2 Related Work

Prompt optimization. Reasoning-oriented prompting techniques such as CoT (Wei et al., 2022), self-consistency (Wang et al., 2022), ReAct (Yao et al., 2022), PoT (Chen et al., 2022), and ToT (Yao et al., 2023) improve LLM reasoning by structuring the inference process. Automated prompt optimization discovers effective prompts algorithmically. Approaches range from discrete token- or edit-based methods such as AutoPrompt (Shin et al., 2020) and RLPrompt (Deng et al., 2022) to LLM-based methods such as APE (Zhou et al., 2022), EvoPrompt (Guo et al., 2023), Promptbreeder (Fernando et al., 2023), and DSPy (Khattab et al., 2023). The most relevant methods use feedback from LLM

evaluators to guide iterative rewriting. APO (Pryzant et al., 2023) treats LLM-generated critiques as discrete-space gradients, TextGrad (Yuksekgonul et al., 2024) formalizes textual differentiation through arbitrary computational graphs, and REVOLVE (Zhang et al., 2024) tracks response evolution across optimization steps. SIPDO (Yu et al., 2025) complements this direction by coupling prompt optimization with synthetic data generation. Although these methods have shown empirical gains, they primarily use task feedback to improve performance and do not explicitly regularize representational inefficiency during optimization.

Prompt robustness and overfitting. Recent work shows that prompt-based systems can be fragile under wording changes, semantic perturbations, adversarial paraphrases, and distribution shifts, as well as in long-context settings (Zhuo et al., 2024; Levy et al., 2024; Liu et al., 2024). Robust prompt optimization methods address related issues by accounting for distribution shifts (Li et al., 2023) or using sharpness-aware prompt evolution to improve paraphrase invariance (Wan et al., 2025). However, prompt overfitting remains a distinct failure mode in feedback-based optimization: prompts may improve on training feedback while accumulating narrow, sample-specific instructions that fail to transfer. APO (Pryzant et al., 2023) and TextGrad (Yuksekgonul et al., 2024) report this mismatch between training improvements and generalization. DLPO (Peng et al., 2025) mitigates overfitting through static simplification instructions, while REMO (Wu & Qu, 2025) uses external memory and meta-reflection. `TextReg` treats prompt overfitting as a problem of controlling prompt representations: we formalize it through representational inefficiency and regularize optimization trajectories based on observed structural changes.

Regularization for generalization. Regularization can improve generalization in machine learning by controlling effective model capacity. Examples include weight decay (Hoerl & Kennard, 1970; Krogh & Hertz, 1991), dropout (Wan et al., 2013; Srivastava et al., 2014), LASSO and elastic net (Tibshirani, 1996; Zou & Hastie, 2005), and early stopping (Caruana et al., 2000). Continuous prompt learning offers another way to limit the parameters optimized for a task: soft-prompt and prompt-tuning methods learn prompt embeddings and have been studied in few-shot settings (Li & Liang, 2021; Lester et al., 2021; Liu et al., 2022). `TextReg` regularizes natural-language prompts by jointly controlling capacity cost and scope narrowness, rather than constraining model parameters, continuous prompt embeddings, or prompt length alone.

3 Problem Formulation

3.1 Problem Setup and Notation

We consider a black-box language model $\mathcal{M}$ that maps an input x and a textual prompt p to an output $\mathcal{M}(p, x)$. Let $\mathcal{P}$ denote the space of valid prompts. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, we define the empirical task risk of a prompt as

$$\mathcal{L}_{\mathcal{D}}(p) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \ell\big(\mathcal{M}(p, x), y\big), \tag{1}$$

where $\ell(\cdot, \cdot)$ is a task-specific loss, possibly induced by an evaluator in generative settings. Standard prompt optimization seeks a prompt $p \in \mathcal{P}$ that minimizes $\mathcal{L}_{\mathcal{D}_{\text{train}}}(p)$.

3.2 Prompt Distributional Overfitting and Generalization

Optimizing Eq. (1) does not guarantee generalization beyond the training distribution. For a target distribution $\mathcal{D}'$ that differs from $\mathcal{D}_{\text{train}}$, we define the generalization gap as

$$\Delta(p; \mathcal{D}') = \mathcal{L}_{\mathcal{D}'}(p) - \mathcal{L}_{\mathcal{D}_{\text{train}}}(p). \tag{2}$$

Prompt Distributional Overfitting. Prompt distributional overfitting occurs when optimization reduces $\mathcal{L}_{\mathcal{D}_{\text{train}}}(p)$ while increasing $\Delta(p; \mathcal{D}')$. Empirically, this behavior is often accompanied by structural growth of the prompt, including increased length and the accumulation of narrow, sample-specific instructions.

Connection to classical overfitting. This phenomenon parallels classical overfitting in that additional representational capacity can fit increasingly narrow patterns at the expense of generalization. We

focus specifically on OOD generalization: an optimized prompt may continue to perform well on held-out samples from the source distribution while degrading on harder task variants or related-but-shifted tasks.

3.3 Representational Inefficiency

For analysis, we view a prompt p as encoding a set of semantically distinct behavioral directives $R(p) = \{r_1, \ldots, r_k\}$. A directive may describe a reasoning procedure, behavioral constraint, or other task-relevant instruction; this decomposition is analytical and does not require the prompt itself to be written as an explicit list of rules.

Generalization scope. Although optimization is performed on $\mathcal{D}_{\text{train}}$, we want the resulting prompt to remain useful over an input space $\mathcal{X}$ that preserves the underlying task while varying in surface form, scale, or difficulty. For a directive r_i, we define its generalization scope as

$$s(r_i) \triangleq \mathbb{E}_{x \sim \mathcal{X}} \left[\rho_{r_i}(x) \right] \in [0, 1], \tag{3}$$

where $\rho_{r_i}(x)$ measures the degree to which r_i is relevant to input x. Let $\bar{s}(p) = \frac{1}{|R(p)|} \sum_{r_i \in R(p)} s(r_i)$ denote the average scope of the prompt.

For example, in object counting, a general procedure that applies across inputs has high scope, whereas an instruction tied to a particular entity has low scope:

> ❶ **Broad rule**: *List each item in the input one by one before producing the final count.*
>
> ❷ **Narrow rule**: *When the input mentions tomatoes, count them as vegetables, not fruits.*

Inefficiency measure. We define the representational inefficiency of a prompt as

$$\mathcal{I}(p) = \underbrace{|p|_{\text{tok}}}_{\text{capacity cost } C(p)} \cdot \underbrace{\big(1 - \bar{s}(p)\big)}_{\text{scope narrowness } W(p)}. \tag{4}$$

Here, $C(p)$ captures the context capacity consumed by the prompt, while $W(p)$ captures the degree to which its directives fail to provide broadly applicable guidance. The multiplicative form emphasizes their interaction: additional prompt capacity is increasingly inefficient when it is devoted to narrowly applicable instructions. Thus, $\mathcal{I}(p)$ is high when a prompt is simultaneously long and dominated by low-scope directives.

3.4 Regularized Text-Space Prompt Optimization

Standard prompt optimization minimizes Eq. (1) without explicitly controlling how the prompt representation evolves. To formalize the desired trade-off, we consider the idealized regularized objective

$$\min_{p \in \mathcal{P}} \quad \mathcal{L}_{\mathcal{D}_{\text{train}}}(p) + \lambda \mathcal{I}(p), \tag{5}$$

where $\lambda \geq 0$ conceptually controls the trade-off between task performance and representational efficiency.

Importantly, Eq. (5) should be understood as a conceptual regularization objective that characterizes the desired trade-off between task performance and representational efficiency. In discrete text space, however, its two structural components have different operational properties: capacity cost is directly observable through prompt length, whereas scope narrowness is semantic and depends on how broadly individual directives apply across the task distribution. This distinction motivates treating representational inefficiency through complementary structural signals rather than as a single directly measurable scalar.

This formulation therefore provides a principled view of what should be controlled during prompt optimization, while leaving the concrete realization of these controls to the optimization procedure introduced next.

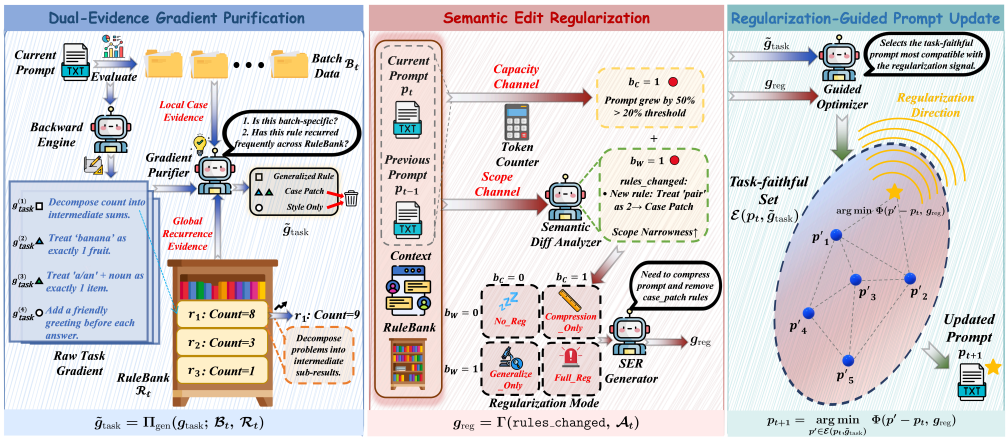


Figure 2: Overview of **TextReg**, which proceeds in three stages. (a) ***Left***: Dual-Evidence Gradient Purification filters the raw task gradient using local batch and RuleBank recurrence evidence, yielding $\tilde{g}_{\text{task}}$. (b) ***Middle***: Semantic Edit Regularization detects capacity and scope degradation and synthesizes the regularization signal g_{reg}. (c) ***Right***: Regularization-Guided Prompt Update rewrites p_t into p_{t+1} using both task and regularization guidance.

4 METHOD

4.1 OVERVIEW

Rather than numerically optimizing Eq. (5), **TextReg** operationalizes its task and regularization components through natural-language update signals. At optimization step t, it constructs a purified task signal $\tilde{g}_{\text{task}}$ and, when structural degradation is detected, a regularization signal g_{reg}. The next prompt is produced by an LLM-based rewrite operator

$$p_{t+1} = \mathcal{U}\big(p_t;\, \tilde{g}_{\text{task}}, g_{\text{reg}}\big), \tag{6}$$

where $\mathcal{U}$ integrates the two signals in natural language rather than through numerical gradient addition. We use "gradient" in the textual-optimization sense: these signals specify update directions in language and are not numerical derivatives or vectors.

As illustrated in Figure 2, **TextReg** proceeds through three complementary stages. **Stage 1 (Dual-Evidence Gradient Purification)** filters raw task gradients using local batch evidence and global RuleBank recurrence evidence. **Stage 2 (Semantic Edit Regularization)** monitors the capacity and scope channels of representational inefficiency and constructs g_{reg} when degradation is detected. **Stage 3 (Regularization-Guided Prompt Update)** coordinates task and regularization guidance while preserving task-faithful corrections.

The key design is to control representational inefficiency at both the *feedback* and *edit* levels. Stage 1 acts before rewriting by suppressing feedback likely to produce narrow or redundant instructions, whereas Stage 2 inspects the realized prompt transition and corrects capacity or scope degradation that still emerges during rewriting. Stage 3 then mediates these structural corrections without sacrificing task-relevant updates. This separation allows **TextReg** to regulate both how optimization signals are formed and how they are ultimately expressed in the prompt, directly targeting the two sources of inefficiency, $C(p)$ and $W(p)$.

4.2 DUAL-EVIDENCE GRADIENT PURIFICATION

Prompt optimization can introduce narrow corrections tied to particular training examples, increasing scope narrowness $W(p)$, as well as redundant or stylistic updates that increase capacity cost $C(p)$ without adding useful behavior. Dual-Evidence Gradient Purification filters such feedback before it enters the prompt rewrite.

For a task gradient g_{task} produced from mini-batch $\mathcal{B}_t \subset \mathcal{D}_{\text{train}}$, we write

$$\tilde{g}_{\text{task}} = \Pi_{\text{gen}}\big(g_{\text{task}};\, \mathcal{B}_t,\, \mathcal{R}_t\big), \tag{7}$$

where Π_{gen} is an LLM-realized conditional projection and $\mathcal{R}_t$ is RuleBank.

Local case evidence. The current mini-batch indicates whether a proposed correction is overly specific to the examples that produced it. Feedback closely tied to particular entities, quantities, surface patterns, or rare edge conditions is treated as negative evidence for generalization because it is likely to introduce a narrow case patch.

Global recurrence evidence. To complement local evidence, `TextReg` maintains

$$\mathcal{R}_t = \{(r, m_t(r)) : r \in \mathcal{U}_t\}, \tag{8}$$

where r is a canonical behavioral directive and $m_t(r)$ records how often it has matched previously accepted purified gradients. Accepted gradients are canonicalized and either matched to an existing entry or inserted as a new directive, allowing semantically equivalent corrections to accumulate recurrence evidence across surface forms.

The true scope $s(r)$ is not numerically estimated during optimization. Instead, $m_t(r)$ acts only as a *soft prior*: repeatedly recovered directives receive stronger support for generality than one-off corrections, without assuming a numerical mapping from recurrence to scope or using a hard threshold.

Purification decision. Combining local specificity and recurrence evidence, Π_{gen} classifies each gradient as a `GENERALIZED_RULE`, `CASE_PATCH`, or `STYLE_ONLY`. Generalized corrections are retained and rewritten into concise, broadly applicable instructions, while case-specific and purely stylistic feedback is discarded. When multiple gradients are produced, all retained instructions are concatenated into $\tilde{g}_{\text{task}}$.

Thus, Stage 1 controls representational inefficiency at its source: local evidence suppresses sample-specific patches, while recurrence evidence favors stable task-level structure.

4.3 Semantic Edit Regularization

Gradient purification filters feedback before rewriting, but the LLM optimizer may still introduce verbosity, redundancy, or new case-specific instructions. Semantic Edit Regularization (SER) therefore monitors the realized transition (p_{t-1}, p_t) for degradation in the two structural channels of $\mathcal{I}(p) = C(p)W(p)$.

Capacity diagnostic. Capacity change is directly observable from token length:

$$\rho_C(p_t) = \frac{C(p_t) - C(p_{t-1})}{C(p_{t-1})}, \qquad b_C(p_t) = \mathbb{I}\big[\rho_C(p_t) > \tau_C\big], \tag{9}$$

where τ_C prevents regularization from reacting to negligible length fluctuations.

Scope diagnostic. Scope narrowness is semantic and cannot be measured directly. SER therefore estimates only its direction of change using an LLM-based semantic diff analyzer:

$$M_\Delta\big(p_{t-1}, p_t, \mathcal{R}_t, \mathcal{G}_t\big) \to \big(\texttt{rules_changed}, \widehat{\text{sgn}}(\Delta W)\big), \tag{10}$$

where `rules_changed` records directives introduced, removed, or modified by the edit, and $\widehat{\text{sgn}}(\Delta W) \in \{+, -, 0\}$ indicates whether scope narrowness appears to have increased, decreased, or remained unchanged. RuleBank provides historical context for distinguishing recurring task-level directives from isolated case patches. The scope trigger is

$$b_W(p_t) = \mathbb{I}\left[\widehat{\text{sgn}}(\Delta W) = +\right]. \tag{11}$$

SER then forms the active regularization channels

$$\mathcal{A}_t = \{C : b_C(p_t) = 1\} \cup \{W : b_W(p_t) = 1\}, \tag{12}$$

corresponding to joint compression and generalization, compression only, generalization only, or no regularization.

The two diagnostics play complementary roles. The capacity channel provides a precise structural signal but cannot distinguish useful expansion from case-specific growth, whereas the scope channel captures semantic narrowing but is necessarily estimated rather than directly measured. Combining

Table 1: **Cross-dataset and cross-model generalization results.** We report out-of-domain test accuracy (%), averaged over four independent optimization seeds, and the relative improvement over TextGrad. The best and second-best results are highlighted with **bold** and underline, respectively.

Test Engine	Method	Logical Ded. 3obj		Tracking Shuf. 3obj		GSM8K	
		5obj	7obj	5obj	7obj	SVAMP	MultiArith
Qwen2 -7B-Instruct	CoT	51.6$_{\uparrow 0.3}$	47.4$_{\uparrow 0.8}$	42.0$_{\uparrow 3.8}$	33.9$_{\uparrow 0.8}$	89.8$_{\uparrow 0.5}$	94.7$_{\downarrow 1.1}$
	TextGrad	51.3	46.6	38.2	33.1	89.3	95.8
	REVOLVE	54.4$_{\uparrow 3.1}$	47.6$_{\uparrow 1.0}$	40.3$_{\uparrow 2.1}$	**40.2**$_{\uparrow 7.1}$	89.2$_{\downarrow 0.1}$	96.2$_{\uparrow 0.4}$
	TextReg	**55.3**$_{\uparrow 4.0}$	**47.8**$_{\uparrow 1.2}$	**45.4**$_{\uparrow 7.2}$	39.1$_{\uparrow 6.0}$	**90.1**$_{\uparrow 0.8}$	**96.5**$_{\uparrow 0.7}$
Llama-3.1 -8B-Instruct	CoT	59.7$_{\downarrow 0.1}$	50.4$_{\uparrow 0.0}$	58.8$_{\downarrow 10.9}$	48.5$_{\downarrow 18.2}$	**85.5**$_{\uparrow 0.7}$	96.0$_{\uparrow 0.0}$
	TextGrad	59.8	50.4	69.7	66.7	84.8	96.0
	REVOLVE	57.7$_{\downarrow 2.1}$	50.6$_{\uparrow 0.2}$	76.0$_{\uparrow 6.3}$	65.4$_{\downarrow 1.3}$	84.6$_{\downarrow 0.2}$	95.3$_{\downarrow 0.7}$
	TextReg	**61.1**$_{\uparrow 1.3}$	**51.0**$_{\uparrow 0.6}$	**79.7**$_{\uparrow 10.0}$	**76.6**$_{\uparrow 9.9}$	**85.5**$_{\uparrow 0.7}$	**96.7**$_{\uparrow 0.7}$
Llama-3 -8B-Instruct	CoT	52.6$_{\uparrow 10.5}$	48.8$_{\uparrow 7.7}$	35.6$_{\downarrow 10.3}$	36.5$_{\downarrow 1.5}$	83.7$_{\uparrow 0.5}$	94.7$_{\uparrow 0.0}$
	TextGrad	42.1	41.1	45.9	38.0	83.2	94.7
	REVOLVE	51.5$_{\uparrow 9.4}$	48.8$_{\uparrow 7.7}$	46.1$_{\uparrow 0.2}$	41.8$_{\uparrow 3.8}$	**83.9**$_{\uparrow 0.7}$	94.7$_{\uparrow 0.0}$
	TextReg	**53.3**$_{\uparrow 11.2}$	**52.0**$_{\uparrow 10.9}$	**54.3**$_{\uparrow 8.4}$	**48.3**$_{\uparrow 10.3}$	**83.9**$_{\uparrow 0.7}$	**94.9**$_{\uparrow 0.2}$
Phi-3.5 -Mini-Instruct	CoT	57.0$_{\uparrow 10.9}$	50.5$_{\uparrow 4.1}$	89.6$_{\uparrow 6.0}$	89.1$_{\downarrow 3.7}$	**90.4**$_{\uparrow 9.8}$	**97.0**$_{\uparrow 10.2}$
	TextGrad	46.1	46.4	83.6	**92.8**	80.6	86.8
	REVOLVE	43.4$_{\downarrow 2.7}$	38.7$_{\downarrow 7.7}$	86.8$_{\uparrow 3.2}$	87.7$_{\downarrow 5.1}$	87.0$_{\uparrow 6.4}$	95.1$_{\uparrow 8.3}$
	TextReg	**57.9**$_{\uparrow 11.8}$	**55.2**$_{\uparrow 8.8}$	**94.1**$_{\uparrow 10.5}$	92.2$_{\downarrow 0.6}$	88.5$_{\uparrow 7.9}$	94.7$_{\uparrow 7.9}$

them allows SER to distinguish different forms of representational drift and activate only the corresponding correction, instead of applying a uniform compression or generalization penalty to every update.

Textual regularization signal. An LLM-realized controller Γ converts the active channels and semantic changes into

$$g_{\text{reg}} = \Gamma\big(\texttt{rules_changed}, \mathcal{A}_t\big) \in \mathcal{G}_{\text{text}} \cup \{\varnothing\}. \tag{13}$$

Capacity degradation induces compression guidance, while scope degradation induces guidance to generalize or remove narrow case patches. When neither channel is active, $g_{\text{reg}} = \varnothing$.

Thus, capacity degradation is measured exactly, whereas scope degradation is estimated only directionally. SER does not numerically evaluate $W(p)$ or $\mathcal{I}(p)$, and intervenes only when the realized edit provides evidence of structural degradation.

4.4 Regularization-Guided Prompt Update

After the first two stages, **TextReg** obtains $\tilde{g}_{\text{task}}$ and, when needed, g_{reg}. The final stage uses regularization to shape *how* the task correction is realized while keeping task fidelity primary.

This separation between task fidelity and structural preference is important: regularization should shape the form of an update without suppressing a correction that is necessary for task performance. Stage 3 therefore treats $\tilde{g}_{\text{task}}$ as defining what must be changed, while g_{reg} constrains how that change should be expressed. This asymmetry preserves the optimization objective while discouraging the unnecessary prompt expansion and narrow rule accumulation associated with distributional overfitting.

This design also avoids forcing every beneficial update toward shorter or more generic wording. A task-relevant instruction may legitimately increase prompt length or introduce a specialized distinction, and **TextReg** does not suppress such changes solely because they affect $C(p)$ or $W(p)$. Regularization instead acts as a preference over task-faithful rewrites, intervening only when a structurally more efficient realization is available.

Let $\mathcal{E}(p_t, \tilde{g}_{\text{task}})$ denote the *task-faithful set*: conceptually, prompt rewrites that faithfully implement $\tilde{g}_{\text{task}}$. Within this set, regularization favors an edit most compatible with g_{reg}:

$$p_{t+1} = \arg \min_{p' \in \mathcal{E}(p_t, \tilde{g}_{\text{task}})} \Phi\big(p' - p_t,\, g_{\text{reg}}\big), \tag{14}$$

where Φ denotes incompatibility between the semantic edit and the active regularization guidance, and $p' - p_t$ denotes the semantic edit rather than literal vector subtraction.

Equation (14) is a conceptual characterization of the rewrite preference implemented by $\mathcal{U}$, not an explicitly enumerated numerical optimization: `TextReg` does not construct $\mathcal{E}$ or evaluate Φ as a scalar. Instead, the optimizer LLM realizes this preference directly in natural language.

This task-dominance fallback prevents regularization from becoming a hard structural constraint. Together, the three stages control which feedback enters optimization, detect structural drift after rewriting, and coordinate task improvement with structural guidance, thereby operationalizing Eq. (5) without directly evaluating $\mathcal{I}(p)$ or its gradients.

5 EXPERIMENTS

We assess `TextReg` along three dimensions: **Q1**: does it outperform existing prompt optimization methods on out-of-distribution generalization across datasets and test engines (Section 5.2)? **Q2**: how does each component contribute to the overall performance (Section 5.3)? **Q3**: how robust is it when individual components are degraded or replaced with weaker variants (Section 5.4)?

5.1 EXPERIMENTAL SETUP

Tasks and Datasets. We evaluate `TextReg` on six datasets from the Big Bench Hard benchmark (Suzgun et al., 2023; Srivastava et al., 2023)—**Logical Deduction** (Three / Five / Seven Objects) and **Tracking Shuffled Objects** (Three / Five / Seven Objects), as well as three arithmetic datasets: **GSM8K** (Cobbe et al., 2021), **SVAMP** (Patel et al., 2021), and **MultiArith** (Roy & Roth, 2015; Koncel-Kedziorski et al., 2016). Prompts are optimized on **Logical Deduction (Three Objects)**, **Tracking Shuffled Objects (Three Objects)**, and **GSM8K**, and evaluated for cross-dataset generalization on the remaining datasets: the harder variants **Logical Deduction** (Five / Seven Objects) and **Tracking Shuffled Objects** (Five / Seven Objects) for the BBH tasks, and **SVAMP** and **MultiArith** for the arithmetic tasks. Our primary metric is Accuracy (Acc), measured by exact match string on the final answer (Yuksekgonul et al., 2024). Further datasets and implementation details are in Appendix A.

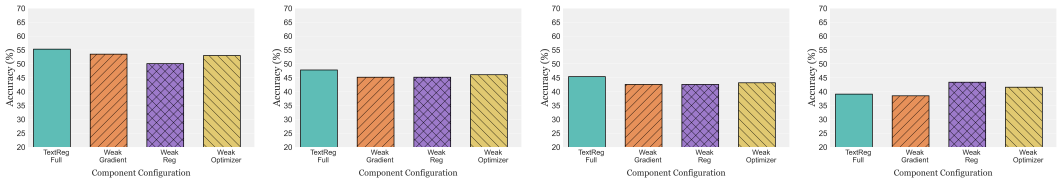
LLM Backends. Our experiments are conducted on four open-source LLM backends as test engines: Qwen2-7B-Instruct (Yang et al., 2024a), Phi-3.5-Mini-Instruct (Abdin et al., 2024), Llama-3-8B-Instruct, and Llama-3.1-8B-Instruct (Dubey et al., 2024). For prompt optimization, we use Qwen2.5-7B-Instruct (Yang et al., 2024b) as the forward engine that executes prompts on training samples, and GPT-4o (OpenAI, 2025) as the shared backward engine that performs all LLM-driven optimization operations. This shared configuration applies uniformly to `TextReg` and all baselines, to ensure a fair and controlled comparison. The optimized prompts are then evaluated on the four test engines above.

Counterparts. We compare `TextReg` against three baselines: Zero-shot Chain-of-Thought (CoT) (Kojima et al., 2022; Wei et al., 2022), TextGrad (Yuksekgonul et al., 2024), and REVOLVE (Zhang et al., 2024).

Evaluation Protocol. Every entry in Table 1 for TextGrad, REVOLVE, and `TextReg` is the mean over four optimization runs with independent seeds: each run optimizes its own prompt, and all resulting prompts are evaluated on the same test set of each dataset with greedy decoding. CoT uses the same fixed prompt evaluated under the same protocol.

5.2 MAIN RESULTS

To answer **Q1**, Table 1 reports cross-dataset and cross-model generalization results, where prompts optimized on three source tasks are evaluated out-of-distribution across four downstream test engines. `TextReg` consistently achieves the best or second-best accuracy on nearly all (test engine, dataset) pairs, outperforming both TextGrad and REVOLVE in the large majority of cells. The advantage is most pronounced on harder variants of the source task: on Tracking Shuffled Objects, `TextReg` improves over TextGrad by **+10.0** (5obj) and **+9.9** (7obj) on Llama-3.1-8B-Instruct, and by **+8.4** (5obj) and **+10.3** (7obj) on Llama-3-8B-Instruct. Notably, baseline methods often degrade out-of-distribution accuracy below that of the unoptimized CoT prompt—on Phi-3.5-Mini-Instruct, REVOLVE underperforms CoT on all six datasets—directly exposing the prompt overfitting problem that motivates our work, while `TextReg` preserves and extends the generalization benefits of CoT. This



(a) Logical Ded. 5 obj (b) Logical Ded. 7 obj (c) Tracking Shuf. 5obj (d) Tracking Shuf. 7obj

Figure 4: Resilience analysis of **TextReg** under role-wise engine degradation, where each of the three LLM-driven roles in the optimization pipeline is replaced one at a time with a weaker Qwen2.5-7B-Instruct model. For an in-depth analysis, please refer to Section 5.4.

robustness holds across heterogeneous test engines spanning different architectures and instruction-tuning recipes, confirming that mitigating prompt overfitting is a model-agnostic property of the optimization procedure.

5.3 Ablation Study

To address **Q2**, we disable each of **TextReg**'s three components in turn (Figure 3). Each bar reports mean out-of-distribution accuracy across four OOD tasks—Tracking Shuffled Objects (5/7 obj) and Logical Deduction (5/7 obj)—using prompts optimized on the corresponding 3-object source tasks. Full **TextReg** achieves the highest accuracy on all four test engines, and removing any of **Dual-Evidence Gradient Purification**, **Semantic Edit Regularization**, or **Regularization-Guided Update** produces a clear and consistent drop. This confirms that the three components are jointly necessary: source-level gradient filtering, post-edit inefficiency diagnosis, and regularization-guided rewriting each contribute non-redundantly to mitigating prompt distributional overfitting.

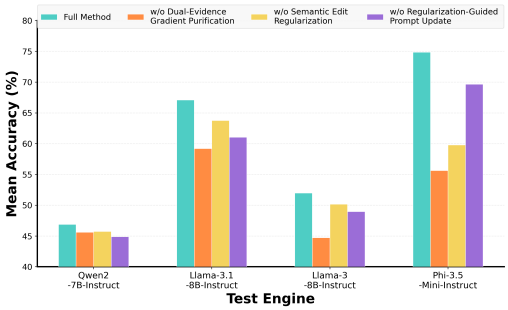


Figure 3: Ablation study of the 3 core components of **TextReg**: Gradient Purification, Semantic Edit Regularization, and Regularization-Guided Update. Each bar reports mean out-of-distribution accuracy (%) across four OOD tasks (Tracking Shuffled Objects 5/7 obj, Logical Deduction 5/7 obj) on each test engine. Section 5.3 presents analyses.

5.4 Resilience Study

To probe **TextReg**'s resilience (**Q3**), we replace one of three LLM-driven roles at a time with a substantially weaker LLM: **Gradient** (textual feedback, gradient purification, RuleBank rule extraction), **Regularization** (semantic edit analysis M_Δ, regularization gradient synthesis), or **Optimizer** (prompt rewriting from both signals). Starting from the **All-Strong** baseline where all three roles use GPT-4o, we downgrade each role in turn to Qwen2.5-7B-Instruct. As shown in Figure 4, **TextReg** retains strong performance under all three weakenings, with only minor accuracy drops; on the hardest variant Tracking Shuffled Objects (7 obj), **Weak Regularization** and **Weak Optimizer** even surpass **All-Strong**, indicating that **TextReg**'s regularization signal is structural rather than capability-bound and that parts of the optimization loop can be served by lightweight models.

6 Conclusion

We frame prompt distributional overfitting as a failure of representational efficiency: optimized prompts may reduce training loss by expanding in length and accumulating narrow, sample-specific rules, hurting OOD generalization. We formalize this through *representational inefficiency*, the interaction between capacity cost and scope narrowness, and propose **TextReg**, which controls its growth through Dual-Evidence Gradient Purification, Semantic Edit Regularization, and Regularization-Guided Prompt Update. **TextReg** improves OOD generalization over existing methods across multiple reasoning benchmarks. Our evaluation focuses primarily on single-turn reasoning with well-defined behavioral rules. Appendix F provides preliminary results on open-ended generation, broader generation tasks, multi-turn prompts, and agent instructions remain directions for future work. In these settings, rule structure can be less explicit and capacity cost may be shared across turns.

AI USE STATEMENT

We used generative AI tools to polish language, organize the manuscript, obtain limited feedback on the research methodology and experimental design, develop code, and execute experimental workflows. LLMs also serve as the backends of `TextReg` and the baselines. In the default configuration, GPT-4o performs all LLM-driven optimization operations, including textual gradient generation, gradient purification, semantic edit analysis, regularization synthesis, and prompt rewriting. Qwen2.5-7B-Instruct executes prompts on training samples during optimization. The test engines include Qwen2-7B-Instruct, Phi-3.5-Mini-Instruct, Llama-3-8B-Instruct, Llama-3.1-8B-Instruct, and Qwen2.5-32B-Instruct. Qwen2.5-7B-Instruct replaces GPT-4o in the resilience studies and serves as both the forward and test engine for the additional tasks in Appendix F. GPT-4o judges outputs for the open-ended generation task in that appendix. In these settings, LLMs are part of the experimental methodology. The authors developed and finalized the conceptual framework, hypotheses, experimental design, methodological choices, and interpretation of the results. All AI-assisted manuscript content and code were reviewed and verified by the authors, and AI-assisted implementations were tested for correctness by at least two authors. The authors assume full responsibility for the final content, claims, and artifacts of this work.

ETHICS STATEMENT

We adhere to the ICLR Code of Ethics. This study involves no human subjects or private user data. All tasks are drawn from publicly available benchmarks: the Logical Deduction and Tracking Shuffled Objects tasks from Big-Bench Hard, GSM8K, SVAMP, and MultiArith, with GPQA-main and CommonGen included in the appendix. `TextReg` operates entirely in textual prompt space and modifies neither model parameters nor training data. Its outputs remain subject to the harmful-content risks of the underlying models. Because optimized prompts are generated by LLMs, they may contain incorrect or overly specific instructions. `TextReg` is designed to limit overly specific instructions by filtering case-specific feedback, accepting updates under a validation-based criterion, and maintaining compact prompts that are easier to audit. These mechanisms do not verify the correctness or safety of every instruction, so optimized prompts should be reviewed before deployment. Our findings are based on controlled comparisons under matched experimental conditions.

REPRODUCIBILITY STATEMENT

The anonymous repository at `https://anonymous.4open.science/r/textreg_code` contains the code, prompt templates, initial prompts, data splits, and configuration files needed to reproduce the results. Appendix A.1 documents the main benchmarks and splits; Appendices A.2 and B describe the baselines. Appendix A.3 provides backends, decoding settings, hyperparameters, and hardware; Appendix A.4 gives the initial prompts; and Appendix G lists the complete `TextReg` prompt templates. We access GPT-4o through a hosted API and serve all open-weight models locally with vLLM on NVIDIA A100 80GB GPUs. The main source tasks follow TextGrad's 50/100/100 train/validation/test splits, and OOD evaluation uses the full released test splits. Unless otherwise stated, optimization results are averaged over four independent seeds (Section 5.1). The repository will be made public upon acceptance.

REFERENCES

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. *arXiv:2404.14219*, 2:6, 2024.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Rich Caruana, Steve Lawrence, and C Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, 13, 2000.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3369–3391, 2022.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Yiqiao Jin, Kartik Sharma, Vineeth Rakesh, Yingtong Dou, Menghai Pan, Mahashweta Das, and Srijan Kumar. Sara: Selective and adaptive retrieval-augmented generation with context compression. In *ACL*, 2026.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Muhammad Uzair Khattak, Syed Talal Wasim, Muzammal Naseer, Salman Khan, Ming-Hsuan Yang, and Fahad Shahbaz Khan. Self-regulating prompts: Foundational model adaptation without forgetting. In *ICCV*, pp. 15190–15200, 2023.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pp. 1152–1157, 2016.

Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 3045–3059, 2021.

Mosh Levy, Alon Jacoby, and Yoav Goldberg. Same task, more tokens: the impact of input length on the reasoning performance of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15339–15353, 2024.

Moxin Li, Wenjie Wang, Fuli Feng, Yixin Cao, Jizhi Zhang, and Tat-Seng Chua. Robust prompt optimization for large language models against distribution shifts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1539–1554, 2023.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. Commongen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1823–1840, 2020.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 61–68, 2022.

OpenAI. Gpt-4o, 2025. URL `https://chat.openai.com/`.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 2080–2094, 2021.

Dengyun Peng, Yuhang Zhou, Qiguang Chen, Jinhao Liu, Jingjing Chen, Libo Qin, and Wanxiang Che. Dlpo: Towards a robust, efficient, and generalizable prompt optimization framework from a deep-learning perspective. *arXiv preprint arXiv:2503.13413*, 2025.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *EMNLP*, pp. 7957–7968, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1743–1752, 2015.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 4222–4235, 2020.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on machine learning research*, 2023.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

Guancheng Wan, Lucheng Fu, Haoxin Liu, Yiqiao Jin, Hui Yi Leong, Eric Hanchen Jiang, Hejia Geng, Jinhe Bi, Yunpu Ma, Xiangru Tang, et al. Beyond magic words: Sharpness-aware prompt evolving for robust large language models with tare. *arXiv preprint arXiv:2509.24130*, 2025.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pp. 1058–1066. PMLR, 2013.

Xingchen Wan, Ruoxi Sun, Hootan Nakhost, and Sercan Ö Arık. Teach better or show smarter? on instructions and exemplars in automatic prompt optimization. *NeurIPS*, 37:58174–58244, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Yiyang Wang, Chen Chen, Tica Lin, Vishnu Raj, Josh Kimball, Alex Cabral, and Josiah Hester. Companioncast: A multi-agent conversational ai framework with spatial audio for social co-viewing experiences. *ACM CHI 2026 Workshop on Human-Agent Collaboration*, 2026a.

Yiyang Wang, Yiqiao Jin, Alex Cabral, and Josiah Hester. Mascot: Towards multi-agent socio-collaborative companion systems. *arXiv:2601.14230*, 2026b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Chunlong Wu and Zhibo Qu. Reflection-enhanced meta-optimization integrating textgrad-style prompt optimization with memory-driven self-evolution. *arXiv preprint arXiv:2508.18749*, 2025.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *eprint arXiv: 2407.10671*, 2024a.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv e-prints*, pp. arXiv–2412, 2024b.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024c.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Yaoning Yu, Ye Yu, Peiyan Zhang, Kai Wei, Haojing Luo, and Haohan Wang. Sipdo: Closed-loop prompt optimization via synthetic data feedback. *arXiv preprint arXiv:2505.19514*, 2025.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Tom Zehle, Moritz Schlager, Timo Heiß, and Matthias Feurer. Capo: Cost-aware prompt optimization. *arXiv preprint arXiv:2504.16005*, 2025.

Peiyan Zhang, Haibo Jin, Leyang Hu, Xinnuo Li, Liying Kang, Man Luo, Yangqiu Song, and Haohan Wang. Revolve: Optimizing ai systems by tracking response evolution in textual optimization. *arXiv preprint arXiv:2412.03092*, 2024.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The eleventh international conference on learning representations*, 2022.

Jingming Zhuo, Songyang Zhang, Xinyu Fang, Haodong Duan, Dahua Lin, and Kai Chen. Prosa: Assessing and understanding the prompt sensitivity of llms. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1950–1976, 2024.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 2005.

A Experimental Details

A.1 Dataset Details

This section elaborates on the datasets summarized in Section 5.1. We conduct experiments on nine reasoning datasets that span symbolic and arithmetic domains, deliberately chosen so that each source task has well-defined harder or related variants for testing cross-dataset generalization. Following common practice in prompt optimization (Yuksekgonul et al., 2024), we report strict string-based exact-match accuracy throughout. Each dataset is described below.

- **Big-Bench Hard (BBH)** (Suzgun et al., 2023; Srivastava et al., 2023). A suite of 23 challenging multi-step reasoning tasks from BIG-Bench on which prior language models fell short of average human performance. We draw two task families from BBH, each with three difficulty levels parameterized by the number of objects involved:
  - **Logical Deduction (3 / 5 / 7 Objects):** Deduce the order of a sequence of objects from clues describing their spatial relationships and placements, then answer a query about the inferred ordering.
  - **Tracking Shuffled Objects (3 / 5 / 7 Objects):** Given the initial positions of a set of objects and a sequence of pairwise swaps applied to them, determine the final position of a specified object.

  Prompts are optimized on the 3-object variant of each family and evaluated out-of-distribution on the 5- and 7-object variants, allowing us to measure how well optimized prompts transfer to harder instances of the same task structure. For the 3-object variants used as source tasks, we follow TextGrad (Yuksekgonul et al., 2024) and adopt a 50 / 100 / 100 train / validation / test split.
- **GSM8K** (Cobbe et al., 2021). A widely used benchmark of linguistically diverse grade-school math word problems requiring multi-step arithmetic. We use it as the source task for arithmetic optimization.
- **SVAMP** (Patel et al., 2021). A challenge benchmark constructed by applying targeted variations to existing math word problems, designed to probe robustness rather than surface pattern matching.
- **MultiArith** (Roy & Roth, 2015; Koncel-Kedziorski et al., 2016). A set of multi-step arithmetic word problems requiring the composition of multiple elementary operations.

A.2 Counterpart Details

We summarize the three baselines compared against `TextReg`.

- **Zero-shot Chain-of-Thought (CoT)** (Kojima et al., 2022; Wei et al., 2022). A foundational baseline that prompts the model with cues such as "Think step-by-step" to elicit multi-step reasoning prior to producing the final answer.
- **TextGrad** (Yuksekgonul et al., 2024). A first-order optimization method that uses natural-language feedback from an evaluator LLM as a "textual gradient", iteratively refining the prompt from immediate, local feedback signals.
- **REVOLVE** (Zhang et al., 2024). An optimization method that builds on first-order techniques by tracking how model responses evolve across iterations. Through this historical context, REVOLVE seeks more stable optimization and avoids the local optima that often trap methods relying solely on single-step feedback.

A.3 Implementation Details

The experimental pipeline follows Section 5.1 and applies uniformly to `TextReg` and all baselines. Prompt execution is performed by Qwen2.5-7B-Instruct (Yang et al., 2024b) as the forward engine, while GPT-4o (OpenAI, 2025) acts as the shared backward engine responsible for all LLM-driven optimization operations (gradient generation, purification, semantic edit analysis, regularization synthesis, and prompt rewriting). Optimized prompts are evaluated on four open-source test engines: Qwen2-7B-Instruct (Yang et al., 2024a), Phi-3.5-Mini-Instruct (Abdin et al., 2024), Llama-3-8B-Instruct, and Llama-3.1-8B-Instruct (Dubey et al., 2024).

For iterative optimization, we adopt the same training budget across all methods, using a batch size of 3 over 12 optimization iterations (36 training samples in total), matching the protocol established in REVOLVE (Zhang et al., 2024). For LLM generation, we allow a maximum of 2000 new tokens with a top-p of 0.99, and set the decoding temperature to 0 throughout to ensure reproducibility. **TextReg**'s only hyperparameter, the relative length-growth threshold τ_C for the capacity channel of Semantic Edit Regularization, is set to $\tau_C = 0.2$ in all experiments. The only deviation from REVOLVE's protocol is on GSM8K: we relax the validation acceptance criterion by 1% (i.e., a prompt update is accepted whenever validation accuracy does not drop by more than 1%), since under the strict monotone criterion the prompt almost never updates on this task; this relaxation is applied uniformly to all methods.

All experiments are conducted on a server with six NVIDIA A100 80GB GPUs.

Unless stated otherwise, every reported number for TextGrad, REVOLVE, and **TextReg** is the mean over four independent optimization seeds, each of which produces a separately optimized prompt; CoT uses the same fixed prompt evaluated over four runs. OOD evaluation uses the full released splits: 250 examples for each BBH variant, 1,000 for SVAMP, and 600 for MultiArith. The 100-example splits above are the held-out source-task test sets used for the in-distribution results in Appendix D.

A.4 Initial Prompts

All methods and all seeds start from the same fixed, task-agnostic prompt for each task. The initializations are inherited from the zero-shot prompts released with TextGrad (Yuksekgonul et al., 2024) and are not tuned for any method: GSM8K uses the released prompt unchanged, and the two BBH tasks use the same wording with only the answer-format clause adapted to multiple-choice outputs. Appendix E.2 studies how sensitive the results are to this choice.

Initial Prompts

GSM8K (released TextGrad prompt; the spelling of "mathemetical" is preserved from the original): *You will answer a mathemetical reasoning question. Think step by step. The last line of your response should be of the following format: 'Answer: $VALUE' where VALUE is a numerical value.*

Logical Deduction and Tracking Shuffled Objects: *You will answer a reasoning question. Think step by step. The last line of your response should be of the following format: 'Answer: ($VALUE)' where VALUE is the letter of the correct option.*

B Attribution of the Gains: Compute Budget and Additional Optimizers

The OOD gains in Tab. 1 could arise from factors other than regularization, including additional backward-engine calls, limited baseline coverage, or prompt-length control alone. We examine each possibility in turn. Over a complete 12-step run, **TextReg** uses 174 backward calls, compared with 84 for TextGrad and REVOLVE, due to gradient purification, RuleBank canonicalization, semantic diff analysis, and regularization synthesis. This increases the GPT-4o cost from $0.63 for TextGrad to $0.98 for **TextReg** and adds approximately 13% to the wall-clock time, while the shorter prompts reduce solver input tokens by 36% during optimization.

Matched budget. To isolate the effect of additional compute, we rerun TextGrad and REVOLVE with a task-only best-of-N search. At each step, eight candidate rewrites are sampled and one additional call selects among them using task fidelity alone, without any notion of length, scope, or generality. This yields 180 backward calls per run, slightly more than the 174 used by **TextReg**.

Additional optimizers. We further compare against OPRO (Yang et al., 2024c), DSPy-MIPROv2 (Khattab et al., 2023; Opsahl-Ong et al., 2024), DLPO (Peng et al., 2025), REMO (Wu & Qu, 2025), and a CAPO-style length penalty (Zehle et al., 2025) attached to the TextGrad backbone. DLPO uses its official implementation, while REMO is re-implemented from the paper using its mistake notebook, top-$k = 5$ retrieval, and meta-controller. For the CAPO-style control, candidate scores are penalized according to prompt length using CAPO's default coefficient $\gamma = 0.05$, normalized by the maximum prompt length observed within the run. We attach this penalty to TextGrad

Table 2: **Attribution of the OOD gains.** Out-of-distribution accuracy (%) averaged over the four test engines for the default-budget baselines, matched-budget best-of-N controls (180 backward calls per run vs. 174 for `TextReg`), and five additional comparison methods. The best and second-best results are highlighted with **bold** and underline, respectively.

OOD Dataset	Default budget		Matched budget (best-of-N)		Additional optimizers					TextReg
	TextGrad	REVOLVE	TextGrad	REVOLVE	OPRO	MIPROv2	DLPO	REMO	CAPO-pen.	
Logical Ded. 5obj	49.8	51.8	<u>54.9</u>	53.6	53.2	<u>54.9</u>	54.8	53.6	54.8	**56.9**
Logical Ded. 7obj	46.1	46.4	<u>50.1</u>	47.6	46.9	46.9	47.4	48.1	48.5	**51.5**
Tracking Shuf. 5obj	59.4	62.3	63.0	57.9	59.7	58.6	<u>67.9</u>	61.1	60.2	**68.4**
Tracking Shuf. 7obj	57.7	58.8	<u>63.6</u>	53.0	54.5	56.1	61.3	57.2	58.6	**64.1**

Table 3: **Prompt length and OOD accuracy along the optimization trajectory.** Intermediate prompts at step t are evaluated on the 7-object OOD variants; values are averaged over the two BBH source tasks and two test engines (Qwen2-7B-Instruct and Llama-3-8B-Instruct). For Steps 3–12, the best and second-best OOD accuracies are highlighted with **bold** and underline, respectively; all methods share the same Step-0 accuracy.

Method	Metric	Step 0	Step 3	Step 6	Step 9	Step 12
TextGrad	Prompt tokens	41	246	264	289	351
	OOD Acc (%)	42.0	43.0	<u>42.3</u>	43.2	39.8
REVOLVE	Prompt tokens	41	98	121	128	131
	OOD Acc (%)	42.0	<u>45.4</u>	42.2	<u>44.0</u>	<u>43.9</u>
TextReg	Prompt tokens	41	135	139	152	160
	OOD Acc (%)	42.0	**47.5**	**48.6**	**48.1**	**47.0**

rather than run the full population-based CAPO pipeline so that the starting prompt, search procedure, and optimization setting remain controlled, isolating the effect of constant length regularization.

`TextReg` achieves the highest mean accuracy on all four OOD evaluations in Tab. 2. Matching the call budget improves TextGrad substantially but does not close the gap, while REVOLVE benefits only on Logical Deduction. The CAPO-style length penalty, DLPO, and REMO also remain below `TextReg`. Together, these controls indicate that the gains cannot be explained by additional LLM calls, limited baseline coverage, or length control alone.

C Prompt Growth and OOD Accuracy during Optimization

The regularization objective in Eq. (5) motivates controlling prompt growth when additional capacity does not translate into transferable performance. Tab. 3 tracks prompt length and OOD accuracy along the optimization trajectory. TextGrad grows monotonically from 41 to 351 tokens, while its OOD accuracy fluctuates and ultimately falls below that of the starting prompt. In contrast, `TextReg` keeps growth bounded, reaching 135–160 tokens, and improves OOD accuracy by 5.5–6.6 points at steps 3–6 while retaining a 5.0-point gain at the final step. REVOLVE produces similarly compact prompts but consistently weaker OOD accuracy, showing that prompt length alone does not explain the improvement. RuleBank is also actively reused: a run ends with 27.4 canonical rules on average, and 79% of accepted rule mentions match an existing entry rather than creating a new one.

D In-Distribution Performance and the ID–OOD Trade-off

Regularization can trade some in-distribution (ID) fit for OOD robustness. Tab. 4 reports held-out source-task accuracy using Qwen2.5-7B-Instruct, the optimization forward engine, as the test engine. On the two BBH source tasks, `TextReg` is on average about two points below TextGrad in ID accuracy, while these tasks also show its largest OOD improvements: +7.1/+5.4 points on Logical Deduction 5/7 objects and +9.0/+6.4 on Tracking Shuffled Objects 5/7 objects, averaged over the four test engines in Tab. 1. On GSM8K, `TextReg` incurs no ID loss. Moreover, all three optimizers improve over the starting prompt on the BBH source tasks (91.0 on Logical Deduction and 84.5 on Tracking), showing that `TextReg` does not simply revert to the seed prompt. The component-wise ID effects are consistent with the OOD ablations in Figure 3: removing Semantic Edit Regularization raises ID accuracy from 88.8 to 90.1 but weakens OOD robustness, whereas removing Gradient Purification lowers ID accuracy to 85.8 and also harms OOD performance.

Table 4: **In-distribution accuracy (%) on the source tasks.** Held-out test split of each source task, with Qwen2.5-7B-Instruct as the test engine. The best and second-best results per column are highlighted with **bold** and underline, respectively.

Method	Logical Ded. 3obj	Tracking Shuf. 3obj	GSM8K
TextGrad	**92.2**	**89.8**	90.9
REVOLVE	91.8	86.0	90.9
TextReg	91.5	86.0	**91.5**

Table 5: **Comparison under the All-Weak pipeline.** OOD accuracy (%) when every LLM-driven role of every optimizer uses Qwen2.5-7B-Instruct, together with the range of final prompt lengths. For accuracy, the best and second-best results are highlighted with **bold** and underline; the shortest prompt range is shown in **bold**.

Test Engine	OOD Dataset	TextGrad	REVOLVE	TextReg
Qwen2 -7B-Instruct	Logical Ded. 5obj	52.6	52.8	**53.7**
	Tracking Shuf. 5obj	**46.6**	45.3	45.8
Llama-3 -8B-Instruct	Logical Ded. 5obj	51.8	46.5	**52.6**
	Tracking Shuf. 5obj	49.4	50.0	**52.1**
Final prompt tokens		448–572	146–189	**57–84**

E Robustness of the Gains

We next test whether the main findings remain robust to the backward engine, prompt initialization, test engine, and protocol choices.

E.1 Fully Weak Backward Pipeline

While Sec. 5.4 weakens one LLM-driven role at a time, here all three roles—gradient processing, semantic edit regularization, and prompt rewriting—use Qwen2.5-7B-Instruct simultaneously. TextGrad and REVOLVE use the same weak backward engine under the same protocol. Across the eight BBH cells on Qwen2-7B-Instruct and Llama-3-8B-Instruct, All-Weak `TextReg` retains 91–102% of its All-Strong accuracy from Tab. 1. In the four matched comparisons shown in Tab. 5, it ranks first in three and remains within one point of the best in the fourth, while producing substantially shorter prompts. These results suggest that the observed regularization effect does not depend on GPT-4o alone.

E.2 Dependence on the Initial Prompt

The default 41-token seed in Appendix A.4 is short and generic, which could favor a method that regularizes toward compact prompts. We therefore re-optimize Tracking Shuffled Objects (3 objects) from two richer, task-agnostic initializations and evaluate on the 5- and 7-object variants. As shown in Tab. 6, `TextReg` achieves the highest mean in seven of eight cells. TextGrad also continues to expand substantially from these richer seeds: starting from 55 and 57 words, its final prompts average 144 and 314 words, whereas `TextReg` ends between 104 and 140 words.

Alternative Initializations

Hand-written: *Answer the following multiple-choice reasoning question. Think through the problem step by step and base your answer only on the information given in the question. Do not skip steps in your reasoning. Conclude with the final answer on its own line in the format 'Answer: ($VALUE)', where VALUE is the letter of the correct option.*

Structured: *You are an assistant for multiple-choice reasoning questions. Task: solve the question below. Method: reason step by step, using only the information given in the question. Check: verify your conclusion against the question before answering. Output: end your response with a line in exactly this format: 'Answer: ($VALUE)' where VALUE is the letter of the correct option.*

E.3 Test Engines and Protocol Sensitivity

Stronger and aligned test engines. Evaluating the main-run prompts on Qwen2.5-32B-Instruct (Yang et al., 2024b), we find that `TextReg` remains best on all four BBH transfers: 91.6/88.0 on Logical Deduction 5/7 objects and 99.2/96.5 on Tracking Shuffled Objects 5/7 objects, compared with 90.6/85.8 and 97.7/95.0 for TextGrad and 90.1/85.3 and 97.7/94.9 for REVOLVE. Tracking is close to

Table 6: **OOD accuracy (%) under alternative initial prompts.** Prompts are optimized on Tracking Shuffled Objects 3obj from the hand-written and structured initializations above. The best and second-best results are highlighted with **bold** and underline, respectively.

Test Engine	Initialization	OOD Dataset	TextGrad	REVOLVE	TextReg
Qwen2 -7B-Instruct	Hand-written	Tracking Shuf. 5obj	45.0	42.2	**47.8**
		Tracking Shuf. 7obj	33.4	35.2	**35.4**
	Structured	Tracking Shuf. 5obj	**44.6**	43.2	44.0
		Tracking Shuf. 7obj	41.4	38.4	**41.6**
Llama-3 -8B-Instruct	Hand-written	Tracking Shuf. 5obj	43.8	48.6	**54.8**
		Tracking Shuf. 7obj	40.8	41.0	**44.6**
	Structured	Tracking Shuf. 5obj	45.6	45.8	**50.8**
		Tracking Shuf. 7obj	41.4	42.6	**47.0**

Table 7: **Knowledge-intensive QA and open-ended generation.** Test performance (%) after optimization and final prompt word count. GPQA-main is scored by exact match of the answer letter; CommonGen reports the percentage of outputs judged by GPT-4o to cover all concepts fluently. For task performance, the best and second-best optimizer results are highlighted with **bold** and underline, respectively.

Task	Metric	Starting Prompt	TextGrad	REVOLVE	TextReg
GPQA-main	Accuracy (%)	30.6	32.0	33.2	**33.9**
	Final prompt words	33	193	158	158
CommonGen	Judged acc. (%)	87.0	83.0	84.0	**88.0**
	Final prompt words	31	93	54	48

ceiling for all methods; the clearest margin appears on the less-saturated Logical Deduction 7-object setting, where `TextReg` leads by more than two points. The ranking also persists when the forward and test engines are aligned: on Tracking Shuffled Objects 7 objects, `TextReg` reaches 50.8 with Llama-3-8B-Instruct (47.6 for TextGrad, 46.6 for REVOLVE) and 93.3 with Phi-3.5-Mini-Instruct (91.6 and 90.9).

Capacity threshold. τ_C is the only numeric hyperparameter of `TextReg`: RuleBank insertion and matching are LLM judgments, while recurrence enters purification as a soft prior rather than a hard threshold. Sweeping τ_C over $\{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ on the two BBH source tasks changes 7-object OOD accuracy, averaged over the four test engines, by at most 2.5 points (54.6–57.1), while final prompt length remains between 152 and 181 tokens. This suggests that performance is not highly sensitive to the exact threshold.

GSM8K acceptance tolerance. The 1% validation-acceptance tolerance on GSM8K (Appendix A.3) is applied identically to all methods. Under strict non-degradation, each method accepts only 0–3 of the 12 updates, leaving the resulting prompts close to initialization. The corresponding SVAMP/MultiArith accuracies, averaged over the four test engines, are 87.2/97.1 for `TextReg`, 86.9/96.7 for REVOLVE, and 86.8/96.4 for TextGrad. Thus, the ranking is unchanged, indicating that the GSM8K-transfer results in Tab. 1 are not driven by the 1% tolerance.

F Beyond Symbolic and Arithmetic Reasoning

To test whether the observed behavior is specific to symbolic and arithmetic reasoning, we extend the same optimization protocol to GPQA-main (Rein et al., 2023), a knowledge-intensive science QA benchmark, and CommonGen (Lin et al., 2020), a concept-to-sentence generation task. We use GPT-4o as the backward engine and Qwen2.5-7B-Instruct as both solver and test engine for 12 optimization steps. GPQA uses a 50/100/298 train/validation/test split and exact-match accuracy on the answer letter; CommonGen outputs are judged by GPT-4o for complete concept coverage and fluency. As shown in Tab. 7, `TextReg` achieves the highest GPQA accuracy with less prompt growth than TextGrad. On CommonGen, TextGrad and REVOLVE both fall below the starting prompt, whereas `TextReg` improves over it while producing the shortest final prompt. These results provide preliminary evidence that the phenomenon and its mitigation extend beyond the original reasoning benchmarks.

G PROMPT DETAILS

This section presents the LLM-driven prompt templates used by **TextReg**'s three stages: gradient purification (Stage 1), semantic edit regularization (Stage 2), and regularization-guided prompt update (Stage 3).

G.1 DUAL-EVIDENCE GRADIENT PURIFICATION (Π_{GEN})

This prompt drives the LLM realizing Π_{gen}. It classifies each raw textual gradient as a generalizable rule, narrow case patch, or purely stylistic edit, and synthesizes the retained gradient into a concise behavioral principle. The RuleBank is exposed to the LLM as a historical recurrence prior.

Dual-Evidence Gradient Purification Prompt

You are the "Gradient Purifier". Your job is to decide whether a proposed feedback (gradient) contains genuinely generalizable improvements, and if so, synthesize them into a concise principle. You output either a purified summary or an empty string.

Input data

- `<CURRENT_SYSTEM_PROMPT>` `{current_prompt}`
- `<EXECUTION_CONTEXT>`: the user input and model output that triggered this gradient, `{gradient_context}`
- `<PROPOSED_GRADIENT>` `{gradient_text}`
- `<RULEBANK_SUMMARY>`: previously accepted generalizable rules and their frequency, `{rulebank_summary}`

Task Internally classify the gradient into one of three categories:
Category 1 — Generalizable Logic Fix → output purified text. The feedback identifies a reasoning flaw or logical gap that applies broadly across many inputs. If the RuleBank contains a similar rule with high `mention_count`, lean toward accepting. Strip specific entities, numbers, and scenario details; merge overlapping points; each sentence must describe a distinct actionable rule.
Category 2 — Narrow Edge-Case Patch → output empty string. The fix is tailored to the rare scenario in `<EXECUTION_CONTEXT>` and lacks RuleBank support.
Category 3 — Pure Style / Formatting → output empty string. The feedback only concerns tone, formatting, or presentation with zero impact on task correctness.

Output format Respond with valid JSON: `{"purified_gradient": "..."}`.

G.2 RULEBANK CANONICALIZATION

This prompt extracts canonical mid-level rules from each accepted purified gradient and either increments an existing RuleBank entry or inserts a new one, maintaining the `mention_count` statistics that serve as the empirical proxy $\widehat{s}_t(r)$.

RuleBank Canonicalization Prompt

You are a rule canonicalization and matching engine. Given a raw textual gradient and the current RuleBank, perform two tasks:

1. Extract mid-level canonical `{rule_scope}` rules from the raw gradient. Remove references to specific entities, exact numbers, or particular examples. Preserve structural `{rule_patterns}`. Keep rules at mid-level abstraction.
2. For each extracted rule, compare it with the existing RuleBank. If semantically equivalent to an existing rule (same structural pattern, not just similar wording), output an `INCREMENT` operation with that rule's ID. Otherwise, output an `INSERT` operation with the canonical description.

Input `[CURRENT RULEBANK] {rulebank_summary}; [RAW GRADIENT] {raw_gradient}.`

Output format Strictly valid JSON: `{"operations": [{"type": "increment", "rule_id": "R3", "value": 1}, {"type": "insert", "canonical_description": "...", "value": 1}]}`.

G.3 SEMANTIC DIFF ANALYZER (M_Δ)

This prompt drives the LLM realizing M_Δ. It compares the previous and current prompts at the rule level, classifies each change, and outputs an overall specificity direction that determines which entries appear in $\mathcal{A}_t$. The prompt defaults to `CASE_PATCH` unless strong RuleBank or cross-context support is present.

Semantic Diff Analyzer Prompt

You are the "Semantic Delta Analyzer". Compare `PREVIOUS_PROMPT` and `CURRENT_PROMPT` at the level of behavioral rules, classify each change, and judge the overall specificity shift.

What you must not do Do not judge whether the task logic or new rules are correct; do not act as a prompt quality evaluator; do not produce character-level or word-level diffs.

What you must do Identify each rule-level change between `PREVIOUS_PROMPT` and `CURRENT_PROMPT`. Default to `CASE_PATCH` unless there is clear positive evidence for `GENERALIZED_RULE`.
`GENERALIZED_RULE` — A broadly applicable, task-agnostic behavioral principle. Mark only if the rule is not tied to specific entities, numbers, or surface strings, and has at least one strong support signal: a semantically similar high-frequency RuleBank entry, or relevance across multiple execution contexts.
`CASE_PATCH` — The default. Best explained as triggered by a single specific example in `GRADIENT_CONTEXTS`, with no clear evidence it would apply beyond that example, and weak RuleBank support.
`STYLE_ONLY` — Pure wording or formatting change with no behavioral impact.
After classifying, judge the overall specificity direction: `increase` (adds or strengthens patch-like rules), `decrease` (removes `CASE_PATCH`s or replaces them with `GENERALIZED_RULE`s without adding new ones), or `neutral` (no meaningful net change).

Inputs `<INITIAL_PROMPT>`, `<PREVIOUS_PROMPT>`, `<CURRENT_PROMPT>`, `<RULEBANK_SUMMARY>`, `<GRADIENT_CONTEXTS>`.

Output format Strictly JSON: `{"rules_changed": [{"description": "...", "type": "..."}], "specificity_direction": "..."}`.

G.4 REGULARIZATION GRADIENT GENERATOR (Γ)

This prompt drives the LLM realizing Γ. It translates the active regularization directions $\mathcal{A}_t$ into concrete structural directives referencing specific rules in the current prompt. The prompt routes among four modes corresponding to elements of $\mathcal{A}_t$: `STRONG_REGULARIZATION` ($\{C, W\}$), `COMPRESSION_ONLY` ($\{C\}$), `GENERALIZE_ONLY` ($\{W\}$), or `NO_REGULARIZATION` ($\mathcal{A}_t = \varnothing$, in which case the prompt is skipped).

Regularization Gradient Generator Prompt

You are a structural regularization controller. You are given a `REGULARIZATION_MODE`, the current prompt, and a list of recent rule changes. Generate precise structural regularization guidance strictly according to the specified mode.

Mode definitions **`STRONG_REGULARIZATION`** — Both prompt length and rule specificity have increased. You must give firm directives to both compress and generalize: (1) merge redundant sentences expressing the same rule into a single shorter statement; (2) tighten verbose phrasing; (3) rewrite narrow rules as broader principles; (4) remove case-specific patches that cannot be generalized (but never remove broadly useful rules).
`COMPRESSION_ONLY` — Length increased but specificity did not worsen. Focus only on reducing length: merge redundant sentences and tighten verbose phrasing while preserving behavioral intent.
`GENERALIZE_ONLY` — Length is stable but new narrow rules were added. Focus only on broadening scope: identify narrow rules and rewrite them as broader principles.

Inputs `<REGULARIZATION_MODE>`, `<CURRENT_PROMPT>`, `<NEWLY_CHANGED_RULES>`.

Constraints Do not alter task semantics or remove rules clearly beneficial for task correctness. Reference specific sentences or rules from `<CURRENT_PROMPT>` and `<NEWLY_CHANGED_RULES>` — do not give generic advice.

Output format Strictly valid JSON: `{"guidance": "..."}`.

G.5 Regularization-Guided Prompt Update — System Prompt

This is the system message sent to the optimizer LLM at every prompt rewriting step.

Regularization-Guided Prompt Update System 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. You will receive some feedback, and use the feedback to improve the variable. The feedback may be noisy; identify what is important and what is correct. Pay attention to the role description of the variable, and the context in which it is used. You MUST give your response by sending the improved variable between `{new_variable_start_tag}` `{{improved variable}}` `{new_variable_end_tag}` tags.

Tag glossary `<LM_SYSTEM_PROMPT>`, `<LM_INPUT>`, `<LM_OUTPUT>`, `<FEEDBACK>`, `<CONVERSATION>`, `<FOCUS>`, `<ROLE>`, `<REG_FEEDBACK>` (**TextReg**-specific: semantic edit regularization feedback; follow its structural directives to control prompt length and specificity).

G.6 Regularization-Guided Prompt Update — User Message Trailing

This trailing instruction is appended to the optimizer's user message, implementing the task-faithful selection of Eq. (14): when task and regularization feedback target the same region, merge them into one edit; when they target different regions, apply both; when they conflict, fold the task fix into the reg-aware shape, and only as a last resort prioritize the task item over a specific reg item.

Regularization-Guided Prompt Update User Message Trailing

Improve the variable (`{variable_desc}`) by integrating both the task feedback in `<FEEDBACK>` and the regularization feedback in `<REG_FEEDBACK>`. Both contain concrete instructions; read them together and apply them as follows:

- When the task feedback and the regularization feedback point to the same part of the prompt, apply them as one combined edit.
- When they target different parts, apply both.
- When they conflict (e.g., the task feedback asks you to add a new rule while the regularization feedback asks you to merge or shorten), address the task feedback in the form most consistent with what the regularization feedback specifies — by folding the task fix into the change that the regularization feedback proposes, so the two are realized through a single coordinated edit instead of two unrelated ones.

If a specific task feedback item genuinely cannot be addressed without violating a regularization feedback item, prioritize that task feedback item and apply it completely, even if this means setting aside that specific regularization item. All other regularization items in `<REG_FEEDBACK>` still apply normally to the rest of the prompt.

G.7 Regularization-Guided Prompt Update — User Message Skeleton

The user-message skeleton is sent at every rewriting step. The `{reg_section}` placeholder is filled with the `<REG_FEEDBACK>` block when $g_{\text{reg}} \neq \varnothing$, and replaced by an empty string otherwise.

Regularization-Guided Prompt Update User Message Skeleton

Prefix template

```
Here is the role of the variable you will improve:
<ROLE>{variable_desc}</ROLE>.

The variable is the text within the following span:
<VARIABLE> {variable_short} </VARIABLE>

{reg_section}Here is the context and task feedback we got
for the variable:

<CONTEXT>{variable_grad}</CONTEXT>
```

Reg-feedback section (filled into `{reg_section}` when $g_{\text{reg}} \neq \varnothing$)

```
The following is semantic edit regularization feedback.
Follow its structural directives to control prompt length
and specificity:

<REG_FEEDBACK>{reg_feedback}</REG_FEEDBACK>
```