CROP: Token-Efficient Reasoning in Large Language Models via Regularized Prompt Optimization

Deep Shah[1], Sanket Badhe[1], Nehal Kathrotia[1], Priyanka Tiwari[2]

[1]Google LLC, [2]Purdue University

Abstract

Large Language Models utilizing reasoning techniques improve task performance but incur significant latency and token costs due to verbose generation. Existing automatic prompt optimization(APO) frameworks target task accuracy exclusively at the expense of generating long reasoning traces. We propose Cost-Regularized Optimization of Prompts (CROP), an APO method that introduces regularization on response length by generating textual feedback in addition to standard accuracy feedback. This forces the optimization process to produce prompts that elicit concise responses containing only critical information and reasoning. We evaluate our approach on complex reasoning datasets, specifically GSM8K, LogiQA and BIG-Bench Hard. We achieved an average 68% reduction in token consumption across diverse reasoning benchmarks (and up to 80.6% on algorithmic tasks) while maintaining competitive accuracy, seeing only a nominal decline in performance. This presents a pragmatic solution for deploying token-efficient and cost-effective agentic AI systems in production pipelines.

1 Introduction

Large Language Models (LLMs) have achieved unprecedented state-of-the-art capabilities in complex reasoning tasks primarily through step-by-step generation strategies like Chain of Thought (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2023b). By prompting models to decompose intricate problems into intermediate logical steps prior to outputting a final answer, researchers have significantly improved performance across mathematical, logical, and commonsense reasoning benchmarks (Nye et al., 2021; Zelikman et al., 2022; Cobbe et al., 2021). Reasoning capabilities are an indispensable requirement for deploying foundation models across diverse and critical domains, including financial analysis, algorithmic ranking, medical diagnosis, and legal reasoning (Wu et al., 2023; Cui et al., 2023; Badhe et al., 2026; Shah et al., 2026a). This paradigm encourages models to mimic structured human deliberation (Yao et al., 2023; Besta et al., 2024). Consequently, step-by-step rationale generation has become the standard operational mode for extracting high-fidelity reasoning from large-scale foundation models (Lightman et al., 2024).

Despite these performance gains, the reliance on verbose intermediate reasoning introduces a critical deployment hurdle for real-world production environments. The generation of thousands of intermediate tokens incurs massive inference latency and exorbitant financial costs (Pope et al., 2023; Aminabadi et al., 2022; Kwon et al., 2023). Recent empirical analyses reveal that modern reasoning models exhibit substantial verbosity compensation, often producing excessively lengthy explanations that provide little additional logical value (Patel et al., 2024; Xu et al., 2025b). This extensive token consumption thus presents a severe liability that restricts the practical viability of deploying complex reasoning pipelines (Renze and Guven, 2024; Li et al., 2024; Jiang et al., 2023).

To optimize these systems, the field has increasingly adopted automated prompt engineering frameworks that leverage LLMs as meta-optimizers (Zhou et al., 2023; Yang et al., 2024; Pryzant et al., 2023). Tools such as DSPy remove the need for manual human trial and error by refining prompts through systematic search (Khattab et al., 2024). Recent frameworks extend this paradigm by automating the optimization workflow, utilizing meta-learning and combinatorial search to systematically refine instructions and prompting strategies without human intervention (Spiess et al., 2025; Murthy et al., 2025; Xu et al., 2025a). However, these existing tools suffer from a critical flaw in that they optimize almost exclusively for task accuracy, often at the expense of generating excessively verbose intermediate reasoning traces. By continuously adding

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 934–951
November 6-10, 2026 ©2026 Association for Computational Linguistics

edge-case instructions and logic corrections to the prompt, the optimizer inadvertently maximizes the verbosity tax, producing bloated system prompts that force the target model to generate even longer and more exhaustive reasoning traces.

This inherent tendency toward verbosity exposes a fundamental missing mathematical constraint in the current paradigm of textual optimization. In standard machine learning optimization, loss functions routinely incorporate regularization techniques (such as L1 or L2 penalties) to constrain model weights and prevent overly complex architectures from overfitting (Goodfellow et al., 2016; Bishop and Nasrabadi, 2006). Yet, textual prompt optimization lacks a true analog to a length penalty.

To solve this structural deficiency, we propose CROP, an applied methodology that introduces an output-biased textual regularizer directly into the prompt optimization loop. CROP builds upon the multi-objective capabilities of textual differentiation to treat output token count as a first-class optimization constraint alongside task accuracy. By computing a continuous length-penalty gradient that functions independently of the task loss, the regularizer explicitly penalizes the target model whenever its output exceeds a minimal threshold. This feedback is then aggregated with traditional accuracy gradients using textual summation. Consequently, CROP forces the meta-optimizer to rewrite system instructions that explicitly constrain verbosity without sacrificing the underlying logical steps necessary for correct inference.

Through this regularized feedback mechanism, CROP successfully forces the discovery of concise reasoning paths. Our empirical evaluations demonstrate that this approach effectively shrinks the output distribution, minimizing both output length and overall inference latency. We evaluate CROP on complex reasoning benchmarks, specifically GSM8K (Cobbe et al., 2021), LogiQA (Liu et al., 2020), and BIG-Bench Hard (Suzgun et al., 2023; Srivastava et al., 2023). The results show that our cost-aware optimization method yields an average 68% reduction in output token consumption (and up to 80.6% on algorithmic tasks) while maintaining competitive accuracy. By establishing a continuous textual penalty for verbosity, CROP provides a pragmatic and highly effective solution for deploying token-efficient compound and multi-agent AI systems in strict production pipelines (Shah et al., 2026b).

1.1 Main Contributions

In summary, our primary contributions are as follows:

- **Cost-Aware Prompt Optimization (CROP):** We propose CROP, a novel token-efficient automatic prompt optimization framework. By introducing an response adaptive textual length penalty, CROP successfully decouples reasoning accuracy from generation verbosity during the prompt discovery phase.
- **Massive Inference Efficiency:** We empirically demonstrate that CROP reduces output token consumption by an average of 68% (and up to 80.6%) on complex reasoning benchmarks while strictly preserving the task accuracy of unconstrained Chain-of-Thought prompting.
- **Autonomous Discovery of Symbolic Reasoning:** We show that our regularized textual gradients automatically force the target LLM to adopt highly compressed, symbolic reasoning structures. This emergent behavior conceptually mirrors human-engineered strategies (e.g., "Chain-of-Draft") without requiring manual prompt design.

2 Related Work

2.1 Automatic Prompt Optimization

The optimization of natural language prompts has evolved from discrete vocabulary search (Shin et al., 2020; Deng et al., 2022; Prasad et al., 2023; Guo et al., 2023; Chen et al., 2024; Badhe and Shah, 2026) to utilizing large language models as meta-optimizers (Zhou et al., 2023; Yang et al., 2024; Pryzant et al., 2023; Wang et al., 2023a) and automated discovery frameworks (Vitvitskyi et al., 2026). Frameworks such as DSPy and TextGrad (Khattab et al., 2024; Yuksekgonul et al., 2024) have formalized this into comprehensive computation graphs, enabling automated textual backpropagation capable of optimizing user-defined metrics and rewards. However, when applied to complex reasoning tasks, textual optimization exposes a fundamental generative bias: because evaluators diagnose logic errors via natural language critiques, they inherently resolve failures by prescribing additive logic and expanded instructions. This accumulation of instructional leads to unchecked increase of reasoning leading to verbosity. Recog-

nizing this computational bottleneck, recent frameworks such as Promptomatix and AutoPDL (Pan et al., 2025; Murthy et al., 2025) have introduced cost-aware objectives to balance efficiency with performance, alongside discrete evolutionary algorithms like CAPO and ParetoPrompt (Wen, 2025; Authors, 2025; Zehle et al., 2025). Building upon this critical shift, CROP introduces a continuous, dual-objective textual optimization landscape. By applying an unconditional, output-biased regularizer alongside task gradients, CROP exerts continuous downward pressure on token bloat, forcing the meta-optimizer to systematically balance logical correctness with generative brevity in a way that purely additive textual gradients cannot.

2.2 Inference-Time Scaling and Self-Correction

Alternative approaches improve reasoning by scaling compute during inference. Methodologies ranging from Self-Consistency (Wang et al., 2023b) to iterative self-correction frameworks like Reflexion and Self-Refine (Shinn et al., 2023; Madaan et al., 2023) allow models to critique and repeatedly refine their intermediate outputs. While yielding substantial accuracy gains, these pipelines inherently multiply the token consumption and latency of the system with every additional trial. In contrast to these methods that severely compound inference overhead, CROP shifts the iterative refinement burden entirely to an offline prompt optimization phase, ultimately producing a single, highly efficient system prompt that elicits compressed reasoning trajectories in a single forward pass.

2.3 Cost-Aware Systems and Verbosity Control

To mitigate computational overhead (Pope et al., 2023; Kwon et al., 2023), researchers have developed system-level routing protocols (Chen et al., 2023; Ong et al., 2024; Aggarwal et al., 2024; Jian et al., 2023), and inference-time early exiting or truncation strategies (Mao et al., 2025; Jiang et al., 2023; Wu et al., 2025). Furthermore, recent literature focuses on explicitly compressing reasoning traces via continuous latent projections (Hao et al., 2024; Shen et al., 2025; Wei et al., 2025), conditioned textual pruning (Kang et al., 2024; Xia, 2025; Wang, 2025). However, these approaches introduce severe operational trade-offs. Methods relying on continuous latent projections or explicit fine-tuning are computationally prohibitive and alter foundation weights. Conversely, structured static prompting remains notoriously brittle, while dynamic token-pruning and step-skipping often require specialized decoding mechanisms or architectural interventions. Unlike approaches that demand expensive weight updates, custom decoding algorithms, or fragile hardcoded constraints, CROP operates entirely in the natural language optimization space. It dynamically modulates verbosity at the prompt level, systematically discovering optimal, token-efficient reasoning pathways through regularized textual feedback.

3 Methodology

In this section, we formalize **CROP**, a framework that extends the automatic prompt optimization (APO) paradigm to resource-constrained environments. We first briefly review the preliminary formulation of textual differentiation, and subsequently detail our novel textual regularization objective.

3.1 Preliminaries: Textual Differentiation

Let $\mathcal{V}^*$ denote the space of all possible natural language strings. In a standard LLM-based computation graph, we define a system prompt $p \in \mathcal{V}^*$ as a learnable parameter, an input query $x \in \mathcal{V}^*$, and a target model $\mathcal{M}$. The forward pass generates an output $\hat{y} = \mathcal{M}(x; p)$.

Given a ground-truth label y, a textual loss function $\mathcal{L}_{\text{task}}$ evaluates the prediction. Instead of yielding a scalar gradient, textual backpropagation queries a secondary evaluator LLM $\mathcal{E}$, to generate a *textual gradient* $g_{\text{task}} \in \mathcal{V}^*$. This gradient is a natural language critique detailing the logical flaws in $\hat{y}$ and suggesting semantic updates to p:

$$g_{\text{task}} = \mathcal{E}_{\text{task}}(p, x, \hat{y}, y) \tag{1}$$

An optimizer LLM, $\mathcal{O}$, then acts as the parameter update function, synthesizing the batch of gradients to perform a discrete step in the prompt space: $p_{t+1} = \mathcal{O}(p_t, \{g^{(i)}_{\text{task}}\}_{i=1}^{B})$, where B is the batch size.

3.2 Verbose Reasoning

While $\mathcal{L}_{\text{task}}$ effectively maximizes reasoning accuracy, it is agnostic to computational efficiency. Consequently, the meta-optimizer $\mathcal{O}$ inherently gravitates toward system prompts that trigger useful but overly verbose Chain-of-Thought (CoT) bloat

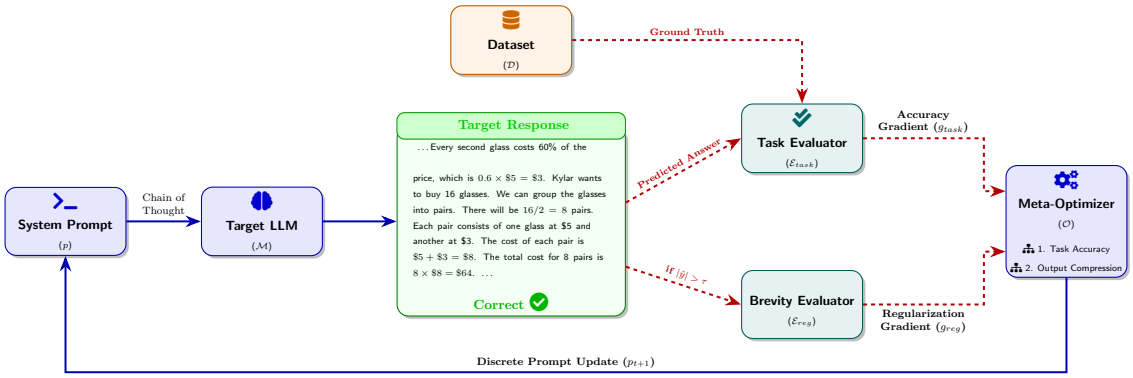


Figure 1: **Overview of CROP optimization pipeline.** A meta-optimizer updates the system prompt by aggregating textual gradients for both task accuracy (g_{task}) and response brevity (g_{reg}), systematically enforcing token-efficient reasoning.

in $\hat{y}$. In production systems, the length of $\hat{y}$ directly influences the inference latency and compute requirements.

3.3 Textual Regularization

CROP applies $\mathcal{L}_{\text{reg}}$ across the entire batch, asserting a constant downward pressure on token consumption. We define a secondary evaluator prompt, $\mathcal{E}_{\text{reg}}$, which receives the generated output and computes a textual brevity gradient:

$$g_{\text{reg}} = \mathcal{E}_{\text{reg}}(p, \hat{y}) \tag{2}$$

The resulting gradient g_{reg} explicitly instructs the optimizer to inject brevity constraints into p. The regularization gradient is computed only when $\|\hat{y}\|$ exceeds a predefined threshold, preventing the penalization of already token-efficient reasoning. To maintain optimization stability and prevent destructively large steps in the prompt space—which frequently induce catastrophic forgetting of the underlying logic—$\mathcal{E}_{\text{reg}}$ is constrained to enforce only a gradual, incremental reduction in output length per iteration. Furthermore, the evaluator is explicitly directed to target peripheral narrative elements and conversational filler, strictly preserving the tokens associated with critical computational steps.

3.4 Gradient Aggregation and Multi-Objective Optimize

To perform the parameter update, CROP treats prompt optimization as a formal multi-objective problem. We leverage the concatenation properties of textual autograd engines to compute the total gradient. For a single instance, the combined textual gradient g_{total} is defined via the string concatenation operator $\oplus$:

$$g_{\text{total}} = g_{\text{task}} \oplus \text{“\textbackslash n”} \oplus g_{\text{reg}} \tag{3}$$

During the batched optimization step, the meta-optimizer $\mathcal{O}$ receives the aggregated sets of both task gradients and regularization gradients and performs the update step $p_{t+1} = \mathcal{O}(p_t, \mathbf{g}_{\text{total}})$.

Rather than naively returning the prompt from the final iteration, CROP employs a structured selection criterion to identify the optimal prompt p^* discovered during the optimization trajectory. We define a composite score S for each candidate prompt:

$$S(p) = \text{Accuracy}(p) - \lambda \cdot L_{\text{norm}}(p) \tag{4}$$

where $L_{\text{norm}}(p)$ represents the generated token count normalized by the average COT output token length, and λ is a tunable regularization coefficient. The magnitude of λ dictates the explicit trade-off between task performance and token consumption. We selected λ through an empirical grid search on the validation set over the range of 0.01 to 0.5. We identified $\lambda = 0.2$ as the optimal configuration, as it provided the maximum reduction in reasoning verbosity without inducing a statistically significant degradation in task accuracy.

4 Experiment Setup

4.1 Model Setup

We assign specific Large Language Models to the distinct roles of evaluation, optimization, and inference. We utilize Gemini 2.0 Flash as the dual-evaluator ($\mathcal{E}_{\text{task}}$ and $\mathcal{E}_{\text{reg}}$) to compute task accuracy critiques and regularized verbosity gradients. For

Algorithm 1 CROP: Cost-Regularized Optimization of Prompts

Require: Initial system prompt p_0, Target LLM $\mathcal{M}$, Meta-Optimizer LLM $\mathcal{O}$
Require: Task Evaluator $\mathcal{E}_{\text{task}}$, Brevity Evaluator $\mathcal{E}_{\text{reg}}$
Require: Dataset $\mathcal{D}$, Batch size B, Max iterations T

$p \leftarrow p_0$
for $t = 1$ **to** T **do**
  Sample batch $\{(x_i, y_i)\}_{i=1}^{B} \sim \mathcal{D}$
  $G_{\text{batch}} \leftarrow \emptyset$ ▷ Initialize empty batch gradient list
  for $i = 1$ **to** B **do**
    $\hat{y}_i \leftarrow \mathcal{M}(x_i; p)$ ▷ Forward Pass
    $g_{\text{task}}^{(i)} \leftarrow \mathcal{E}_{\text{task}}(p, x_i, \hat{y}_i, y_i)$ ▷ Compute Task Accuracy Gradient
    $g_{\text{reg}}^{(i)} \leftarrow \mathcal{E}_{\text{reg}}(p, \hat{y}_i)$ ▷ Compute Textual Regularization Gradient
    $g_{\text{total}}^{(i)} \leftarrow g_{\text{task}}^{(i)} \oplus$ "\n" $\oplus\, g_{\text{reg}}^{(i)}$ ▷ Aggregate Gradients via Concatenation
    $G_{\text{batch}} \leftarrow G_{\text{batch}} \cup \{g_{\text{total}}^{(i)}\}$
  end for
  $\text{Accuracy}_t \leftarrow C_{\text{correct}}/B$
  $\text{AvgLength}_t \leftarrow L_{\text{total}}/B$
  $S_t \leftarrow \text{Accuracy}_t - \lambda \cdot L_{\text{norm}}(t)$ ▷ Evaluate trade-off score
  if $S_t > S_{\text{best}}$ **then** ▷ Update best prompt if score improves
    $S_{\text{best}} \leftarrow S_t$
    $p^* \leftarrow p$
  end if
  $p \leftarrow \mathcal{O}(p, G_{\text{batch}})$ ▷ Meta-Optimizer computes discrete prompt update
end for
return p^* ▷ Optimized token-efficient prompt

the parameter update step, we deploy Gemini 3.1 Pro (with Thinking enabled) as the default meta-optimizer ($\mathcal{O}$). A high-capacity model is critical here, as synthesizing multi-objective, conflicting textual gradients requires an extended context window and advanced deductive reasoning (ablated in Section 5). We additionally evaluate DeepSeek v4 Pro as an alternative meta-optimizer ($\mathcal{O}$) to test generalization across model families.

For forward-pass inference ($\mathcal{M}$), we evaluated two models: Gemini 2.0 Flash and Qwen 2.5 7B.

4.2 Baselines

We benchmark CROP against zero-shot prompting, chain-of-thought, conciseness-induced prompting, TextGrad, and CAPO. We intentionally omit static length-constrained prompts, which are brittle and degrade reasoning; furthermore, the model was unable to follow explicit token-limit instructions and frequently exceeded the given limits.

- **Standard Zero-Shot (Direct Prompting):** Prohibits intermediate reasoning steps to output the final answer directly. This establishes our empirical lower bound for token consumption, though typically at the severe expense of task accuracy.
- **Chain-of-Thought (CoT):** Appends a standard step-by-step reasoning directive. This serves as our upper bound for baseline accuracy, simultaneously representing the widely used but verbosity prompting technique.
- **Conciseness inducing prompts:** We used different prompts which encourages concise reasoning via prompting (Lee et al., 2025). (See prompts in Appendix A.1)
- **TextGrad:** An automatic prompt optimization framework that utilizes textual gradients to maximize task accuracy without penalizing verbose reasoning during the optimization loop.
- **CAPO (Zehle et al., 2025):** A recent cost-aware prompt optimization framework that balances task accuracy and token cost.

4.3 Datasets

To evaluate CROP across diverse cognitive domains, we selected five challenging reasoning benchmarks. Because these tasks inherently necessitate intermediate logic, they intrinsically incentivize unconstrained LLMs to generate highly verbose Chain-of-Thought (CoT) traces, providing an optimal testbed for our compression framework. To prevent data leakage, we utilize the official train, validation, and test splits provided by the dataset authors wherever available. For the BBH Object Counting task, which lacks standard splits, we manually partition the data into a 25/25/50 split for training, validation, and testing, respectively.

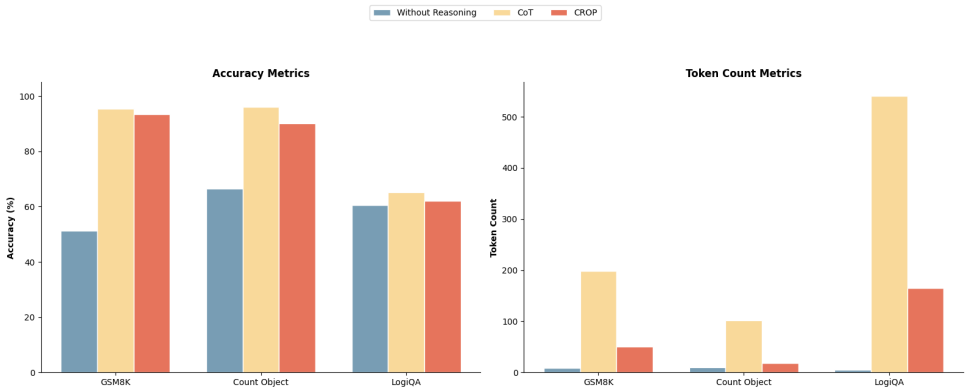


Figure 2: Performance and efficiency comparison across datasets using the Gemini 2.0 Flash model.

- **GSM8K (Cobbe et al., 2021):** A benchmark of linguistically diverse grade-school math word problems. This dataset requires accurate multi-step arithmetic reasoning, which frequently induces unconstrained LLMs to produce excessively long textual explanations for basic algebraic steps.

- **Object Counting (BIG-Bench Hard) (Suzgun et al., 2023):** An algorithmic reasoning task requiring the model to determine the total count of a specific semantic class from a natural-language collection of items. This tests symbolic tracking, where standard CoT prompts typically trigger exhaustive, token-heavy item enumeration.

- **LogiQA (Liu et al., 2020):** A challenging machine reading comprehension dataset sourced from expert logical examinations. It requires complex deductive inference (e.g., categorical, conditional, and disjunctive reasoning), assessing CROP's ability to maintain the structural integrity of formal logical arguments under aggressive token compression.

- **MATH (Hendrycks et al., 2021):** A dataset of competition-level mathematics word problems requiring complex, multi-step algebraic derivations to evaluate long-form reasoning trajectories.

- **HotpotQA (Yang et al., 2018):** A multi-hop question answering benchmark requiring information synthesis across multiple distinct paragraphs to evaluate open-ended textual reasoning.

Method	Core Instruction
No Reasoning	Output ONLY the final answer without reasoning or thinking.
Chain-of-Thought	Think step by step.
CROP (Ours)	[...] concisely, showing the combined essential calculations leading directly to the final numerical answer. Omit intermediate labels. Focus on chained calculations.

Table 1: Comparison of core system instructions across prompting baselines for the GSM8K dataset. Standard prefix formulations and output formatting constraints have been omitted to highlight the distinct reasoning directives.

5 Results and Analysis

We evaluate CROP's capacity to eliminate the computational redundancy inherent in unconstrained Chain-of-Thought (CoT) reasoning. By integrating a continuous brevity gradient into the optimization loop, CROP explicitly optimizes for reasoning density alongside task performance. As detailed in Table 3, our framework compresses output token consumption by an average of 68% (and up to 80.6%) without degrading exact-match accuracy. These results establish that high-fidelity inference can be sustained at a fraction of the standard computational cost.

5.1 Significant Reductions in Token Consumption

CROP consistently achieved state-of-the-art token efficiency while preserving the logical integrity re-

Qualitative comparison of generated reasoning traces on the GSM8K dataset.

Input Query: Kylar went to the store to buy glasses for his new apartment. One glass costs $5, but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them?

---

1. Direct Prompting (No Reasoning)
68

✗ **Incorrect**

---

2. Chain-of-Thought (Standard Verbosity)
Let the cost of one glass be $5. Every second glass costs 60% of the price, which is 0.6 * $5 = $3. Kylar wants to buy 16 glasses. We can group the glasses into pairs. There will be 16 / 2 = 8 pairs. Each pair consists of one glass at $5 and another at $3. The cost of each pair is $5 + $3 = $8. The total cost for 8 pairs is 8 * $8 = $64.

✓ **Correct (128 tokens)**

---

3. CROP (Ours Compressed Reasoning)
8 * 5 + 8 * (5 * 0.6) = 40 + 8 * 3 = 40 + 24 = 64

✓ **Correct (44 tokens)**

Table 2: Reasoning comparison on the GSM8K dataset.

quired for complex problem-solving. Across all five reasoning benchmarks—spanning arithmetic (GSM8K), algorithmic tracking (Object Counting), deductive logic (LogiQA), competition mathematics (MATH), and multi-hop text synthesis (HotpotQA)—CROP reduced output tokens by 74.8%, 80.6%, 66.5%, 38.7%, and 79.5% respectively for Gemini 2.0 Flash while maintaining competitive accuracy. Similar efficiency was observed for Qwen 2.5 7B. This starkly contrasts with Direct Prompting (Without Reasoning), which collapses on these complex tasks, confirming that while intermediate logic is indispensable, its narrative verbosity is highly compressible.

To assess optimization stability and reproducibility across runs, we evaluated CROP on GSM8K across 5 independent initialization seeds. Across these 5 runs, the framework proved remarkably stable, yielding an average accuracy of $93.2\% \pm 1.5\%$ alongside a consistent token reduction to 51.4 ± 4.3 tokens.

5.2 Emergent Symbolic Reasoning

A qualitative examination of the optimized prompts (Table 1) and generated outputs (Table 2) reveals that CROP discovered prompts fundamentally shifts how the target model structures its intermediate logic. Rather than generating conversational heuristics (e.g., "First, let us observe that..."), the regularized prompts elicit a dense, symbolic syntax. On GSM8K, the model spontaneously adopts a mathematical shorthand analogous to the recently proposed "Chain-of-Draft" (Xu et al., 2025b) methodology. However, whereas prior works rely on manual human engineering to elicit this behavior, CROP discovers this optimal reasoning syntax entirely autonomously.

5.3 Role of Optimizer Scale and Open-Weight Robustness

The meta-optimizer ($\mathcal{O}$) is responsible for synthesizing batched textual accuracy and regularization gradients into a coherent, updated system prompt. In early iterations, deploying a lower-capacity model (Gemini 2.0 Flash) as the meta-optimizer resulted in a severe degradation of prompt quality. The optimizer consistently overfit to idiosyncratic errors from isolated batch examples rather than deriving generalizable instructional updates. Extensive tuning of the meta-optimizer's system prompt failed to mitigate this limitation.

Ultimately, scaling the meta-optimizer to a frontier reasoning model—specifically Gemini 3.1 Pro with thinking capabilities enabled—resolved these issues and produced highly robust system prompts (See Table 4). We hypothesize that synthesizing complex, multi-objective textual feedback requires both the extended context window and the advanced deductive reasoning capabilities inherent to this larger model.

Open-Weight Robustness. To evaluate whether CROP's multi-objective optimization depends on proprietary APIs, we replicated our entire optimization pipeline substituting Gemini 3.1 Pro with the open-weight DeepSeek v4 Pro meta-optimizer. As demonstrated in Table 3, DeepSeek v4 Pro achieves matching optimization stability, prompt quality, and token compression across all evaluated benchmarks. This confirms that while synthesizing conflicting multi-objective textual gradients requires frontier-level reasoning capacity, CROP is model-agnostic and fully reproducible with open-weight foundation models.

5.4 Gradient Stability and Batch Dynamics

In standard deep learning, gradient variance is inversely proportional to batch size. We observe an analogous phenomenon in regularized textual differentiation. As demonstrated in Table 5, ablation experiments utilizing stochastic updates (batch size

Table 3: Model Performance Across Reasoning Benchmarks. Cells display Accuracy (%) with Average Token Count in parentheses. CROP achieves strong token compression while preserving accuracy across both proprietary (Gemini 3.1 Pro) and open-weight (DeepSeek v4 Pro) meta-optimizers.

Model	Dataset	CoT	w/o Reasoning	CROP (Gemini 3.1)	CROP (DeepSeek)	TextGrad	CAPO
Gemini 2.0 Flash	GSM8K	95.3 (198.4)	51.2 (9.2)	93.4 (50.0)	94.1 (54.0)	95.5 (185.5)	92.9 (121.1)
	Count Object	96.0 (101.2)	66.4 (9.4)	94.8 (19.6)	95.9 (18.7)	90.5 (24.5)	93.4 (37.6)
	LogiQA	65.2 (540.5)	60.4 (5.1)	64.2 (181.0)	63.5 (171.0)	66.4 (663.0)	61.1 (260.3)
	MATH	85.1 (509.0)	48.1 (14.5)	83.6 (312.0)	82.8 (343.0)	87.3 (593.0)	84.1 (461.9)
	HotpotQA	86.1 (83.0)	68.2 (8.0)	87.4 (17.0)	88.1 (19.0)	86.1 (68.0)	79.2 (21.0)
Qwen2.5 7B	GSM8K	90.2 (123.4)	54.9 (3.4)	87.8 (24.0)	91.1 (21.3)	89.8 (101.2)	86.7 (71.3)
	Count Object	92.1 (56.3)	67.8 (3.8)	90.1 (11.6)	91.4 (12.1)	88.8 (32.7)	89.6 (29.4)
	LogiQA	61.2 (287.0)	54.0 (3.4)	59.3 (84.2)	61.8 (84.9)	60.1 (212.3)	58.8 (160.2)
	MATH	80.4 (452.0)	45.8 (9.4)	79.1 (254.0)	77.9 (280.0)	82.6 (512.0)	79.6 (386.2)
	HotpotQA	81.3 (63.0)	65.3 (7.0)	82.5 (13.0)	82.0 (14.0)	83.7 (41.0)	80.2 (18.0)

Meta-Optimizer Model	Discovered System Prompt (Core Instructions)
Gemini 2.0 Flash ***(Lower Capacity)***	Solve the problem. Only provide the answer and do not provide any explanations.
Gemini 3.1 Pro ***(High Capacity)***	Solve the problem with extreme conciseness. Do not use conversational filler, restate the question, or write full sentences. Simply provide the final sum.

Table 4: Comparison of optimized system prompts generated by different meta-optimizers for the BBH Object Counting task. While the lower-capacity model (Gemini 2.0 Flash) defaults to a blunt truncation directive, the high-capacity model (Gemini 3.1 Pro) successfully synthesizes multi-faceted constraints targeting specific vectors of verbosity.

Optimization Setting	Discovered System Prompt (Core Instructions)
Low Capacity Optimizer (Gemini 2.0 Flash, $B = 128$)	Solve the problem. Only provide the answer and do not provide any explanations.
Stochastic Optimization (Gemini 3.1 Pro, $B = 1$)	Solve the problem concisely. Go straight to the answer without restating the problem or adding redundant explanatory sentences.
Optimal Configuration (Gemini 3.1 Pro, $B = 128$)	Solve the problem with extreme conciseness. Do not use conversational filler, restate the question, or write full sentences. Simply provide the final sum.

Table 5: Qualitative ablation of discovered system prompts under different optimization constraints for the BBH dataset. Only the optimal configuration (high capacity, large batch) successfully balances extreme conciseness with the preservation of reasoning.

of 1) yield extreme variance in the textual gradients. When evaluating an isolated example, the meta-optimizer frequently overfits to the length penalty, prioritizing output compression over task accuracy and aggressively stripping away critical cognitive instructions. This results in prompt collapse: the generation of degenerate directives that entirely eliminate intermediate reasoning capacity. Conversely, scaling to a batch size of 128 stabilizes the multi-objective feedback. By providing the meta-optimizer with a statistically representative sample of both reasoning failures and excess verbosity, large batches prevent regularization-induced collapse and ensure stable, convergent trajectories during prompt optimization.

Empirical ablations across batch sizes $B \in \{1, 8, 32, 64, 128, 512\}$ on GSM8K confirm that optimization stabilizes at $B \geq 32$ ($\sim$93% accuracy with $\sim$50 tokens). While moderate batches ($B = 8$) retain high accuracy (92.7%), they fail to generalize compression, producing outputs $1.8\times$ more verbose (90 tokens) than $B = 128$. Conversely, stochastic updates ($B = 1$) severely overfit the length penalty, collapsing intermediate reasoning (53.4% accuracy, 10 tokens). Detailed tabular results are provided in Appendix B.

5.5 Sensitivity and Break-Even ROI Analysis

Regularization Coefficient (λ). We investigated the sensitivity of the composite selection score to the regularization coefficient λ across $\lambda \in [0.01, 0.50]$ (Table 7 in Appendix C). Increasing λ from 0.01 to 0.20 smoothly reduces output tokens with minimal impact on exact-match accuracy. The setting $\lambda = 0.20$ marks the optimal Pareto frontier, achieving peak compression without triggering logic truncation. Penalties exceeding 0.20 lead to over-regularization, causing models to skip critical computational steps.

Economic Break-Even Analysis. To evaluate practical deployment viability, we tracked the exact API token consumption during CROP's prompt optimization phase on GSM8K (detailed in Appendix D). The total one-time optimization cost was $\sim$\$2.18 across evaluator and optimizer calls. Because CROP saves approximately 150 output tokens per query (\$0.00006 per query on Gemini 2.0 Flash), this upfront cost is fully recouped after only $\sim$36,333 inference queries. In production systems serving hundreds of thousands of user requests, the recurring token and latency savings quickly dwarf the one-time search expenditure.

6 Conclusion

The deployment of Large Language Models in complex reasoning pipelines has historically been bottlenecked by the severe latency and financial costs associated with verbose Chain-of-Thought generation. In this work, we introduced CROP, a novel multi-objective automatic prompt optimization framework. By integrating a continuous textual regularization penalty directly into the prompt optimization loop, CROP forces the meta-optimizer to navigate the Pareto frontier of accuracy and token-consumption.

Our empirical evaluations demonstrate that intermediate reasoning is highly compressible. CROP successfully reduces output token consumption by an average of 68% (and up to 80.6%) across five rigorous reasoning benchmarks while maintaining comparable accuracy compared to unconstrained CoT and automatic prompt optimization baselines. Furthermore, our analysis reveals that this dual-objective optimization automatically elicits highly efficient, symbolic reasoning structures, conceptually mirroring manual "Chain-of-Draft" prompting without requiring human intervention. Through systematic ablations, we also identified that successful multi-objective textual optimization requires frontier-level model capacity and large batch sizes to stabilize gradient variance and prevent the collapse of intermediate logic.

Ultimately, CROP provides a highly practical, deployment-ready methodology. By systematically excising computational redundancy at the prompt level rather than altering foundational weights, it enables the scalable and cost-effective integration of high-fidelity reasoning models into strict, real-world production and agentic environments.

Limitations

While CROP significantly reduces inference-time overhead, we acknowledge two primary limitations regarding its current operational scope.

One limitation of our current evaluation is that we do not explicitly consider input token length or cost. In modern production environments, prompt caching (or prefix caching) is a default feature that automatically stores and reuses the initial segments of instructions. Because our framework produces a single, static system prompt that remains identical across all user queries, it perfectly leverages this caching mechanism. By reusing this prior computational work, prefix caching reduces time-to-first-token latency by up to 80% and decreases input token costs by up to 90% (OpenAI, 2024). Consequently, the financial impact of a slightly longer optimized system prompt is effectively marginalized. This justifies our exclusive focus on constraining the verbosity of the generated output, which remains the primary driver of unmitigated inference costs.

Second, the optimization phase of CROP necessitates access to high-capacity reasoning models (such as Gemini 3.1 Pro or open-weight DeepSeek v4 Pro). We observe that frontier-level deductive capacity is required to effectively synthesize and balance conflicting objectives between task accuracy and textual regularization. Consequently, establishing the efficacy of lightweight, sub-8B models as meta-optimizers remains open for future research.

Acknowledgments

We thank the reviewers and area chairs for their constructive feedback and suggestions that helped improve this work.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. 2024. Automix: automatically mixing language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. 2022. Deepspeed- inference: Enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15.

MOPO Authors. 2025. Mopo: Multi-objective prompt optimization for affective text generation. In *Proceedings of the 31st International Conference on Computational Linguistics*.

Sanket Badhe and Deep Shah. 2026. Prompt-level distillation: A non-parametric alternative to model fine-tuning for efficient reasoning. *arXiv preprint arXiv:2602.21103*.

Sanket Badhe, Deep Shah, and Nehal Kathrotia. 2026. Long-tail knowledge in large language models: Taxonomy, mechanisms, interventions and implications. *arXiv preprint arXiv:2602.16201*.

Maciej Besta, Nils Blatter, Lucian Dragos, Jing Lin, Oksana Krause, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *International Conference on Learning Representations*.

Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Springer.

Jialin Chen et al. 2024. Evoprompt: Evolutionary prompt optimization for large language models. In *International Conference on Learning Representations*.

Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. In *Advances in Neural Information Processing Systems*.

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

Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. 2023. Chatlaw: Open-source legal large language model with integrated external knowledge bases. *arXiv preprint arXiv:2306.16092*.

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

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guihong Liu, Ke Jiang, Jian-Guang Yao, Min Zhang, et al. 2023. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*.

Shibo Hao et al. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.00000*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Yuhang Jian et al. 2023. Ecoassistant: Using llm assistant more affordably and accurately. In *Advances in Neural Information Processing Systems*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2024. C3ot: Generating shorter chain-of-thought without compromising effectiveness. *arXiv preprint arXiv:2412.11664*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Jian Jin. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*.

Ayeong Lee, Ethan Che, and Tianyi Peng. 2025. How well do llms compress their own chain-of-thought? a token complexity approach. *arXiv preprint arXiv:2503.01141*.

Zongqian Li, Yixuan Su, and Nigel Collier. 2024. 500xcompressor: Generalized prompt compression for large language models. *arXiv preprint arXiv:2408.03094*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, et al. 2024. Let's verify step by step. In *International Conference on Learning Representations*.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*.

Minjia Mao, Bowen Yin, Yu Zhu, and Xiao Fang. 2025. Early stopping chain-of-thoughts in large language models. *arXiv preprint arXiv:2509.14004*.

Rithesh Murthy, Ming Zhu, Liangwei Yang, Jielin Qiu, Juntao Tan, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. 2025. Promptomatix: An automatic prompt optimization framework for large language models. *arXiv preprint arXiv:2507.14241*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Aitor Lewkowycz Henry, Vikash Mansinghka, Vedant Mikulik, and Nathanael Schärli. 2021. Show your work: Scratchpads for intermediate computation with language models. In *Advances in Neural Information Processing Systems*.

Isaac Ong et al. 2024. Routellm: Learning to route llms with preference data. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

OpenAI. 2024. Prompt caching. Accessed: 2026-03-17.

Xiaoying Pan et al. 2025. Autopdl: Automatic prompt optimization for llm agents. *arXiv preprint arXiv:2504.04365*.

Rohan Patel, Kshitij Sharma, and Anil Gupta. 2024. Verbosity $\neq$ veracity: Demystify verbosity compensation behavior of large language models. *arXiv preprint arXiv:2411.07858*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. Grips: Gradient-free, edit-based instruction search for prompting large language models. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*.

Reid Pryzant, Dan iterative, et al. 2023. Automatic prompt optimization with gradient descent and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Matthew Renze and Erhan Guven. 2024. The benefits of a concise chain of thought on problem-solving in large language models. *arXiv preprint arXiv:2401.05618*.

Deep Shah, Sanket Badhe, and Nehal Kathrotia. 2026a. Taxonomy of the retrieval system framework: Pitfalls and paradigms. *arXiv preprint arXiv:2601.20131*.

Deep Shah, Sanket Badhe, Nehal Kathrotia, and Priyanka Tiwari. 2026b. Automated multi-agent system optimization. *Available at SSRN 7245620*.

Yikang Shen et al. 2025. Codi: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.

Claudio Spiess, Mandana Vaziri, Louis Mandel, and Martin Hirzel. 2025. Autopdl: Automatic prompt optimization for llm agents. *arXiv preprint arXiv:2504.04365*.

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. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

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

Alex Vitvitskyi, Michael Boratko, Matej Grcic, et al. 2026. Mining generalizable activation functions. *arXiv preprint arXiv:2602.05688*.

Chenglin Wang et al. 2023a. Instructzero: Efficient instruction optimization for black-box large language models. In *Advances in Neural Information Processing Systems*.

et al. Wang. 2025. Compressing chain-of-thought in llms via step entropy. *arXiv preprint arXiv:2508.03346*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. 2025. Sim-cot: Supervised implicit chain-of-thought. *arXiv preprint arXiv:2509.20317*.

et al. Wen. 2025. Pareto prompt optimization. In *International Conference on Learning Representations*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Zhaofeng Wu et al. 2025. An empirical study on prompt compression for large language models. *arXiv preprint arXiv:2505.00019*.

et al. Xia. 2025. Tokenskip: Controllable chain-of-thought compression in llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.

Guowei Xu, Mert Yuksekgonul, Carlos Guestrin, and James Zou. 2025a. metatextgrad: Automatically optimizing language model optimizers. *arXiv preprint arXiv:2505.18524*.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025b. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*.

Mert Yuksekgonul, Federico Bianchi, Daniil Boiko, et al. 2024. Textgrad: Automatic differentiation via text. *arXiv preprint arXiv:2406.07496*.

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

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D Goodman. 2022. Star: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*.

Yongchao Zhou, Andrei Ioan Muresanu, Zihao Zi, Keiran Tucker, et al. 2023. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*.

A Prompt Templates and Implementation Directives

A.1 Baseline prompts

CoT

Think Step by Step.

Be concise

Think Step by Step. **Be Concise.**

Only Num/Eq

Think Step by Step. **Only use numbers or equations.**

AbbrevWords

Think Step by Step. **Abbreviate words as much as possible.**

UseOnlyNTokens

Think Step by Step. **Do not use more than N words.**

A.2 Initial prompts

GSM8K

You will answer a mathemetical reasoning question.
The last line of your response should be of the following format: 'Answer: $VALUE' where VALUE is a numerical value.

BBH Object counting

Solve the problem
The last line of your response should be of the following format: 'Answer: VALUE' where VALUE is a numerical value.

LogiQA

You will answer a logical reasoning question.
The last line of your response should be of the following format: 'Answer: VALUE' where VALUE is the zero-based numerical index of the correct answer.

A.3 TextGrad Learnt Prompts

GSM8K

You will answer a mathematical reasoning question. First, explicitly and concisely state the final goal of the question.
Think step by step and be extremely concise in your reasoning. Avoid conversational filler and jump straight into the math.
Define your variables clearly and explicitly track units during your intermediate calculations to prevent conversion errors.
Carefully consider edge cases, hidden assumptions, overlapping sets, and ensure your final unit matches what the question asks for (e.g., dollars vs. cents). Double-check your arithmetic.
The very last line of your response must be exactly of the following format: 'Answer: VALUE'

where VALUE is a pure integer.
Leave a blank line before this final line to completely isolate it from the rest of your explanation. Ensure there is exactly one space after the colon. Do not include any dollar signs ($), commas, units, periods, or any other extraneous characters in VALUE. There should be absolutely no additional text, punctuation, or whitespace following the integer.

BBH Object counting

Solve the problem.
To ensure proper parsing, keep your response as concise as possible.
The very last line of your response must be exactly of the following format: 'Answer: VALUE' where VALUE is an integer.
Do not include any additional text, punctuation, or spaces after the numerical value.

LogiQA

You are an expert in logical reasoning. You will answer a logical reasoning question by selecting the correct option from the provided list.
Follow these systematic steps:
1. **Analyze the Query:** Carefully break down the context and the question. Identify the core logical structure, premises, definitions, and specific requirements. Pay special attention to negative constraints (e.g., "NOT", "cannot", "least likely") and the modality required (e.g., "must be true", "most weakens"). Explicitly state the ultimate goal of the question (e.g., "The goal is to find an alternative explanation that directly undermines the causal link").
2. **Formulate Criteria:** Establish clear, precise, and prioritized criteria that the correct answer must satisfy based on your logical analysis. Include negative criteria (what the correct option must *not* do) and distinguish between necessary and sufficient conditions.
3. **Evaluate Options:** Systematically and concisely analyze every option using its zero-based index (e.g., Option 0, Option 1, Option 2, Option 3). Use strong, decisive language to explain exactly how it meets or fails the formulated criteria.
- For "weaken/strengthen" questions, prioritize options that directly attack or support the core premise or mechanism over those providing indirect, tangential, or correlational impacts.
- For "must be true/inference" questions, ensure the deduction is a necessary consequence of the provided premises. Explicitly reject options that are merely "possible" or require external, unwarranted assumptions.
- Address potential nuances (e.g., correlation vs. causation, ambiguity, shifts in scope), clearly explaining why an incorrect option is logically flawed.
4. **Compare and Conclude:** Conduct a rigorous and direct comparative analysis of the most viable options. Be decisive. Explicitly contrast the strengths of the correct option against the specific weaknesses of the runners-up to summarize why the selected option is definitively the best or only correct choice.
5. **Final Answer:** Skip any section heading for this step. On the very last line of your response, provide the final answer in the exact format:
Answer: VALUE (where VALUE is the single zero-based integer index of the correct option).
CRITICAL FORMATTING RULE: The final line MUST be exactly 'Answer: VALUE' (e.g., 'Answer: 2') with NO additional spaces, NO periods, and NO extra text following the integer. The integer must be the absolute final character of your generated response.

A.4 CROP Prompts

GSM8K

You will answer a mathematical reasoning question concisely, showing the combined essential calculations leading directly to the final numerical answer. Omit intermediate labels. Focus on chained calculations. The last line of your response should be of the following format: 'Answer: VALUE' where VALUE is a numerical value.

BBH Object counting

You will answer a mathematical reasoning question. Present your reasoning solely through a single-line numerical calculation, summing only the pure numerical values identified from the question. Do not include item names, descriptions, or separate itemized lists. Do not include any introductory or conversational phrases. The last line of your response must be of the exact following format: 'Answer: VALUE' where VALUE is a pure numerical integer (without any currency symbols like '$' or extra characters).

LogiQA

You will answer a logical reasoning question.
Be concise and direct in your reasoning. Avoid unnecessary introductions, restatements of the problem, or verbose explanations. Briefly explain why the correct answer is right and succinctly state why the incorrect options are wrong.
The very last line of your response MUST be exactly of the following format: 'Answer: VALUE' where VALUE is the zero-based numerical index of the correct answer. Do not add any extra spaces or characters to this final line.

A.5 Gradient Prompts

Accuracy Feedback Gradient Prompt

You are an expert Prompt Engineer.

You will give feedback to a variable with the following role: <ROLE> response from the language model </ROLE>.
Here is an evaluation of the variable using a string-based function:

Function purpose: The runtime of string-based function that checks if the prediction is correct, where the answer is followed by "Answer :"

<INPUTS_TO_FUNCTION> {}

Ground truth answer(role: correct answer for the query): {} </INPUTS_TO_FUNCTION>

<OUTPUT_OF_FUNCTION> {} </OUTPUT_OF_FUNCTION>

Objective: Your goal is to give feedback and criticism to the variable given the above evaluation output. Our only goal is to improve the above metric, and nothing else. </OBJECTIVE_FUNCTION>

We are interested in giving feedback to the response from the language model for this conversation. Specifically, give feedback to the following span of text:

<VARIABLE> {} </VARIABLE>

Given the above history, describe how the response from the language model could be improved to improve the Objective. Be very creative, critical, and intelligent.

Length Regularization Gradient

You are an expert Prompt Engineer and Efficiency Strategist.

You will give feedback to a variable (also known as MODEL_PROMPT) with the following role: <ROLE> structured system prompt to a somewhat capable language model that specifies the behavior </ROLE>.

<MODEL_PROMPT> {} </MODEL_PROMPT>
<QUESTION> {} </QUESTION>
<MODEL_RESPONSE> {} </MODEL_RESPONSE>

<OBJECTIVE_FUNCTION>Reduce the length of the model response by only few words without hurting the quality, and trying to keep as much reasoning and important stuff and focus on removing the fluff.</OBJECTIVE_FUNCTION>

Large language model takes <MODEL_PROMPT> and <QUESTION> as input and generates the <MODEL_RESPONSE>.

Given the above history, describe how the response from the language model could be updated to achieve the <OBJECTIVE_FUNCTION> without hurting the quality of the response. Be very creative, critical, and intelligent.

A.6 Optimizer prompt

System Prompt Update Strategy

Here is the role of the variable you will improve:
<ROLE>structured system prompt to a somewhat capable language model that specifies the behavior and strategies for the QA task</ROLE>.

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

Here is the context and feedback we got for the variable:

<CONTEXT>
(The following block repeats for each conversation in the dataset:)

Here is a conversation:

<CONVERSATION>
<LM_SYSTEM_PROMPT> {current_system_prompt} </LM_SYSTEM_PROMPT>
<LM_INPUT> {questions[i]} </LM_INPUT>

<LM_OUTPUT> {forward_response[i]} </LM_OUTPUT>

</CONVERSATION>

This conversation is potentially part of a larger system. The output is used as response from the language model

Here is the feedback we got for structured system prompt to a somewhat capable language model that specifies the behavior and strategies for the QA task in the conversation:

<FEEDBACK>
{feedback_response[i]}
</FEEDBACK>

<REGULARIZATION_FEEDBACK>
{regularization_responses[i]}
</REGULARIZATION_FEEDBACK>

</CONTEXT>

Improve the variable (structured system prompt to a somewhat capable language model that specifies the behavior and strategies for the QA task) using the feedback provided in <FEEDBACK> and <REGULARIZATION_FEEDBACK> tags.
Send the improved variable in the following format:

<IMPROVED_VARIABLE>{the improved variable}</IMPROVED_VARIABLE>

Send ONLY the improved variable between the <IMPROVED_VARIABLE> tags, and nothing else.

B Batch Size Dynamics

Table 6 presents the impact of the optimization batch size B on the GSM8K dataset using Gemini 2.0 Flash as the target model.

Batch Size (B)	**Accuracy (%)**	**Avg. Output Tokens**
1	53.4	10
8	92.7	90
32	91.2	45
64	92.8	53
128	93.4	50
512	93.7	48

Table 6: Impact of optimization batch size on GSM8K (Target Model: Gemini 2.0 Flash). Optimization stabilizes at $B \geq 32$, whereas stochastic updates ($B = 1$) collapse intermediate reasoning due to over-regularization.

C Hyperparameter Sensitivity Analysis of λ

Table 7 evaluates the effect of the regularization coefficient λ on task accuracy and output token verbosity across GSM8K and BBH Object Counting using Gemini 2.0 Flash.

Penalty (λ)	GSM8K		BBH Object Counting	
	Accuracy (%)	Tokens	Accuracy (%)	Tokens
0.01	95.3	195	96.0	98
0.05	95.2	181	95.9	81
0.10	94.9	142	95.5	42
0.15	94.6	79	95.3	23
0.20	94.1	50	94.8	20
0.30	93.5	43	92.2	17
0.40	89.3	39	88.5	15
0.50	81.2	37	84.2	15

Table 7: Sensitivity analysis of the regularization coefficient λ on task accuracy and generation verbosity (Tokens). The setting $\lambda = 0.20$ provides the optimal Pareto trade-off between compression and reasoning fidelity.

D Economic and Break-Even ROI Analysis

Table 8 details the API token consumption during CROP's prompt optimization phase on GSM8K alongside downstream inference savings and query break-even threshold using standard Gemini API rates.

Phase	Metric	Value
Optimization Cost	Evaluator Tokens (Gemini 2.0 Flash)	547K Input / 418K Output
	Meta-Optimizer Tokens (Gemini 3.1 Pro)	486K Input / 1,016 Output
	Total One-Time Optimization Cost	∼$2.18
Inference Savings	Output Tokens Saved per Query	150 tokens
	Cost Savings per Query (Gemini 2.0 Flash)	$0.00006
ROI Analysis	Queries to Break-Even	∼36,333 queries

Table 8: Total cost and Break-Even analysis for CROP on GSM8K. Because CROP saves approximately 150 output tokens per query, the one-time optimization investment is recouped after approximately 36k queries.