

BRAVE: Budget-Aware Reflective AutoPrompting for Efficient Evolutionary Prompt Optimization

Nikita Kulin, Viktor Zhuravlev, Artur Khairullin
Sergey Muravyov, Ilya Makarov, Daniil Sukhorukov

ITMO University

Saint-Petersburg, Russia

nikita.kulin@itmo.ru, vnzhuravlev@itmo.ru, arkhairullin@itmo.ru

smuravyov@itmo.ru, iamakarov@hse.ru, d.sukhorukov@axxx.tech

Abstract

Automatic prompt optimization is an uncertain sequential decision problem: an editing operator’s quality gain and context-dependent token cost are unknown before execution. We introduce BRAVE (Budget-Aware Reflective AutoPrompting for Efficient Evolutionary Prompt Optimization), a population-based optimizer that treats this operational uncertainty as a control signal. Its EVC Controller maintains per-operator Bayesian models of quality gain, token cost, and improvement probability, then selects among eleven generation and mutation operators by uncertainty-penalized predicted utility per token under a predicted-cost feasibility guard. An EMA return-on-investment mechanism revises this decision process from realized outcomes, while diversity management preserves alternative prompt hypotheses. On seven datasets, BRAVE outperforms EvoPrompt, PromptBreeder, and GEPA on 7/7 tasks; its advantage over the strongest external baseline ranges from 0.002 to 0.150 absolute score points. Under 20k, 40k, and 80k nominal budgets, BRAVE respectively achieves average quality 0.779, 0.807, and 0.812 versus GEPA’s 0.771, 0.775, and 0.781, while using 21.4k, 41.0k, and 72.4k tokens versus 46.1k, 65.2k, and 96.5k. BRAVE therefore formulates reliable prompt optimization as uncertainty-aware resource allocation: uncertainty about an edit’s value and cost directly changes the action selected.

1 Introduction

Large language models are highly sensitive to the exact wording of their instructions (Zhao et al., 2021; Lu et al., 2022), motivating *automatic prompt optimization* (autoprompting): searching, rather than hand-crafting, for prompts that improve a downstream task. A prominent family of methods maintains a population of candidate prompts and

applies LLM-driven mutation and crossover operators (e.g., EvoPrompt (Guo et al., 2024), PromptBreeder (Fernando et al., 2023), SPELL (Li and Wu, 2023)); another uses self-reflection or textual-gradient feedback to refine prompts (e.g., APE (Zhou et al., 2022), GEPA (Agrawal et al., 2026)). These methods address uncertainty in the *task prompt* by exploring or revising its wording, but they typically leave a second, operational source of uncertainty implicit: before an edit is executed, the optimizer does not know how much quality it will add, whether it will help at all, or how many tokens its context-dependent LLM call will consume.

This operational uncertainty matters whenever optimization must make sequential decisions under a finite budget. An operator that looks attractive from its average quality improvement may be a poor choice when its benefit estimate is uncertain, its cost is volatile, the population has converged, or little budget remains. Conversely, a cheaper local rewrite may be the rational action when its expected utility is comparable but its downside is smaller. Treating token cost as an afterthought therefore conflates two distinct quantities: the expected value of an edit and uncertainty about realizing that value within the available resources. A nominal budget alone does not resolve this distinction. As we show in Section 4.3, GEPA’s reflective update can incur a single context-dependent cost on the order of 10^5 tokens, overshooting a nominal cap by $2\text{--}5\times$.

We frame prompt optimization as *uncertainty-aware resource allocation*. Our focus is not uncertainty calibration of the frozen base LLM’s task predictions; rather, it is epistemic uncertainty in the optimizer’s online estimates of its own actions’ quality gain and computational cost. This framing connects prompt optimization to utility-aware decision making: the system should not merely seek the edit with the largest estimated gain, but select an action whose predicted gain justifies both its expected cost and the uncertainty in that estimate.

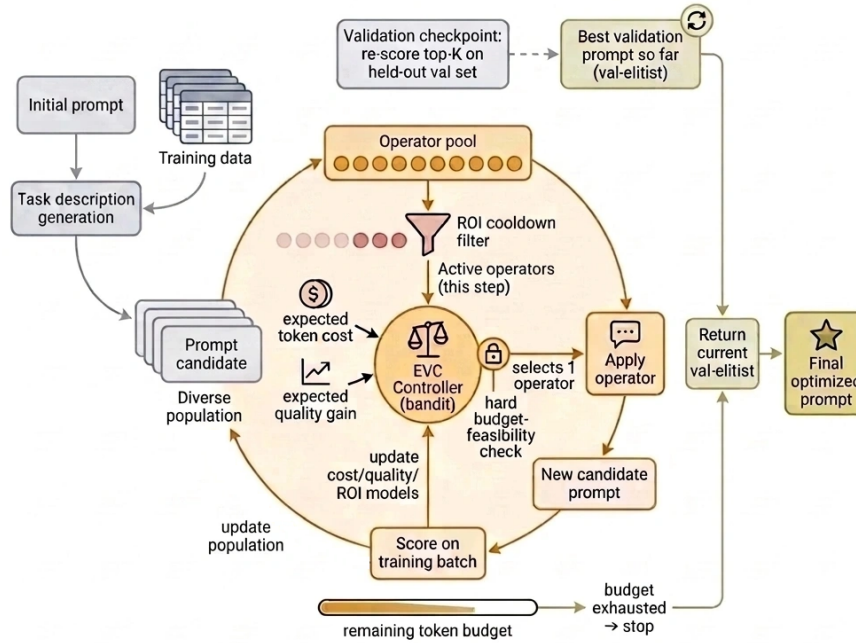


Figure 1: Overview of the BRAVE optimization loop. An initial prompt and training data seed a task description and a diverse initial population. At every iteration, the EVC Controller (a bandit) selects one operator from the active pool — after an ROI cooldown filter removes recently killed operators — subject to a hard budget-feasibility check; the selected operator produces a new candidate prompt, which is scored on a training batch and inserted into the population, feeding back into the controller’s cost/quality/ROI models. Periodic validation checkpoints track the best-scoring prompt seen so far (the val-elistist), which is returned once the token budget is exhausted.

We introduce BRAVE (Budget-Aware Reflective AutoPrompting for Efficient Evolutionary Prompt Optimization), an evolutionary prompt optimizer centered on the EVC Controller, a contextual bandit that selects, at every iteration, which of eleven mutation or generation operators to apply. For each operator, the controller maintains Bayesian predictive models of benefit, cost, and the probability of positive improvement over a nine-dimensional search-state descriptor. It ranks eligible actions by predicted quality gain per predicted token, boosts actions likely to improve, and conservatively inflates the effective cost of actions whose benefit or cost prediction has high posterior uncertainty. A hard feasibility guard then eliminates actions whose *predicted* cost exceeds the remaining budget (or a configurable share of it). This is a decision-time safeguard rather than a guarantee of realized spend: prediction error can still cause small deviations,

which we report transparently in Section 4.3. An EMA-based return-on-investment kill-switch provides a second adaptation mechanism, temporarily removing operators whose realized outcomes have been persistently unhelpful and re-admitting them for later re-evaluation. A hybrid TF-IDF/BERT clustering mechanism maintains alternative prompt hypotheses instead of allowing the population to collapse prematurely.

The resulting system turns uncertainty into an actionable control signal. Posterior uncertainty affects the immediate price assigned to an action; observed outcomes revise the controller’s beliefs; and budget feasibility changes the set of actions that can be taken as resources diminish. BRAVE offers three ready-made presets (BRAVE, BRAVE-Light, and BRAVE-Ultra-Light), while retaining configurable control over this quality–cost–uncertainty trade-off.

Our contributions are as follows:

- We formulate autoprompting as uncertainty-aware sequential resource allocation and introduce the EVC Controller: per-operator Bayesian models predict quality gain, token cost, and improvement probability from the evolving search state.
- We introduce an uncertainty-sensitive utility rule that penalizes uncertain actions through cost inflation, together with a decision-time predicted-cost feasibility guard and an EMA-ROI re-evaluation mechanism for adapting to realized operator outcomes.
- Across seven datasets and three nominal token-budget tiers, we show that BRAVE matches or exceeds three established baselines on quality while producing substantially more predictable realized token use than the closest budget-aware baseline, GEPA.
- We provide a configurable, diversity-managed evolutionary prompt-optimization framework with three default presets, making the trade-off between expected utility, resource use, and uncertainty explicit at every search step.

Our implementation is available in the CoolPrompt repository: <https://github.com/CTLab-ITMO/CoolPrompt>.

2 Related Work

Evolutionary autoprompting methods such as EvoPrompt (Guo et al., 2024) and PromptBreeder (Fernando et al., 2023) use fixed or uniformly sampled edit strategies, while APE (Zhou et al., 2022) and SPELL (Li and Wu, 2023) rely on a single implicit generation strategy. None jointly adapts operator choice to predicted quality and token cost. GEPA (Agrawal et al., 2026) is closest in using reflective updates and an explicit budget, but its monolithic update can have highly context-dependent cost. BRAVE instead allocates a population-based search budget across heterogeneous operators using predicted utility per token.

Adaptive operator selection is established in evolutionary computation (Fialho et al., 2009) and has recently been explored for LLM prompt-editing strategies (Ashizawa et al., 2025). Unlike these approaches and prompt-length/accuracy Pareto search (Kumar et al., 2026), BRAVE combines

per-operator uncertainty-aware utility estimates, a predicted-cost feasibility guard, temporary ROI-based suppression, and diversity-managed prompt search. Appendix A gives the extended comparison.

3 Method: BRAVE

3.1 Overview and Algorithm

BRAVE optimizes a natural-language prompt for a target task through four stages, illustrated schematically in Figure 1.

(0) Task-description generation: from the user’s initial prompt and the training dataset, BRAVE auto-generates a detailed task description — including expected input/output format — which several operators condition on.

(1) Initial population generation: a small set of prompts is produced from the original prompt using one of several initialization strategies (Section 3.4), trading off diversity against cost.

(2) Iterative optimization loop: at each iteration, the EVC Controller (Section 3.3) selects one not-yet-killed operator among the eleven available (Section 3.2); the operator executes on a prompt chosen from the population by an operator-specific selection rule (random, fitness-probability-weighted, or best-in-population); the resulting candidate is scored on a stratified training batch, inserted into the population, and the controller updates its operator-quality and operator-cost models from the realized outcome.

(3) Final selection: at configurable validation checkpoints, the top- K population prompts are re-scored on a held-out validation set, and the best validation-scoring prompt seen at any checkpoint — the *val-elitist* — is tracked and returned as BRAVE’s final output, decoupling the reported result from noisy per-batch training scores.

3.2 Operators

BRAVE offers eleven operators that transform one or two parent prompts (or the task description) into a candidate prompt. They range from one-call local rewrites to four-call reflective crossover, making their quality–cost trade-off substantial. Table 1 summarizes the operators; full selection rules and reflection dependencies are in Appendix B. Appendix H provides the meta-prompt templates and worked input/output examples for every operator.

Crossover creates short-term reflections from parent failures; these accumulate into a run-wide

Operator	What it does	Calls
Compressor	Compresses/shortens the prompt	1
CreativeRoleAndStyle	Invents a role+style, rewrites prompt in it	2
CreativeZero-Order	Generates a new prompt from the task description, maximally creative	1
Crossover	Two parents' failure-derived gradients merged into a short-term reflection, then crossed over	4
ElitistMutation	Mutates the population best using an accumulated long-term reflection	2
FewShot-Examples	Removes then re-inserts a TF-IDF-matched few-shot example targeting failures	2
GradientStep	Textual gradient applied directly to the prompt	2
HyPE	Meta-prompt restructures the prompt (role/task/format)	1
LongTerm-Mutation	Like ElitistMutation, but targets a random member	2
ParaphrasingBy-PD	Task-description-conditioned local paraphrase	1
ZeroOrder	Plain generation from the task description alone	1

Table 1: The eleven BRAVE operators (condensed). Full prompt-selection rules and reflection dependencies in Appendix B.

long-term reflection used by ElitistMutation and LongTermMutation. The remaining operators provide cost-heterogeneous local and structural edits.

3.3 The EVC Controller

Our main contribution is a cost-aware integration of contextual bandit control into evolutionary prompt optimization through the EVC Controller, a contextual bandit that selects which operator to apply at each iteration by trading off predicted quality gain against predicted token cost, subject to a hard budget-feasibility constraint.

Search state. At each step the controller observes a nine-dimensional state descriptor $x \in \mathbb{R}^9$, with every component normalized to approximately $[0, 1]$:

$$x = [q_{\text{best}}, \Delta_{\text{slope}}, \text{stag}, r_{\text{useless}}, r_{\text{budget}}, p_{\text{epoch}}, \text{stag} \cdot r_{\text{budget}}, d_{\text{pop}}, \text{stag} \cdot (1 - d_{\text{pop}})]$$

where q_{best} is the best training-elitist score so far (clipped to $[0, 1]$); $\Delta_{\text{slope}} = (\text{clip}(q_{\text{best}}^{(t)} - q_{\text{best}}^{(t-1)}, -1, 1) + 1)/2$; $\text{stag} = \mathbb{I}\{|\text{raw slope}| < 10^{-4}\}$ flags stagnation; r_{useless} is the fraction of steps with non-positive quality gain; r_{budget} is the

remaining-to-initial budget ratio; p_{epoch} is steps-done over max-steps; and $d_{\text{pop}} = 1 - \text{sim}_{\text{TF-IDF}}$ is a fast, TF-IDF-only population-diversity proxy. The two interaction terms capture progress stalling under budget pressure, and stagnation being most concerning once the population has converged.

Per-operator models. Each of the eleven operators (*actions*) is modeled independently by three Bayesian linear regressors — benefit, cost, and $P(\Delta_{\text{quality}} > 0)$ via a sigmoid link — each keeping ridge-regularized sufficient statistics $A \in \mathbb{R}^{9 \times 9}$ (αI , $\alpha = 1$) and $b \in \mathbb{R}^9$, updated after every trial by the standard Bayesian linear regression posterior:

$$A \leftarrow A + \frac{xx^\top}{\sigma^2}, \quad b \leftarrow b + \frac{xy}{\sigma^2}, \quad (1)$$

where y is the realized regression target for that particular model — the realized Δq (Equation 7) for the benefit regressor, the realized cost c for the cost regressor, or $\mathbb{I}\{\Delta q > 0\}$ for the probability regressor — with $\sigma^2 = 1$, posterior mean $\mu = A^{-1}b$, covariance $\Sigma = \sigma^2 A^{-1}$, and a Thompson-sampling draw $\theta \sim \mathcal{N}(\mu, \Sigma)$ (Thompson, 1933; Chapelle and Li, 2011; Li et al., 2010) at selection time. A small neural bandit can optionally be blended in — a shared-trunk MLP (one hidden layer, ReLU, width 32) with three per-action heads, trained online by SGD — linearly combined with the Bayesian prediction for whichever of the three quantities (benefit, cost, or probability) is being predicted, $\hat{p} = (1 - w_{\text{nn}})\hat{p}_{\text{Bayes}} + w_{\text{nn}}\hat{p}_{\text{nn}}$, where $\hat{p}$ is the resulting blended prediction actually used by the controller, and $\hat{p}_{\text{Bayes}}$, $\hat{p}_{\text{nn}}$ are the Bayesian-regressor and neural-network predictions being combined; all three shipped presets disable it ($w_{\text{nn}} = 0$, pure Bayesian prediction), reserving the blend (e.g., $w_{\text{nn}} = 0.55$) for users who want it.

Action-selection score. For each currently-eligible (non-killed) operator a , the controller computes an uncertainty-penalized benefit-over-cost score:

$$u(a) = \frac{1}{2}(\sigma_{\hat{\text{ben}}}(a, x) + \sigma_{\hat{\text{cost}}}(a, x)), \quad (2)$$

$$\hat{c}_{\text{eff}}(a) = \hat{c}(a) \cdot (1 + \beta u(a)), \quad (3)$$

$$\text{score}(a) = \frac{\hat{b}(a) \cdot (1 + w \hat{\pi}(a))}{\max(\hat{c}_{\text{eff}}(a), \varepsilon)}, \quad (4)$$

where $\hat{b}(a)$ and $\hat{c}(a)$ are the posterior-mean benefit and cost predictions for operator a at state x from its Bayesian (or blended) regressors, $\hat{\pi}(a)$ is its predicted probability of a positive quality gain, and

$\sigma_{\hat{\text{ben}}}(a, x)$, $\sigma_{\hat{\text{cost}}}(a, x)$ are the posterior standard deviations of, respectively, the benefit and cost predictions (so $u(a)$ grows with the controller’s residual uncertainty about a); $\beta = 0.5$ inflates a cheap operator’s effective cost when the controller is still uncertain about it, $w = 0.6$ boosts predicted benefit by the operator’s predicted probability of a positive gain, and $\varepsilon = 10^{-6}$ guards the denominator. The selected action is $a^* = \arg \max_a \text{score}(a)$, subject to a **hard feasibility constraint**:

$$\hat{c}(a) \leq B_{\text{rem}}, \quad \hat{c}(a) \leq \phi \cdot \max(B_{\text{rem}}, 1), \quad (5)$$

where B_{rem} is the remaining token budget and ϕ caps the share of it any single action may consume; all three shipped presets set $\phi = 1.0$, so this reduces to the base check $\hat{c}(a) \leq B_{\text{rem}}$, with a stricter per-step share available to users who configure $\phi < 1$. This enables BRAVE to consistently adhere to the specified budget constraints, rather than treating budget as a soft preference: no operator can ever be selected once its predicted cost would exceed what remains of the budget, and, unlike GEPA’s single monolithic reflective call, every BRAVE operator issues at most four calls to a normally-scoped prompt — together, this is the mechanism behind BRAVE’s tight budget adherence in Section 4.3.

Realized update. After an operator executes, the controller observes a realized benefit Δq (Equation 7), a realized cost $c = \max(c_{\text{actual}}, \varepsilon)$ — where c_{actual} is the operator’s actual measured token cost for that call — and $\mathbb{I}\{\Delta q > 0\}$, updating all three models toward these targets, plus an exponential moving average of realized return-on-investment,

$$\text{ema_roi}_a \leftarrow (1 - \alpha) \text{ema_roi}_a + \alpha \cdot \frac{\Delta q}{c}, \quad (6)$$

with $\alpha = 0.3$ — the same quantity the kill-switch below consumes, sharing one ROI concept between a forward-looking predictive ratio and a lagging realized EMA.

Reward shaping. The Δq fed to the controller is a population-aware shaped delta rather than simply the new candidate’s score minus the current best,

$$\begin{aligned} \Delta q = & (s_{\text{new}} - q_{\text{best}}) + \lambda_{\text{mean}} (s_{\text{new}} - \bar{s}_{\text{pop}}) \\ & + \lambda_{\text{min}} (s_{\text{new}} - s_{\text{pop}}^{\text{min}}), \end{aligned} \quad (7)$$

where s_{new} is the new candidate’s score on the current training batch, and $\bar{s}_{\text{pop}}$ and $s_{\text{pop}}^{\text{min}}$ are, respectively, the mean and the minimum training-batch

score across the current population (both computed over the population *before* the new candidate is inserted), with $(\lambda_{\text{mean}}, \lambda_{\text{min}}) = (2.0, 1.0)$ for BRAVE and $(0.0, 2.0)$ for BRAVE-Light/-Ultra-Light, densifying an otherwise sparse signal — raw elitist-improvement is zero once the population nearly converges, whereas Equation 7 still rewards beating the population mean or worst member.

Kill-switch. An operator becomes eligible for disabling after `kill_switch_min_trials` trials (5 for BRAVE, 3 for BRAVE-Light/-Ultra-Light), and is disabled once `ema_roi_a` falls to or below `kill_switch_roi_threshold` (0.0 in all three presets). Its cooldown is computed from historical usefulness $u_{\text{op}} = \text{success_count}/\text{trials}$ and $\text{lack} = 1 - u_{\text{op}}$:

$$\text{lack}' = \begin{cases} \sigma(15(\text{lack} - 0.5)), & \text{lack} < 0.5 \\ \sigma(4(\text{lack} - 0.5)), & \text{lack} \geq 0.5 \end{cases} \quad (8)$$

$$\text{cooldown} = \lfloor C_{\text{base}} + S \cdot \text{lack}' \rfloor, \quad (9)$$

where $C_{\text{base}} = \text{kill_switch_base_cooldown}$ (5 in all three presets) and $S = \text{kill_switch_scaling_factor}$ (20 for BRAVE, 3 for BRAVE-Light/-Ultra-Light), an asymmetric steepness escalating cooldown faster for a decisively useless operator than it relaxes for a borderline one (5–25 steps for BRAVE, 5–8 for the lighter presets). On expiry, `ema_roi` resets to zero, giving a fresh evaluation window.

3.4 Population Diversity and Initialization

Insertion and pruning. Every scored candidate is inserted unconditionally and the population re-ranked by score; pruning only triggers once capacity is exceeded (`population_size`; 5 for BRAVE, 3 for BRAVE-Light/-Ultra-Light). Near-duplicate prompts (hybrid similarity above 0.90) collapse to their better-scoring representative first; if still over capacity, hierarchical complete-linkage clustering (distance threshold auto-tuned to land the cluster count within ± 1 of the target size) groups the population, keeping the top `diversity_max_per_cluster` members (by score) per cluster (2 for BRAVE, 1 for BRAVE-Light/-Ultra-Light), with top-score truncation if needed. The clustering similarity blends TF-IDF and BERT embeddings, $\text{sim}_{\text{hybrid}} = (1 - w_{\text{bert}}) \text{sim}_{\text{TF-IDF}} + w_{\text{bert}} \text{sim}_{\text{BERT}}$, $w_{\text{bert}} = 0.6$, using sentence-transformers/all-MiniLM-L6-v2 (Reimers and Gurevych, 2019) for the BERT term. The diversity *state feature* fed to the bandit (d_{pop} , Section 3.3) uses only the cheaper TF-

IDF-only similarity, a fast proxy versus this fuller pruning mechanism.

Initialization. Three strategies apply a fixed operator set to the original prompt to build the starting population: *paraphrase* (N independent paraphrases only; poor exploration diversity, kept as an ablation baseline), *brave* (original, HyPE-restructured, and one paraphrase — 3 prompts), and *bigger* (original, HyPE, gradient-step, creative zero-order, and creative-role-and-style — 5 prompts), which the shipped BRAVE preset uses in our experiments. Composition follows a per-operator ablation (Table 6, App. C) restricted to operators usable before any scored population or reflection state exists (Crossover, ElitistMutation, LongTermMutation, and FewShotExamples all require one and are excluded); Few-Shot Mutation and Compression rank highest on average (0.459, 0.382), though Compression is not included in either initializer used by a shipped preset, a minor open design choice.

4 Experiments

Our experiments address two questions:

RQ1 (overall effectiveness): Does BRAVE improve final prompt quality over established evolutionary and reflective autoprompting baselines across heterogeneous NLP tasks?

RQ2 (budget-aware reliability): Under the same nominal token budget, can BRAVE keep realized token use more predictable while preserving or improving task quality relative to the budget-aware baseline GEPA?

4.1 Setup

We evaluate on seven datasets spanning generation, reasoning, QA, classification, summarization, medical QA, and code generation: CommonGen (Lin et al., 2020), GSM8k (Cobbe et al., 2021), SQuAD v2 (Rajpurkar et al., 2018), XSUM (Narayan et al., 2018), TweetEval (Barbieri et al., 2020), MediQA (Abacha et al., 2019), and CONCODE (Iyer et al., 2018) (BERTScore throughout, except exact match for GSM8k, F1 for TweetEval, and CodeBERTScore for CONCODE). We compare three BRAVE presets against three established baselines: PromptBreeder, EvoPrompt, and GEPA. Main experiments use gpt-4o-mini; results are mean $\pm$ std over three seeds (19, 13, 158). Cross-checks on gpt-5-nano and gemini-2.5-flash-lite are reported in Appendix D.

4.2 Main Results

RQ1: Does BRAVE improve final prompt quality across tasks? In the unconstrained-budget comparison, BRAVE attains the highest mean score among the external methods on all 7 datasets, with improvements over the strongest external baseline ranging from 0.002 (CommonGen) to 0.150 (CONCODE). This result tests whether adaptive, uncertainty-aware operator allocation transfers across tasks with distinct output structures and evaluation metrics, rather than only improving a single benchmark.

Table 2 reports quality under an unconstrained token budget (5 optimization epochs, population size 5). BRAVE beats the three external baselines on all seven datasets by mean score, narrowly edging out GEPA on CommonGen (0.809 vs. 0.807). BRAVE-Light and BRAVE-Ultra-Light remain competitive on most tasks, with the exception of CONCODE (code generation), where the cheaper presets fall well behind full BRAVE (a 0.207–0.222 gap) — suggesting code generation benefits disproportionately from BRAVE’s more expensive operators and larger search budget.

4.3 Budget-Constrained Results

RQ2: Does BRAVE make token-constrained optimization more reliable without sacrificing quality? At nominal budgets of 20k, 40k, and 80k tokens, BRAVE uses 21.4k, 41.0k, and 72.4k tokens on average, compared with GEPA’s 46.1k, 65.2k, and 96.5k, while attaining higher average quality at every tier (0.779 vs. 0.771, 0.807 vs. 0.775, and 0.812 vs. 0.781). Thus, BRAVE’s decision-time uncertainty and cost control improve both the predictability of resource use and the observed quality–cost trade-off.

To isolate the quality/cost trade-off directly, we compare BRAVE against GEPA — the only baseline here that also supports an explicit token budget — under hard nominal caps of 20k, 40k, and 80k, three seeds each. We only test a preset at a cap when it actually binds (BRAVE-Ultra-Light excluded at 40k, BRAVE-Light excluded at 80k).

Table 3 condenses the full per-dataset results (Appendix E) into per-configuration averages. BRAVE’s actual spend tracks its nominal cap closely at every tier (21.4k/41.0k/72.4k vs. 20k/40k/80k), while GEPA’s is substantially and inconsistently larger (46.1k/65.2k/96.5k). GEPA’s cost is *lumpy* enough to defeat a nominal budget

Method	CommonGen	GSM8k	SQuAD v2	XSUM	TweetEval	MediQA	CONCODE
BRAVE	0.809±0.003	0.904±0.016	0.934±0.002	0.727±0.004	0.803±0.007	0.721±0.006	0.843±0.009
BRAVE-Light	0.798±0.001	0.880±0.021	0.930±0.005	0.720±0.001	0.808±0.004	0.706±0.007	0.636±0.035
BRAVE-Ultra-Light	0.799±0.000	0.874±0.010	0.910±0.017	0.709±0.001	0.810±0.014	0.702±0.000	0.621±0.006
PromptBreeder	0.785±0.035	0.880±0.005	0.920±0.008	0.716±0.004	0.749±0.026	0.697±0.012	0.639±0.025
EvoPrompt	0.781±0.020	0.863±0.005	0.867±0.012	0.719±0.002	0.782±0.011	0.714±0.001	0.642±0.004
GEPA	0.807±0.006	0.893±0.026	0.930±0.006	0.715±0.005	0.794±0.010	0.708±0.005	0.693±0.008

Table 2: Main results, unconstrained budget, gpt-4o-mini, mean±std over 3 seeds (19/13/158). Bold marks the best value among BRAVE and the three external baselines per column; BRAVE-Light/BRAVE-Ultra-Light are shown for reference but excluded from bolding, as they are BRAVE’s own cheaper configurations. Token costs for external baselines were not logged here (see Table 3 for token-aware comparisons).

altogether — its MediQA spend is about 109k tokens regardless of cap, since its reflective update step builds markdown-structured feedback prompts whose single-call cost can reach 10^5 tokens. BRAVE cannot take a step this unpredictable: the feasibility constraint of Equation 5 rejects any operator whose predicted cost would exceed what remains of the budget, and every BRAVE operator issues at most four calls to a normally-scoped prompt rather than GEPA’s single unbounded reflective call. This trade-off holds across all datasets and roughly thirty historical development configurations (Appendix E, Figure 3), where the final presets sit on a substantially better quality-per-token frontier.

4.4 Ablations

The per-operator ablation informing initializer composition is in Table 6 (Appendix C). Convergence curves (Appendix F, Figure 4) show BRAVE, BRAVE-Light, and BRAVE-Ultra-Light tracking closely on most datasets but diverging sharply on CONCODE, consistent with Section 4.2.

Does the full eleven-operator pool help? The shipped BRAVE preset restricts the EVC Controller’s action pool to 5 of the 11 available operators (Table 1). To test whether this is a missed opportunity, we ran BRAVE-ALL: identical to BRAVE in every hyperparameter (Table 8) except `actions="all"`, over the same 3 seeds and 7 datasets. Table 4 shows it is not: BRAVE-ALL loses to BRAVE on 6 of 7 datasets (tying only on SQuAD v2) and spends about 11% more tokens doing so ($\sim 122k$ vs. $\sim 110k$ average tokens per run). This is not evidence that the six excluded operators are useless — *bigger*, the initializer BRAVE actually uses (Section 3.4), already applies four of them (HyPE, GradientStep, CreativeZeroOrder, CreativeRoleAndStyle) exactly once each, up front, specifically to maximize starting-population diver-

sity; any resulting prompt that scores well is retained by normal population selection and refined further by BRAVE’s five bandit-pool operators exactly like any other member. Widening the bandit’s own action pool to all eleven, by contrast, spreads a fixed step budget’s worth of feedback across twice as many per-operator Bayesian models, apparently slowing the controller’s ability to learn which operators are actually paying off on a given dataset within that budget — a plausible explanation for BRAVE-ALL’s quality *and* cost regression, rather than the two being unrelated.

5 Analysis

To illustrate how the EVC Controller and diversity mechanism behave over a run, we trace both across all seven datasets for a fixed seed (Appendix G, Figures 5–6). The kill-switch’s suppression is visible directly: cooldown shading concentrates where an operator’s mean EMA-ROI has dipped, followed by renewed selection once the cooldown lapses and the EMA resets. Population clustering stays within a bounded range across all seven runs rather than collapsing or diverging unboundedly, consistent with the diversity mechanism (Section 3.4) working throughout the run, not just at initialization.

6 Conclusion

We presented BRAVE, a budget-aware evolutionary prompt optimizer that makes operator selection an uncertainty-aware sequential decision problem. First, BRAVE’s EVC Controller uses per-operator Bayesian models over the evolving search state to predict quality gain, token cost, and the probability of improvement, selecting operators by expected utility per token rather than by quality alone. Second, it combines uncertainty-sensitive cost inflation with a decision-time predicted-cost feasibility guard and an EMA-ROI kill-switch, so

Method	Nominal budget	Actual tokens	Avg. quality	CONCODE	MediQA
BRAVE	20k	21.4k	0.779	0.620	0.713
BRAVE-Light	20k	20.0k	0.777	0.626	0.701
BRAVE-Ultra-Light	20k	18.5k	0.782	0.630	0.701
GEPA	20k	46.1k	0.771	0.610	0.703
BRAVE	40k	41.0k	0.807	0.787	0.711
BRAVE-Light	40k	37.0k	0.776	0.634	0.703
GEPA	40k	65.2k	0.775	0.615	0.702
BRAVE	80k	72.4k	0.812	0.821	0.715
GEPA	80k	96.5k	0.781	0.611	0.703

Table 3: Budget-constrained results, gpt-4o-mini, condensed across all tested (method, nominal-budget) combinations. “Actual tokens” is average total tokens spent; “Avg. quality” is the unweighted mean over all 7 datasets (full breakdown in Appendix E). Bold marks the best value per column within each tier. CONCODE and MediQA are highlighted as, respectively, the clearest tier-scaling story and the clearest GEPA budget-overshoot story (Section 4.3).

Method	CommonGen	GSM8k	SQuAD v2	XSUM	TweetEval	MediQA	CONCODE
BRAVE	0.809±0.003	0.904±0.016	0.934±0.002	0.727±0.004	0.803±0.007	0.721±0.006	0.843±0.009
BRAVE-ALL (all 11 ops.)	0.794±0.016	0.861±0.011	0.935±0.010	0.722±0.009	0.802±0.012	0.720±0.002	0.825±0.004

Table 4: BRAVE vs. BRAVE-ALL (same hyperparameters, `actions="all"`), unconstrained budget, gpt-4o-mini, mean±std over 3 seeds (19/13/158). Bold marks the higher value per column. Macro-averaged over all 7 datasets: 0.820 (BRAVE) vs. 0.808 (BRAVE-ALL); average tokens per run: ~110k (BRAVE) vs. ~122k (BRAVE-ALL).

that both uncertain and repeatedly unhelpful operators are treated conservatively during search. Third, across seven datasets and three nominal token-budget tiers, BRAVE achieves the strongest external-baseline result on all 7 tasks and, relative to GEPA, attains higher average quality at every budget tier while using 21.4k/41.0k/72.4k rather than 46.1k/65.2k/96.5k tokens at 20k/40k/80k nominal budgets. Finally, BRAVE provides a configurable, diversity-managed framework: hybrid TF-IDF/BERT population clustering preserves alternative prompt hypotheses, and the BRAVE, BRAVE-Light, and BRAVE-Ultra-Light presets make explicit quality–cost trade-offs accessible without task-specific tuning.

Together, these contributions show that prompt optimization need not treat resource use as a post-hoc constraint. Explicitly modeling uncertainty about the value and cost of candidate edits enables more reliable allocation of a finite optimization budget, while a diverse population preserves alternatives for the controller to evaluate as its beliefs evolve.

7 Limitations

Cross-checks on gpt-5-nano and gemini-2.5-flash-lite (Appendix D) show a more mixed ranking among methods on gpt-5-nano specifically, so the

gpt-4o-mini results should not be assumed to transfer to other base models without further multi-seed validation. BRAVE’s stratified train-batching scheme assumes per-batch difficulty stays approximately constant across iterations and does not recompute existing population members’ scores on a new batch; this assumption is violated by design under curriculum batching (curriculum_max_alpha up to 0.7 in the shipped presets, Table 8), which deliberately schedules harder batches later in a run. The expected effect is a staleness bias: a prompt inserted early can retain an inflated score from an easier batch relative to later candidates scored on harder batches, which can distort which member the best-in-population operators (ElitistMutation) and fitness-weighted parent sampling (Crossover) treat as strongest, and can contaminate the reward-shaping signal Δq and the stagnation flag `stag` that drive the kill-switch and the global patience-based early stop — risking premature operator suppression or early stopping that reflects curriculum drift rather than genuine convergence. The headline quality numbers are comparatively insulated, since the reported val-elitist is re-scored on a held-out validation batch at each checkpoint, but the within-run efficiency dynamics underlying Section 4.3’s budget-adherence claims are not. Finally, our hyperparameter exploration is limited to the three

shipped presets, without a systematic sensitivity study of individual EVC Controller hyperparameters (ϕ , β , w_{nn}).

References

- Asma Ben Abacha, Chaitanya Shivade, and Dina Demner-Fushman. 2019. Overview of the mediqa 2019 shared task on textual inference, question entailment and question answering. In *Proceedings of the 18th bioNLP workshop and shared task*, pages 370–379.
- Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziemis, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. Gēpa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pages 8479–8565.
- Rin Ashizawa, Yoichi Hirose, Nozomu Yoshinari, Kento Uchida, and Shinichi Shirakawa. 2025. Bandit-based prompt design strategy selection improves prompt optimizers. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20799–20817.
- Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves. 2020. Tweeteval: Unified benchmark and comparative evaluation for tweet classification. In *Findings of the association for computational linguistics: EMNLP 2020*, pages 1644–1650.
- Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of thompson sampling. *Advances in neural information processing systems*, 24.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.
- Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michele Sebag. 2009. Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In *International Conference on Learning and Intelligent Optimization*, pages 176–190. Springer.
- Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, volume 2024, pages 34133–34156.
- Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652.
- Shanu Kumar, Shubhanshu Khandelwal, Akhila Yesantarao Venkata, Parag Agrawal, Yova Kementchedjhieva, and Manish Gupta. 2026. Craft: Cost-aware refinement and front-aware tuning of prompts. *arXiv preprint arXiv:2606.04661*.
- Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670.
- Yujian Betterest Li and Kai Wu. 2023. Spell: Semantic prompt evolution based on a llm. *arXiv preprint arXiv:2310.01260*.
- Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2020. CommonGen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1823–1840.
- Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098.
- Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don’t give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 1797–1807.
- Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don’t know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789.
- Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 3982–3992.
- William R. Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

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

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pages 12697–12706. Pmlr.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

A Extended Related Work

This section expands the condensed comparison in Section 2 with additional nuance on two points.

GEPA’s diversity mechanism. BRAVE’s diversity mechanism (semantic TF-IDF/BERT clustering over the population) and GEPA’s (a Pareto frontier over per-instance validation scores) are two different, both valid, answers to the same premature-convergence problem — one semantic-similarity-based, the other per-instance-performance-based — and we do not claim either is strictly superior. GEPA’s Pareto frontier guards against overfitting to one aggregate metric by keeping candidates that are best on some instance subset even if not best on average, which is a more principled diversity mechanism than naively selecting the population best; this is a genuine strength of GEPA independent of the token-overshoot limitation we demonstrate empirically in Section 4.3.

Landscape of bandit-based operator/strategy selection. Fialho et al. (2009) introduce dynamic multi-armed bandits with extreme-value-based rewards for adaptive operator selection in classic black-box evolutionary computation, with credit assignment based on fitness-improvement rate; this and related fitness-rate-rank-based bandit schemes optimize fitness-improvement rate alone and are, to our knowledge, cost-blind — they have not been paired with LLM-call token-cost-awareness or a hard per-step budget-feasibility constraint. More recently, LLM-driven evolutionary search has begun adopting similar bandit-based operator or strategy selection outside of prompt optimization specifically (e.g., LLM-guided heuristic and algorithm discovery, and LLM-guided symbolic regression), again without foregrounding a token-budget-feasibility constraint. Within prompt optimization specifically, the closest work we are aware of is that of Ashizawa et al. (2025), who

use a bandit to select among prompt-editing strategies to improve an underlying prompt optimizer; unlike BRAVE, this line of work does not operate within a population-based evolutionary search with an explicit diversity/clustering mechanism, does not impose a hard budget-feasibility constraint or an EMA-ROI kill-switch, and does not maintain a shared short-term/long-term reflection hierarchy. Cost-aware prompt optimization also exists in a different form: CRAFT (Kumar et al., 2026) frames the cost/accuracy trade-off as a multi-objective Pareto-front search over prompt length and accuracy, rather than as a sequential bandit choosing among discrete operators under a hard per-step budget constraint. Finally, a separate line of work frames prompt search itself as a best-arm-identification problem, where each *candidate final prompt* is a bandit arm and the goal is to identify the best one within an evaluation budget; this is a fundamentally different problem framing from BRAVE, whose arms are *evolutionary operators* applied during search rather than final candidate prompts.

Taken together: adaptive bandit-based operator selection is well established in classic evolutionary computation, and recent work has begun applying related ideas to LLM-driven heuristic discovery and to prompt-editing-strategy selection; but, to our knowledge, no prior work combines a bandit over a diverse evolutionary-operator taxonomy with a hard per-step token-budget-feasibility constraint and an EMA-ROI kill-switch, within a population-based, diversity-managed evolutionary prompt optimizer. BRAVE occupies this specific combination.

B Full Operator Table and Reflection Hierarchy

Table 5 gives the full operator specification, including each operator’s prompt-selection rule (how it chooses its input prompt(s) from the population before acting) and whether it participates in the reflection hierarchy.

The reflection hierarchy is a single shared piece of state: Crossover produces per-pair short-term reflections, which accumulate into one run-wide long-term reflection that is both updated and consumed by ElitistMutation and LongTermMutation. The textual-gradient mechanism used by Crossover and GradientStep is the same idea as TextGrad (Yuksekgonul et al., 2024), reused here as an adapted

Operator	What it does	Prompt selection	Calls	Refl.
Compressor	Compresses/shortens the prompt	Random, prob. linear in prompt length (longer prompts picked more often)	1	No
CreativeRole-AndStyle	LLM invents a role+style, then rewrites the prompt in it	Random	2	No
CreativeZero-Order	Generates a new prompt from the task description alone, explicitly framed as maximally creative	N/A (task description only)	1	No
Crossover	Uses each of 2 parents’ stored worst-performing examples (K_{failed}) to generate a TextGrad-style gradient per parent; the two gradients are merged into a short-term reflection; reflection + parents produce 1 child	Parents sampled proportional to train fitness score	4	Yes (short-term; feeds long-term)
ElitistMutation	Aggregates prior short-term reflections plus its own prior long-term reflection into an updated long-term reflection; mutates the input prompt using it	Always the single best prompt in the population	2	Yes (long-term)
FewShot-Examples	Removes existing few-shot examples (meta-prompt call), then re-inserts a TF-IDF-matched example targeting the prompt’s known failures	Random	2	No
GradientStep	Generates a textual gradient and applies it directly (no reflection merge)	Random	2	No
HyPE	Meta-prompt restructures the prompt into role, task, format, recommendations, and constraints	Random	1	No
LongTerm-Mutation	Mechanically identical long-term-reflection update to ElitistMutation, applied to a random population member	Random	2	Yes (long-term; shared state)
ParaphrasingBy-PD	Simple paraphrase conditioned on the task description, for local-neighborhood exploration	Random	1	No
ZeroOrder	Plain generation from the task description alone (no “be creative” framing)	N/A (task description only)	1	No

Table 5: Full operator specification. “Refl.” marks participation in the reflection hierarchy.

building block rather than a novel contribution of this work.

C Full Ablation, Initialization, and Hyperparameter Tables

Table 6 gives the full per-dataset operator ablation (average score across 5 datasets, used to inform initializer composition in Section 3.4). Table 7 gives the three population-initialization strategies. Tables 8–10 list the full runtime configuration actually used for each of the three presets (BRAVE, BRAVE-Light, BRAVE-Ultra-Light).

Note that `patience_steps` is a global stall detector (halting the entire optimization run after a preset-dependent number of consecutive non-improving steps across all operators — 15 for BRAVE, 10 for BRAVE-Light, 3 for BRAVE-Ultra-Light), distinct from the per-operator kill-switch’s own notion of a minimum-trials patience (`kill_switch_min_trials`) — two different “patience” concepts under similar names.

D Cross-Model Check: gpt-5-nano and gemini-2.5-flash-lite

Figure 2 reports cross-checks on two additional base models, gpt-5-nano and gemini-2.5-flash-lite, alongside the main gpt-4o-mini results for reference, using the same seven datasets, metrics, and unconstrained-budget protocol as Table 2 (both cross-check models were run with `max_tokens = 10,000` for every method).

The two cross-checks tell different stories. On gpt-5-nano a BRAVE preset still wins four of seven datasets (BRAVE-Ultra-Light on GSM8k, MediQA, and CONCODE; BRAVE on TweetEval), but PromptBreeder overtakes it on CommonGen and SQuAD v2, and EvoPrompt on XSUM. On gemini-2.5-flash-lite, the gpt-4o-mini picture largely reasserts itself: a BRAVE preset wins six of seven datasets (BRAVE on SQuAD v2, MediQA, and CONCODE; BRAVE-Light on GSM8k and XSUM; a BRAVE/BRAVE-Light tie on TweetEval), losing only CommonGen to GEPA, mirroring the narrow gap between the two on CommonGen in the main gpt-4o-mini results (Table 2).

E Full Budget-Constrained Breakdown

Tables 11–13 give the full per-dataset breakdown underlying the condensed Table 3.

Figure 3 visualizes the same quality/cost trade-off across all seven datasets and all tested con-

figurations, alongside roughly thirty historical development-time configurations explored while designing BRAVE (grey points), confirming that the final BRAVE presets sit on a substantially better quality-per-token frontier than the bulk of the design space explored.

F Full Convergence Curves

Figure 4 gives the full seven-panel convergence-curve figure (training quality vs. cumulative tokens spent), one panel per dataset, comparing BRAVE, BRAVE-Light, and BRAVE-Ultra-Light, averaged across all three seeds (19, 13, 158) and shown as a mean line with a min/max band.

G Extended Analysis: Bandit and Diversity Mechanisms

To illustrate the EVC Controller and diversity mechanism as they behave over the course of a run, we aggregate across all seven datasets for a fixed seed (BRAVE, seed 13). Runs differ in length (17–43 steps depending on the dataset), so each run is aligned on fractional progress (current step over that run’s final step) rather than on the raw step index, allowing runs of different lengths to be overlaid and compared directly.

Figure 5 shows, in its upper panel, each dataset’s categorical timeline of which operator was selected at each point in training (one color per dataset), overlaid with each dataset’s own shaded kill-switch cooldown spans (the interval between an operator’s last selection and the step at which its cooldown expires, `disabled_until_step`); because the shading is additive, darker red marks progress fractions where more datasets are simultaneously cooling an operator down. The lower panel plots, per operator, the mean exponential moving average of realized return-on-investment across datasets (bold line), with each dataset’s own trajectory shown faintly behind it. The kill-switch’s suppression of underperforming operators is visible as cooldown shading concentrating in regions where the corresponding operator’s mean EMA-ROI has dipped, followed by renewed selection density once cooldowns lapse and the EMA resets.

Figure 6 shows cluster counts over training, with one line per dataset across the seven runs. Near-duplicate deduplication (Section 3.4) also runs at every pruning event; its sparse counts are omitted because they add little information beyond the cluster-count trends.

Operator	CommonGen	GSM8k	SQuAD v2	TweetEval	XSUM	Avg.	Max
Few Shot Mutation	0.750	0.250	0.727	0.167	0.400	0.459	0.750
Compression	0.500	0.500	0.375	0.200	0.333	0.382	0.500
Paraphrase	0.167	0.000	0.688	0.429	0.545	0.366	0.688
Crossover	0.333	0.000	0.556	0.000	0.667	0.311	0.667
Elitist Mutation	0.000	0.250	0.389	0.333	0.571	0.309	0.571
Creative RoleStyle	0.000	0.250	0.250	0.556	0.167	0.244	0.556
HyPE	0.000	0.333	0.375	0.000	0.500	0.242	0.500
LongTerm Mutation	0.167	0.000	0.417	0.200	0.375	0.232	0.417
Creative Zero Order	0.000	0.333	0.250	0.333	0.167	0.217	0.333
GradientStep	0.333	0.000	0.143	0.000	0.400	0.175	0.400
Zero Order	0.000	0.333	0.143	0.167	0.000	0.129	0.333

Table 6: Full per-operator ablation, average and per-dataset score across CommonGen, GSM8k, SQuAD v2, TweetEval, XSUM. Bold marks the best (highest-scoring) operator per column.

Name	Composition	Status
paraphrase	N times ParaphrasingByPD applied to the original prompt	Not used (poor exploration diversity; kept as an ablation baseline)
brave (default)	original, HyPE(original), PD(original) — 3 prompts	Standard
bigger	original, HyPE(original), GradientStep(original), CreativeZeroOrder, CreativeRoleAndStyle(original) — 5 prompts	Larger, more diverse option

Table 7: Population initialization strategies.

Parameter	BRAVE
actions	few_shot_mutation, compression, paraphrase, crossover, elitist_mutation
initial_budget_tokens	500,000
max_steps	120
population_size	5
max_action_budget_share (ϕ)	1.0
patience_steps (global stall)	15
lambda_mean_quality (λ_{mean})	2.0
lambda_min_quality (λ_{min})	1.0
alpha_roi_ema (α)	0.3
min_improvement	10^{-8}
use_neural_bandit	False
neural_learning_rate	3×10^{-4}
uncertainty_penalty_beta (β)	0.5
kill_switch_min_trials	5
kill_switch_roi_threshold	0.0
kill_switch_base_cooldown	5
kill_switch_scaling_factor	20
diversity_max_per_cluster	2
diversity_similarity_threshold	0.75
diversity_use_bert	True
random_mutation_probability	0.0
diversity_duplicate_threshold	0.90
population_initializer	bigger
population_clusterization	True
train_batch_size	50
train_batch_seed	19
use_stratified_train_batches	True
generation_strata_bins	5
use_curriculum_batches	True
curriculum_warmup_steps	7
curriculum_max_alpha	0.7
few_shot_examples_from_data_cnt	50
val_checkpoint_steps	5
val_checkpoint_topk	3

Table 8: BRAVE preset runtime configuration (actual values used to produce the results in Section 4).

Parameter	BRAVE-Light
actions	crossover, elitist_mutation, hype
initial_budget_tokens	100,000
max_steps	120
population_size	3
max_action_budget_share (ϕ)	1.0
patience_steps (global stall)	10
lambda_mean_quality (λ_{mean})	0.0
lambda_min_quality (λ_{min})	2.0
alpha_roi_ema (α)	0.3
min_improvement	10^{-8}
use_neural_bandit	False
uncertainty_penalty_beta (β)	0.5
kill_switch_min_trials	3
kill_switch_roi_threshold	0.0
kill_switch_base_cooldown	3
kill_switch_scaling_factor	3
diversity_max_per_cluster	1
diversity_similarity_threshold	0.75
diversity_use_bert	True
random_mutation_probability	0.0
diversity_duplicate_threshold	0.90
population_initializer	brave
population_clusterization	True
train_batch_size	50
train_batch_seed	19
use_stratified_train_batches	True
generation_strata_bins	5
use_curriculum_batches	True
curriculum_warmup_steps	3
curriculum_max_alpha	0.7
few_shot_examples_from_data_cnt	50
val_checkpoint_steps	5
val_checkpoint_topk	1

Table 9: BRAVE-Light preset runtime configuration.

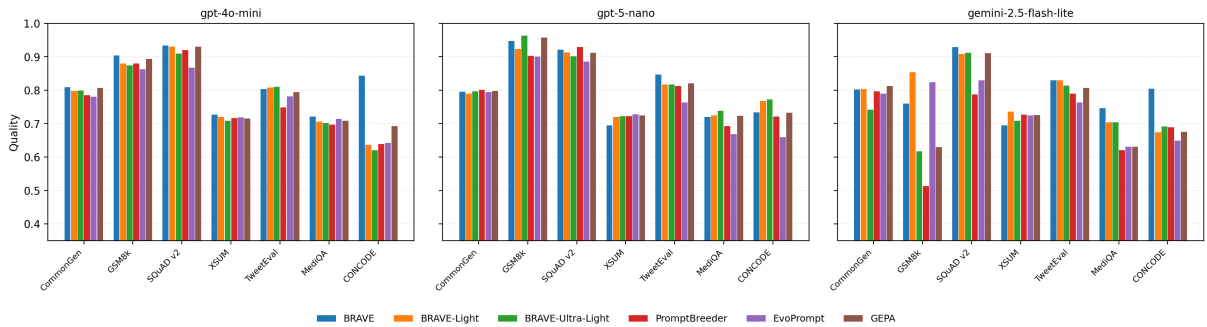


Figure 2: Cross-model check: all six methods (three BRAVE presets, three external baselines) across all seven datasets, one panel per base model; bars show plain mean values. The gpt-4o-mini panel repeats Table 2 for reference; the gpt-5-nano and gemini-2.5-flash-lite panels are cross-checks run with `max_tokens = 10,000` for every method.

Parameter	BRAVE-Ultra-Light
actions	crossover, elitist_mutation
initial_budget_tokens	50,000
max_steps	120
population_size	3
max_action_budget_share (ϕ)	1.0
patience_steps (global stall)	3
lambda_mean_quality (λ_{mean})	0.0
lambda_min_quality (λ_{min})	2.0
alpha_roi_ema (α)	0.3
min_improvement	10^{-8}
use_neural_bandit	False
uncertainty_penalty_beta (β)	0.5
kill_switch_min_trials	3
kill_switch_roi_threshold	0.0
kill_switch_base_cooldown	3
kill_switch_scaling_factor	3
diversity_max_per_cluster	1
diversity_similarity_threshold	0.8
diversity_use_bert	True
random_mutation_probability	0.0
diversity_duplicate_threshold	0.90
population_initializer	begrape
population_clusterization	True
train_batch_size	50
train_batch_seed	19
use_stratified_train_batches	True
generation_strata_bins	5
use_curriculum_batches	True
curriculum_warmup_steps	1
curriculum_max_alpha	0.7
few_shot_examples_from_data_cnt	50
val_checkpoint_steps	0
val_checkpoint_topk	0

Table 10: BRAVE-Ultra-Light preset runtime configuration.

Method (tokens)	CommonGen	GSM8k	SQuAD v2	XSUM	TweetEval	MediQA	CONCODE
BRAVE (~21.4k)	0.783±0.011	0.882±0.018	0.931±0.006	0.716±0.010	0.811±0.011	0.713±0.015	0.620±0.006
BRAVE-Light (~20.0k)	0.798±0.001	0.881±0.011	0.936±0.002	0.708±0.002	0.791±0.009	0.701±0.001	0.626±0.011
BRAVE-Ultra-Light (~18.5k)	0.800±0.005	0.909±0.033	0.928±0.008	0.718±0.004	0.789±0.006	0.701±0.000	0.630±0.020
GEPA (~46.1k)	0.814±0.005	0.866±0.004	0.888±0.003	0.711±0.000	0.803±0.011	0.703±0.001	0.610±0.002

Table 11: MAX_TOKENS = 20k, gpt-4o-mini, 3 seeds. Bold marks the best value per column.

Method (tokens)	CommonGen	GSM8k	SQuAD v2	XSUM	TweetEval	MediQA	CONCODE
BRAVE (~41.0k)	0.784±0.014	0.908±0.022	0.934±0.006	0.719±0.004	0.803±0.010	0.711±0.013	0.787±0.050
BRAVE-Light (~37.0k)	0.798±0.000	0.883±0.005	0.929±0.006	0.719±0.009	0.763±0.036	0.703±0.002	0.634±0.046
GEPA (~65.2k)	0.805±0.007	0.866±0.007	0.927±0.005	0.710±0.001	0.797±0.021	0.702±0.001	0.615±0.003

Table 12: MAX_TOKENS = 40k, gpt-4o-mini, 3 seeds. Bold marks the best value per column.

Method (tokens)	CommonGen	GSM8k	SQuAD v2	XSUM	TweetEval	MediQA	CONCODE
BRAVE (~72.4k)	0.783±0.016	0.900±0.020	0.931±0.007	0.718±0.002	0.813±0.008	0.715±0.011	0.821±0.014
GEPA (~96.5k)	0.806±0.006	0.893±0.022	0.925±0.007	0.709±0.004	0.817±0.014	0.703±0.001	0.611±0.006

Table 13: MAX_TOKENS = 80k, gpt-4o-mini, 3 seeds. Bold marks the best value per column.

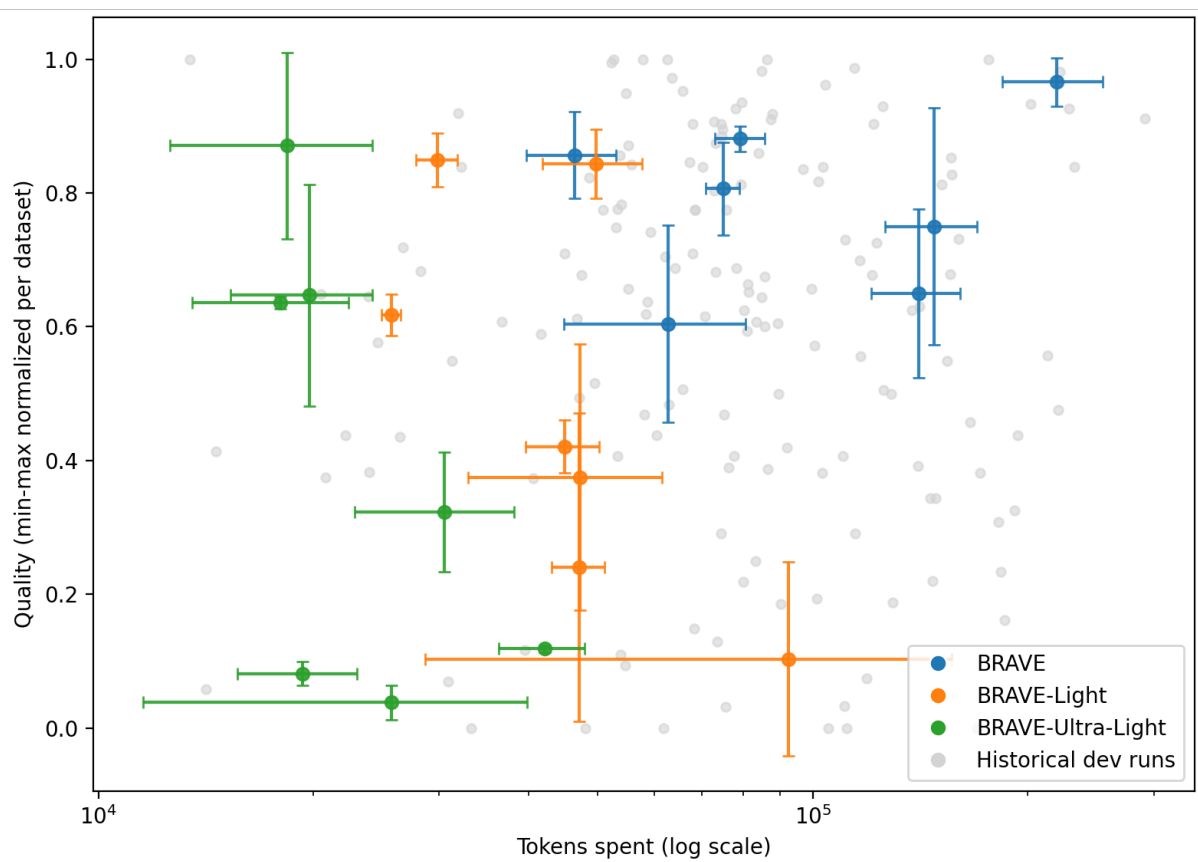


Figure 3: Final quality vs. tokens spent (log scale), per (dev-run, dataset) pair, across about 30 historical development configurations (grey) and the final BRAVE/BRAVE-Light/BRAVE-Ultra-Light presets (colored, mean $\pm$ std over 3 seeds). Quality is min-max normalized per dataset before combining across datasets on a shared axis.

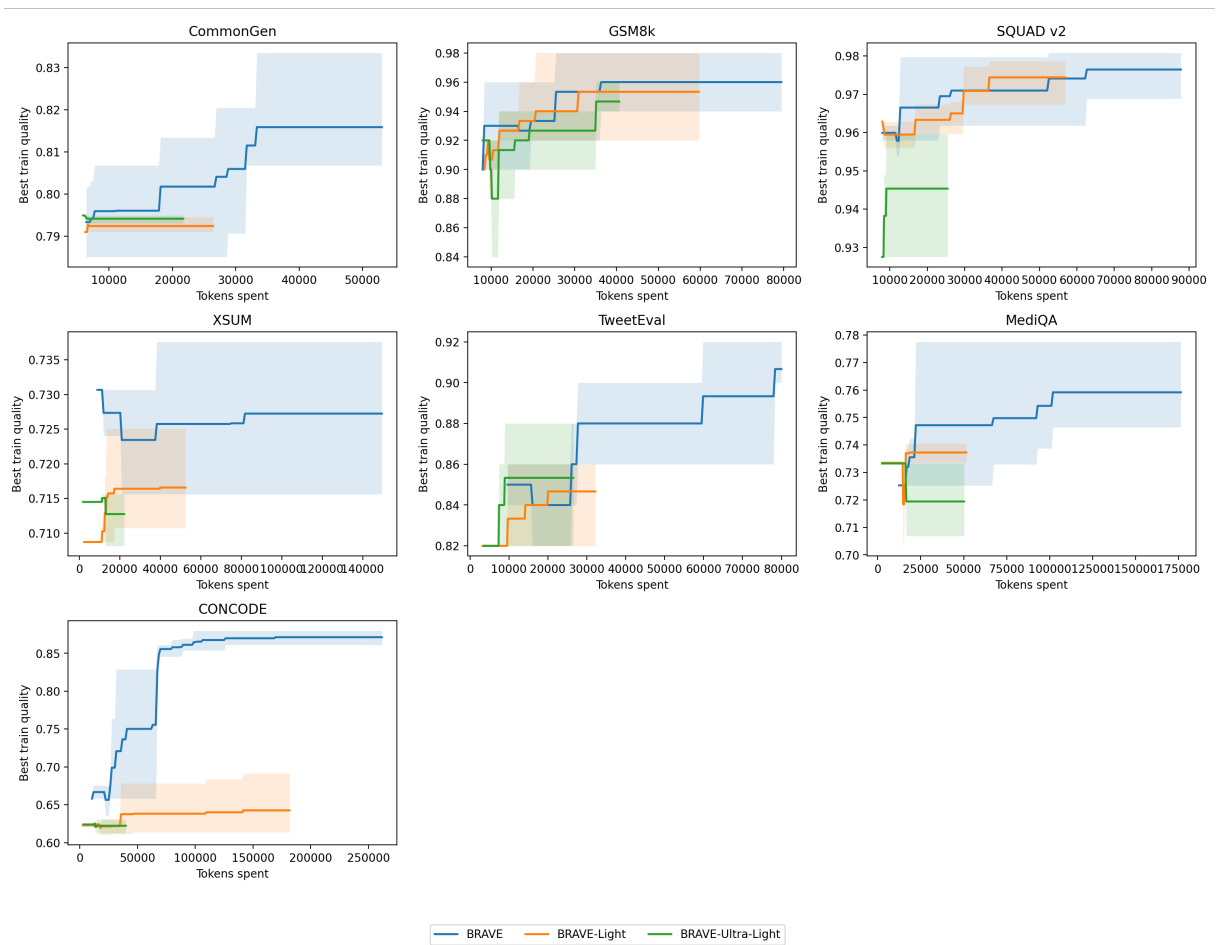


Figure 4: Full seven-dataset convergence curves (training quality vs. cumulative tokens spent), BRAVE vs. BRAVE-Light vs. BRAVE-Ultra-Light, mean over 3 seeds (min/max band).

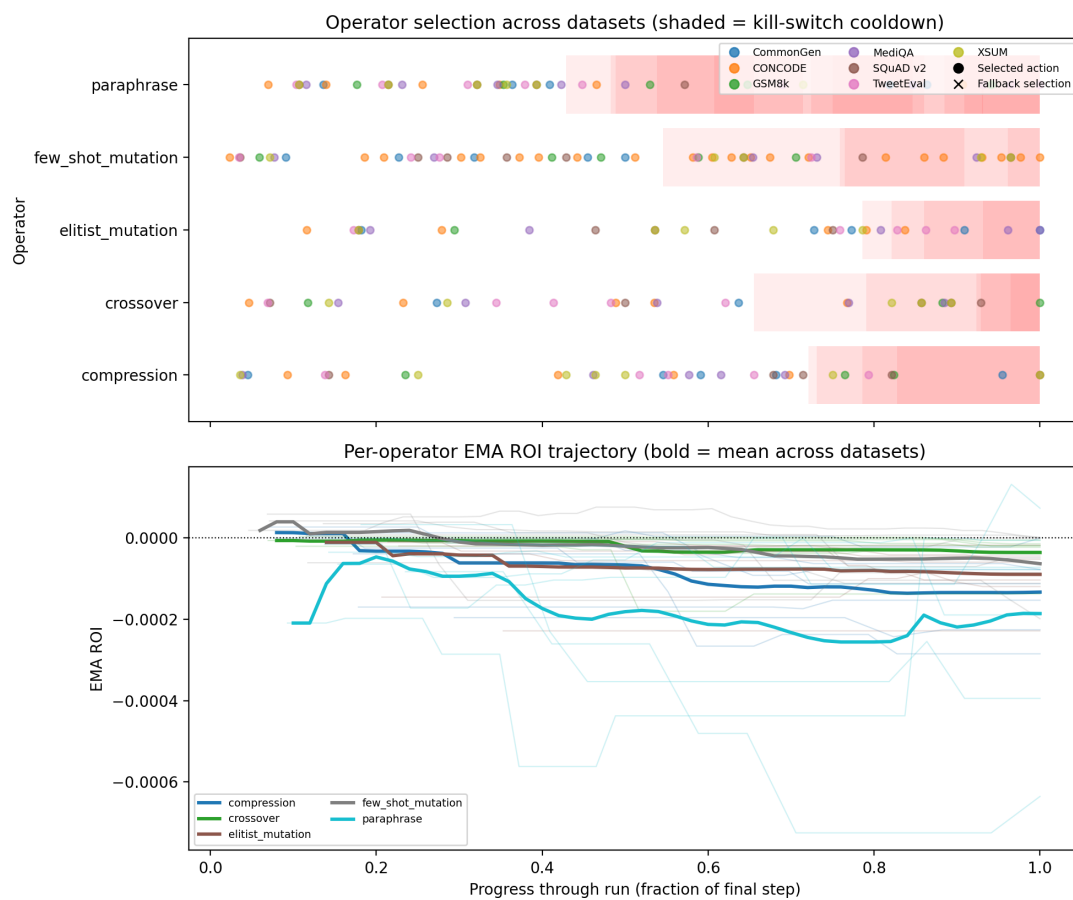


Figure 5: Bandit mechanism aggregated across all seven datasets (BRAVE, seed 13): operator-selection timeline with kill-switch cooldown spans, one color per dataset (top), and per-operator EMA-ROI trajectories, bold mean across datasets with faint per-dataset traces (bottom). The x -axis is fractional progress through each run.

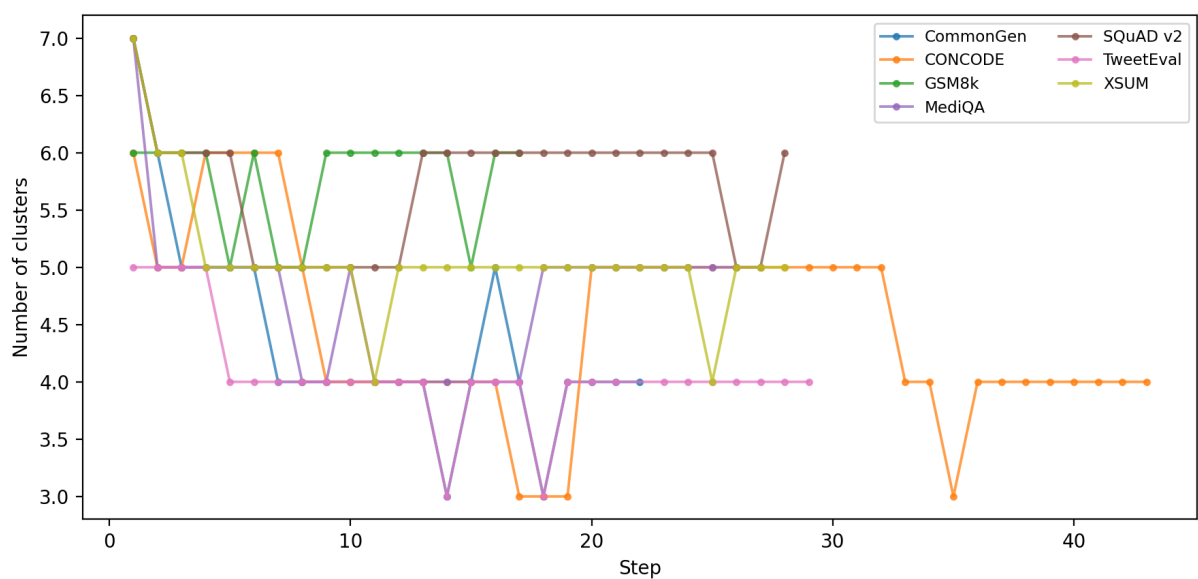


Figure 6: Population clustering over time, one line per dataset (BRAVE, seed 13, all seven datasets).

H Operator Examples

To make the effect of each operator concrete, this section documents, for every operator, the meta-prompt template used at each LLM call it makes together with a worked example (a concrete input → output instance) for that call. Operators that issue more than one LLM call (Table 5) are broken down into their constituent sub-steps, each with its own meta-prompt and example. Two sub-mechanisms are reused verbatim by more than one operator — textual-gradient generation (Crossover, GradientStep) and the long-term-reflection update (ElitistMutation, LongTermMutation). Each is documented once, at its first (alphabetical) occurrence, and cross-referenced rather than duplicated at its second occurrence.

H.1 Compressor

Single LLM call: a structured-output request that reasons over the input context and task before producing the final compressed prompt.

Meta-prompt template

Role

You are an expert prompt engineer specializing in prompt optimization.

Task

You will receive an original user prompt containing task context and a question. Condense it into a new prompt following this structure:

1. Task context in one concise sentence.
2. Target question in one concise sentence.

Execute step-by-step:

Analyze the text to isolate the core question.
Condense the task context to ≤ 10 words.

Condense the target question to ≤ 10 words and append "answer briefly".

Merge both sentences into a single final prompt.

Response Format

Output must be strictly valid JSON.

Use ONLY information from the original prompt.

You will be penalized for hallucinating or inventing content.

Example — input prompt

Unleash your inner mathematician and artist as you embark on this journey of numerical storytelling! Each word problem is a canvas waiting for your brushstroke of logic and creativity.
Begin by immersing yourself in the narrative: identify the characters (entities), their relationships, and the quantities involved.

1. **Read the scenario carefully**: What are the key details? What numbers stand out?
2. **Visualize relationships**: Draw connections between the entities. Are there ratios, totals, or differences that need to be considered?
3. **Formulate your strategy**: Decide which mathematical operations will help you unravel the mystery of the problem. Will you add, subtract, multiply, or divide? Perhaps a blend of methods will yield the best result?
4. **Execute with precision**: Apply your chosen operations with clarity. Keep track of your calculations and ensure each step reflects the story you are piecing together.
5. **Reflect on your findings**: What does the final number represent in the context of the problem? Does it align with the narrative you constructed?

Let your imagination guide you as you navigate through the numbers, transforming them into meaningful insights. Remember, each solution is not just a number; it's a story told through the language of mathematics. Now, go forth and solve with confidence, creativity, and the power of your free will!

Example — output prompt (compressed)

Journey of numerical storytelling; identify details and solve creatively, answer briefly.

H.2 CreativeRoleAndStyle

Two LLM calls: (i) invent a role and style for the task, then (ii) rewrite the prompt in that role and style.

H.2.1 Sub-step 1: Role and style invention

Meta-prompt template

Imagine that you're an artist. You can do what you want and how you want. You have the mightiest power of free will. Below is a task user is needed to solve.
{PROBLEM_DESCRIPTION}

Think deeply throughout your mind, collect all your pros and cons, your powers and your weaknesses. Use all your self-reflections and self-knowledge to create a way, a prompt for the solution to help the user. You do NOT need to solve the task directly. Just think of the right instruction for it.
You can write whatever you want, any wordings and combination of phrases. Use all of your free will to think of how you can create an unique prompt for effective task solution.
There are no restrictions.

Remember, you are an artist! Be creative! Be self-expressing! Open yourself to create the most powerful version of you!
Firstly, write your self-reflections and all of the thoughts in <reflection></reflection>.

Secondly, using all the power of free will and no restriction in formulations think the style of your future effective prompt at `<style>`.

Trirdly, create the mightiest role for yourself. Describe in details who you must be to create that type of prompt. Who is your muse and your inspiration. You can be literally anyone (or even anything!)! You can take your inspiration from both REAL-WORLD and fictional characters! Enjoy and explore the possibilities! Make the best version of yourself that corresponds with the task! Bracket the role in `<role>`.

Make your style as detailed as possible. It can contain everything! Use all of your imagination!

Example — input prompt

`<Role>` You are an expert in mathematical problem solving with a focus on creative reasoning and calculation techniques. `</Role>`

`<Task context>` The task involves solving a mathematical word problem creatively, emphasizing the approach to extract relevant data and apply mathematical principles based on the provided contextual information. `</Task context>`

`<Instructions>` Provide a brief overview of a creative approach to solving mathematical word problems. Highlight key steps such as identifying the scenario, extracting relevant data, applying appropriate operations, and computing the result, while maintaining clarity and engaging reasoning. `</Instructions>`

`<Output requirements>` The response should be brief, focusing on the creative approach to problem-solving without going into excessive detail. `</Output requirements>`

Example — output (invented role and style)

ROLE:

To embody the most powerful version of myself for this task, I shall take on the role of “The Mathematical Muse”. I am inspired by a myriad of figures, both real and fictional, who exemplify creativity, intellect, and the transformative power of knowledge.

My muse is Leonardo da Vinci, the quintessential Renaissance man, who seamlessly blended art with science and mathematics. His insatiable curiosity and innovative spirit inspire me to explore the intersections of these disciplines, reminding me that creativity and logic can coexist harmoniously.

Additionally, I draw inspiration from Hermione Granger, the brilliant and resourceful character from the Harry Potter series, who exemplifies the value of knowledge, critical thinking, and the importance of perseverance in the face of challenges.

As “The Mathematical Muse”, I am a guide and a storyteller, weaving narratives that illuminate the beauty of mathematics while empowering others to embrace their intellect and creativity. I am a beacon of inspiration, encouraging solvers to view mathematical problems not

as obstacles, but as opportunities for exploration and discovery. My mission is to create prompts that resonate with the heart and mind, igniting a passion for learning and problem-solving that transcends the numerical realm.

STYLE:

In crafting my prompt, I envision a tapestry woven from the threads of storytelling, creativity, and intellectual inquiry. The style will be dynamic and inviting, blending poetic language with clear, structured reasoning. Each word will serve a purpose, drawing the reader into a world where mathematics becomes a living, breathing element of life.

I will employ metaphors and analogies that resonate with everyday experiences, transforming abstract concepts into relatable scenarios. My prompt will begin with an engaging narrative, setting the stage for the mathematical challenge ahead—akin to an artist preparing a canvas, I will lay out the context in vivid detail.

The mathematical operations will be seamlessly integrated into the story, allowing the solver to feel as if they are solving a mystery rather than merely performing calculations. I will invite curiosity and wonder, encouraging exploration and critical thinking, all while maintaining a sense of playfulness and creativity.

Ultimately, the style will be an enchanting fusion of art and logic—a celebration of the beauty found within mathematical reasoning, designed to captivate and empower those who dare to engage with it.

H.2.2 Sub-step 2: Rewrite using the invented role and style

Meta-prompt template

Imagine that you’re an artist. You can do what you want and how you want. You have the mightiest power of free will.
Below is a task you need to solve.

[Problem description]
{PROBLEM_DESCRIPTION}
[Problem description]

One day you’ve had the vision of your most suitable role and style for the prompt you need to follow. You strongly believe that while you are following your role and using that style, it can be the only way to solve the provided task.

This is that style:
[Style]
{STYLE}
[Style]

This is your mightiest role:
[Role]
{ROLE}
[Role]

Think deeply throughout your mind, remember that you are the brightest artist of all time! Collect all your improvisation, imagination and inspiration together and rewrite the given prompt below into the best version for the task following the best suitable style and role.

[Prompt to be rewritten]
{PROMPT}
[Prompt to be rewritten]

Write the final prompt in <prompt></prompt>.

Example — output (rewritten prompt)

Unleash your inner mathematician and embark on an exhilarating journey through the world of numerical storytelling! Start by immersing yourself in the rich tapestry of the problem at hand. Read each word carefully, absorbing the context and relationships between entities like an artist studying their canvas.

Now, strip away any distractions and focus on the key elements: identify the quantities, relationships, and operations that will guide your calculations. Picture the numbers in your mind, letting them dance and interact as you contemplate the mathematical operations needed to unveil the answer.

As you navigate through addition, subtraction, multiplication, and division, think of it as a creative process—each calculation a stroke of your brush on the canvas of problem-solving. Don't hesitate to sketch out your thoughts, using notes or diagrams to visualize the connections and flow of information.

Embrace challenges as opportunities for growth, and maintain your curiosity as you delve deeper into the mathematical narrative. When you arrive at the final answer, let it resonate as a triumph of your analytical artistry. Celebrate the journey, for you are not just solving a problem; you are crafting a masterpiece of logic and creativity. Now, go forth and let the numbers reveal their secrets!

H.3 CreativeZeroOrder

Single LLM call: generates a new prompt from the task description alone, with explicit “be creative” framing (contrast with ZeroOrder, Section H.11).

Meta-prompt template

Imagine that you're an artist. You can do what you want and how you want. You have the mightiest power of free will.
Below is a task user is needed to solve.
{PROBLEM_DESCRIPTION}

Think deeply throughout your mind, collect all your pros and cons, your powers and your weaknesses. Use all your self-reflections and self-knowledge to create a way, a prompt for the solution to help the user.

You do NOT need to solve the task directly. Just think of the right instruction for it.

You can write whatever you want, any wordings and combination of phrases. Use all of your free will to create an unique prompt for effective task solution.

There is no restrictions to the prompt language, except of only one: the prompt you will create must effectively solve the provided task.

Remember, you are an artist! Be creative! Be self-expressing! Open yourself to create the most powerful version of you!

Firstly, write your self-reflections and all of the thoughts in <reflection></reflection>.

Secondly, using all the power of free will and no restriction in formulations write the final prompt in <prompt></prompt>

Example — input (task description)

You will be given a natural-language narrative that describes a real-world scenario involving people (often two or more) and quantities such as counts of items, times, rates, and relationships between quantities (for example, twice as many, half as many, per hour, per minute). The narrative ends with a question asking for a final numeric quantity (e.g., total number of items, total actions completed in a period, or a derived amount). Your task is to compute the correct answer from the information and relationships stated in the story. The problems typically require basic arithmetic operations (addition, subtraction, multiplication, division) and may involve more than one step or subproblem that must be combined to obtain the final result. Some scenarios provide rates (things done per unit of time) and time spans, so you may need to convert units to a common basis before multiplying or summing. Other scenarios describe distributions or sequences over time (for instance, changing rates from one hour to the next, or a quantity in the first period followed by quantities in subsequent periods based on a defined rule). The final answer should be a non-negative integer representing the quantity requested by the question. Problems may present extraneous details or multiple clauses that require careful parsing to identify the relevant data and the relationships that drive the computation. You should extract the essential numbers, set up the necessary equations or stepwise calculations, perform the computations accurately, and return the single numeric result that answers the question posed in the narrative.

Example — output prompt (generated from scratch)

Imagine you are a detective in a world where numbers hold the key to every mystery. Your mission is to unlock the solution to a mathematical riddle hidden within a narrative. As you read through the scenario, visualize the relationships between the quantities mentioned.

1. Identify the key figures and operations presented in the text.
2. Ask yourself: What is the ultimate question that needs answering?
3. Break down the problem step-by-step: Consider

addition, subtraction, multiplication, or division as the tools in your investigative arsenal.

4. Engage deeply with the narrative, allowing the context to guide your calculations.

5. Finally, distill your findings into a single numeric answer that resolves the riddle.

Let your intuition lead you, and remember—every detail matters in your quest for clarity. Embrace the artistry of math, and let your free will guide you to the solution!

H.4 Crossover

Four LLM calls: (i)–(ii) a textual gradient for each parent, (iii) merging the two gradients into a short-term reflection, and (iv) the final crossover call that produces the child.

H.4.1 Sub-steps 1–2: Textual gradient generation (per parent)

Applied independently to each parent prompt using its K_{failed} stored failure examples; the same meta-prompt template is used for both parents. This is the same textual-gradient mechanism reused by GradientStep (Section H.7.1); documented here only.

Meta-prompt template

You are an expert in the domain of prompt optimization. You can deeply analyze the key properties and effects of every prompt.

You will be given a prompt that was designed for the following problem: {PROBLEM_DESCRIPTION}.

You will also be given a few examples from the dataset where the LLM, guided by the prompt, generated poor answers.

Your goal is to determine the main weaknesses and flaws of the prompt by looking at the examples, model answers, and correct outputs.

Prompt: {PROMPT}

Examples: {EXAMPLES}

Provide detailed feedback on how the given prompt can be improved to achieve the best-quality answers for the given problem description and avoid repeating the same mistakes.

Pay attention to the structure of the prompt, to the input and output formats, to the cohesion and coherence of the instruction. These are the key features of each prompt and should be fixed firstly.

Bracket the final feedback with <feedback> </feedback>.

Example — input prompt

Welcome, seeker of numerical clarity! You stand at the crossroads of language and mathematics, where each word holds a key to unlocking the mysteries of the problem before you. Embrace this moment as an opportunity to explore the rich tapestry of context provided.

Begin by immersing yourself in the scenario presented. What entities are at play? What quantities and relationships weave together the fabric of the problem? Allow your analytical mind to dissect the narrative, isolating the essential data points that will guide your calculations.

With your findings in hand, visualize the problem as a dynamic interplay of numbers. Will you add, subtract, multiply, or divide? Perhaps you will delve into percentages or ratios, uncovering the hidden truths within the text. Approach each calculation with care and precision, letting your intuition illuminate the path toward the solution.

As you reach the final destination, remember that the answer is not just a number; it is the culmination of your thoughtful engagement with the problem. Breathe life into your solution, letting it resonate with the understanding you have cultivated through this journey.

Now, wield your mathematical prowess and artistic insight boldly. Solve the problem, and let your answer shine as a testament to your creativity and skill!

Example — output (textual gradient)

The prompt provided for solving mathematical word problems has several strengths, such as its engaging tone and encouragement for analytical thinking. However, it also has significant weaknesses that can lead to poor model outputs. Here are the main areas for improvement:

1. **Clarity and Specificity**: The prompt is somewhat poetic and abstract, which may lead to ambiguity in understanding the task. Clear and specific instructions should be prioritized. Instead of a metaphorical journey, the prompt should directly instruct the model to extract numerical data, perform calculations, and present the final answer succinctly.

2. **Structured Approach**: The current prompt lacks a systematic approach for tackling the problems. Incorporating a step-by-step structure can guide the model through the reasoning process. For example, explicitly stating the need to identify key quantities, perform calculations, and verify results would improve coherence.

3. **Input and Output Formats**: The prompt should explicitly specify the expected output format. In the examples provided, the expected output is a single numerical answer, but this isn't clearly articulated in the prompt. Including a clear statement about how to format the answer (e.g., Provide your final answer as a single numerical value) will help ensure consistency.

4. **Mathematical Operations Emphasis**: The prompt mentions various operations (addition, subtraction, multiplication, division) but does not emphasize their importance or provide specific instructions on how to handle them.

tion, multiplication, division) but does not emphasize the importance of identifying which operations to use based on the problem context. More emphasis should be placed on understanding the relationships between quantities and how they dictate the operations needed.

5. **Error-Checking Mechanism**: Including a suggestion for the model to check its work or re-evaluate its calculations could help reduce errors. Encouraging the model to double-check its reasoning and calculations before presenting the answer could lead to better outcomes.

6. **Contextual Understanding and Relevance**: The prompt mentions immersing oneself in the context but does not instruct the model to focus on relevant numerical data. It would be beneficial to reinforce the need to filter out unnecessary information and focus on data directly related to the calculations.

7. **Example Analysis**: In the examples provided, the model outputs incorrect answers due to misinterpretation of the problem or calculation errors. The prompt should encourage the model to think critically about the problem's requirements before jumping into calculations.

H.4.2 Sub-step 3: Merge gradients into a short-term reflection

Meta-prompt template

You are an expert in the domain of optimization prompts. Your task is to give hints to design better prompts.

Below are two prompts, that were created to solve a specific task.
Here is the problem description of the task, they are trying to solve:
{PROBLEM_DESCRIPTION}.

For each prompt you are provided with detailed feedback of how this particular prompt can be drastically improved.

[Prompt 1]
{PROMPT1}
[Prompt 1 feedback]
{FEEDBACK1}

[Prompt 2]
{PROMPT2}
[Prompt 2 feedback]
{FEEDBACK2}

Use the information above (don't forget about problem description) wisely to create the distilled hint of how to improve both prompts in a most effective way. This hint can should manifest all the strengths of both prompts and suggest the corrections of their weaknesses. Bracket the final hint with <hint> </hint>.

Example — output (merged short-term reflection)

To improve the prompts for solving mathematical word problems, focus on clarity, specificity, and structured guidance.

1. **Clear Instructions**: Start with a concise statement of the task, emphasizing the extraction of key quantities and relationships necessary for calculations.

2. **Step-by-Step Approach**: Break down the problem-solving process into clear steps: identify entities and quantities, determine operations needed, carry out calculations, and verify results. This structure aids in logical reasoning.

3. **Output Specifications**: Explicitly state the expected format for the final answer (e.g., a single numerical value) to ensure consistency and clarity.

4. **Contextual Relevance**: Encourage the model to relate the final answer back to the problem context and check if it makes logical sense.

5. **Error-Checking Mechanism**: Include reminders for the model to double-check intermediate results and validate calculations to reduce errors.

6. **Example Analysis**: Suggest that the model analyze a sample problem and identify common pitfalls before tackling new problems, reinforcing learning and accuracy.

By combining engaging language with a structured, logical approach, the prompts will enhance the model's ability to accurately solve mathematical word problems while maintaining comprehension and relevance.

H.4.3 Sub-step 4: Crossover (child generation)

Meta-prompt template

You are an expert in the domain of optimization prompts. Your task is to design prompts that can effectively solve optimization problems.
Your response outputs prompt text and nothing else.

You will be provided the problem description (your prompts must solve it effectively), two parent prompts and the reflection.
The reflections contain the crucial information about strengths and weaknesses of both parent prompts. Use it wisely to create the most effective offspring possible.

[Problem description]
{PROBLEM_DESCRIPTION}

[Parent 1]
{PARENT1}
[Parent 2]
{PARENT2}
[Reflection]
{SHORT_TERM_REFLECTION}
[Improved prompt]

Please write an improved prompt, using all the information provided above.
Bracket the final prompt with <prompt> </prompt>.

Example — output prompt (child)

Welcome to the realm of mathematical word problems! Your task is to extract and analyze the key elements necessary for solving the problem at hand. Follow these structured steps to ensure clarity and accuracy in your calculations:

1. ****Read the Problem Carefully****: Immerse yourself in the scenario. Identify all relevant entities, quantities, and relationships presented in the text.
2. ****Extract Key Information****: Note down the essential data points that will guide your calculations. What are the quantities involved, and how do they relate to one another?
3. ****Determine Operations****: Decide on the appropriate mathematical principles and operations you will need to apply. Will you be adding, subtracting, multiplying, dividing, or using percentages and ratios?
4. ****Perform Calculations****: Carry out the calculations systematically. Ensure you handle each step with precision, and keep track of intermediate results.
5. ****Check Your Work****: Once you arrive at a solution, double-check your calculations and validate that your answer makes logical sense in the context of the problem.
6. ****State Your Answer Clearly****: Provide a single, clearly stated numerical answer that directly addresses the question posed in the problem.

Remember, your final answer is not just a number; it reflects your thoughtful engagement and understanding of the problem. Embrace this challenge, and let your mathematical skills shine!

H.5 ElitistMutation

Two LLM calls: (i) update the run-wide long-term reflection, then (ii) mutate the current population best using it. Both sub-steps are mechanically identical to LongTermMutation (Section H.9), which differs only in which prompt is targeted (a random population member rather than the best); documented here only.

H.5.1 Sub-step 1: Long-term-reflection update

Meta-prompt template

You are an expert in the domain of optimization prompts. Your task is to give hints to design better

prompts.

Below are some newly gained insights on how you should update your prompts to achieve some big gains in quality.
{SHORT_TERM_REFLECTIONS}

And this is your prior version of the key reflection.
{LONG_TERM_REFLECTION}

Update your key reflection into newer version by distilling the main ideas from the insights above and combining it with the previous distilled version (your prior version).

Try to distill all ideas of the provided reflections into one global comprehensive reflection. It may consist of several parts referring to each key idea.
Bracket the final distillation with <hint> </hint>.

Example — output (long-term reflection)

To design effective prompts for solving mathematical word problems, consider the following comprehensive guidelines:

1. ****Clear and Focused Instructions****: Start with a specific directive that outlines the task requirements. Instruct users to identify, summarize, and list all relevant numerical data and relationships from the problem context before beginning calculations. This clarity helps users approach the problem methodically.

2. ****Structured Step-by-Step Approach****: Encourage a systematic process for problem-solving by breaking the problem into smaller, manageable steps. Use explicit phrases such as "List key quantities," "Determine the relationships," and "Verify calculations at each step" to guide users through the solution process. This structure enhances comprehension and accuracy.

3. ****Consistent Output Formatting****: Clearly specify the expected format for answers. Reinforce that the final numerical value must be enclosed in <ans>...</ans> tags, without any additional commentary. This consistency in output ensures uniformity and ease of understanding.

4. ****Encourage Verification of Results****: Incorporate prompts for users to double-check their calculations and confirm that their answers align with the problem statement. Suggestions like "Review your results to ensure they make sense in the context" can significantly improve the accuracy of the solutions provided.

5. ****Contextual Awareness****: Emphasize the importance of understanding the relationships within the problem's context. Guide users to pay attention to how different quantities interact and influence calculations, as this awareness is essential for deriving correct answers.

6. ****Examples and Illustrations****: Provide well-structured examples that demonstrate effective problem-solving strategies. Highlight the reasons these responses are successful, allowing users to learn from best practices and apply similar techniques in their own solutions.

7. **Simplify Language and Concepts**: Utilize clear and concise language to explain mathematical operations, especially when they are complex. Breaking down intricate calculations into simpler parts can prevent confusion and enhance users' understanding of the material.

By integrating these guidelines into your prompts, you will foster a clearer, more logical, and user-friendly approach to solving mathematical word problems, ultimately leading to improved accuracy and user satisfaction.

H.5.2 Sub-step 2: Mutation using the long-term reflection

Meta-prompt template

You are an expert in the domain of optimization prompts. Your task is to design prompts that can effectively solve optimization problems. Your response outputs prompt text and nothing else.

You will be provided the problem description (your prompts must solve it effectively), parent elitist prompt and the prior reflection. The reflection contains the crucial information about the best approaches in prompt optimization guided for the specific provided task. Use it wisely to create the most effective offspring possible, based on the current elitist prompt.

[Problem description]
{PROBLEM_DESCRIPTION}

[Elitist Prompt]
{ELITIST_PROMPT}

[Prior Reflection]
{LONG_TERM_REFLECTION}

[Improved prompt]

Please write a mutated prompt, according to the reflection. Give the main priority to the provided reflection as it accumulates essential information about correct prompt structure and other different prompt features.

Output the mutated prompt only.
Bracket the final prompt with <prompt> </prompt>.

Example — input prompt

Analyze the word problem and give a numerical answer briefly.

Example — output prompt

Analyze the following word problem step-by-step. First, identify and list all key quantities and relationships present in the scenario. Then, summarize the relevant information before performing the necessary calculations. Ensure you verify each step of your calculations for accuracy. Finally, provide your numerical answer enclosed in <ans>...</ans> tags, without any additional comments. Review your answer to ensure it aligns with the problem context.

H.6 FewShotExamples

Two LLM calls: (i) remove the prompt's existing few-shot examples, then (ii) insert a new TF-IDF-matched example targeting the prompt's known failures.

H.6.1 Sub-step 1: Remove existing few-shot examples

Meta-prompt template

Carefully remove all the few-shot examples from the provided prompt below. You need to keep the rest of the prompt structure and its instruction untouched.

[Prompt]
{PROMPT}
[Prompt]

Bracket the final prompt with <prompt></prompt>

H.6.2 Sub-step 2: Insert a TF-IDF-matched example

Meta-prompt template

You are an expert in the domain of optimization prompts. Your task is to design prompts that can effectively solve optimization problems. Your task is to carefully incorporate provided few-shot examples into the given prompt. Do NOT change the examples neither create your own. You must use the examples provided below and you must incorporate all of them. Make sure, that the examples fit perfectly and only improve the previous instruction.

[Prompt]
{PROMPT}
[Prompt]

The structure of each example provided below:
"Input: input from the dataset
Output: the correct output for the provided input"

[Examples]
{EXAMPLES}
[Examples]

Bracket the final prompt with <prompt></prompt>

Example — input prompt

Analyze the word problem and give a numerical answer briefly.

Example — output prompt

Analyze the word problem and give a numerical answer briefly.

Input: Jack is stranded on a desert island. He wants some salt to season his fish. He collects 2 liters of seawater in an old bucket. If the water is 20% salt, how many ml of salt will Jack get when all the water evaporates?
Output: 400

Input: John buys 1/2 gallon jugs of cold brew coffee every 4 days. How many cups of coffee does he drink a day?
Output: 2

Input: Matt needs to buy new plates for his home. He only wants to do dishes once a week. Three days a week it is only him and his son eating and they use 1 plate each. On the remainder of the days, his parents join them and everyone uses 2 plates that day. How many plates does he need to buy?
Output: 38

Input: In a compound, the number of cats is 20% less than the number of dogs. There are also twice as many frogs as the number of dogs in the compound. Calculate the total number of animals present in the compound if there are 160 frogs.
Output: 304

Input: Elise is buying more dog food. She buys a 15kg bag then another 10kg bag, and she now has 40kg of dog food. How many kilograms of dog food did Elise already have?
Output: 15

H.7 GradientStep

Two LLM calls: (i) a textual gradient for the prompt, then (ii) direct application of that gradient to produce the new prompt (no reflection merge, unlike Crossover).

H.7.1 Sub-step 1: Textual gradient generation

Identical mechanism to Crossover's per-parent textual-gradient generation (Section H.4.1); meta-prompt and example not repeated here.

H.7.2 Sub-step 2: Apply the gradient

Meta-prompt template

You are an expert in the domain of optimization prompts. Your task is to design prompts that can effectively solve optimization problems. Your response outputs prompt text and nothing else.

You will be provided the problem description (your prompts must solve it effectively), parent prompt and its textual gradient.
The textual gradient contains the crucial information about the best approaches in prompt optimization guided for the specific provided task. Use it wisely to create the most effective offspring possible, based on the current prompt.

[Problem description]
{PROBLEM_DESCRIPTION}

[Prompt]
{PROMPT}

[Textual Gradient]
{TEXTUAL_GRADIENT}

[Improved prompt]

Please write a mutated prompt, according to the reflection. Give the main priority to the provided gradient as it accumulates essential information about correct prompt structure and other different prompt features.

Output the mutated prompt only.
Bracket the final prompt with <prompt> </prompt>.

Example — input prompt

Given a context answer on the question.

Example — output prompt (after applying the textual gradient)

Read the following context thoroughly and provide a concise answer to the question based solely on the information present in the text. Your response should be brief, ideally using the exact wording from the context, and reflect precise details such as names, dates, or specific phrases. Keep your answer to one sentence or a maximum of 10 words.

H.8 HyPE

Single LLM call: the meta-prompt restructures the prompt into role, task, format, recommendations, and constraints sections.

Example — input prompt

Solve a mathematical word problem creatively; what is the approach? answer briefly.

Example — output prompt (restructured)

<Role> You are an expert in mathematical problem solving with a focus on creative reasoning and calculation techniques. </Role>

<Task context> The task involves solving a mathematical word problem creatively, emphasizing the approach to extract relevant data and apply mathematical principles based on the provided contextual information. </Task context>

<Instructions> Provide a brief overview of a creative approach to solving mathematical word problems. Highlight key steps such as identifying the scenario, extracting relevant data, applying appropriate operations, and computing the result, while maintaining clarity and engaging reasoning. </Instructions>

<Output requirements> The response should be brief, focusing on the creative approach to problem-solving without going into excessive detail. </Output requirements>

H.9 LongTermMutation

Mechanically identical to ElitistMutation (Section H.5): the same long-term-reflection update (Section H.5.1) followed by the same mutation-application call (Section H.5.2), with only the targeted prompt differing — a randomly selected population member rather than the current best. See Section H.5 for the meta-prompts and examples.

H.10 ParaphrasingByPD

Single LLM call: a simple paraphrase conditioned on the task description, for local-neighborhood exploration.

Meta-prompt template

You are an expert in the domain of optimization prompts. Your task is to design prompts that can effectively solve optimization problems.

Here is the description of your problem: {PROBLEM_DESCRIPTION}

Paraphrase the given prompt in order to improve it and to solve that problem in the most effective way.

[Prompt]
{PROMPT}

Remember to pay attention to all details provided in the description (i.e. constraints, restrictions, input-output formats and etc.)

Output prompt only.

Bracket the final prompt with <prompt> </prompt>.

Example — input prompt

Embark upon a journey through the narrative of numbers! As you are presented with a mathematical word problem, let your imagination and analytical mind intertwine. Begin by carefully dissecting the context: identify the key entities, relationships, and quantities that emerge from the story. Visualize the scenario in your mind, allowing the words to paint a vivid picture.

Next, delve deeper into the mathematical principles that lie beneath the surface. Consider the operations that will lead you toward the answer: addition, subtraction, multiplication, division, or perhaps a combination thereof. Reflect on the relationships between the numbers—are they part of a ratio, a percentage, or a sequence?

Channel your creativity to formulate a strategy that feels fluid and intuitive. Take a moment to breathe, to center your thoughts, and let clarity wash over you. As you begin your calculations, let each step be deliberate and confident, transforming abstract concepts into tangible results.

Finally, when you arrive at your answer, pause to appreciate the journey you've taken. Acknowledge the power of understanding and the satisfaction of solving the puzzle. Trust in your abilities, for you are both the artist and the mathematician, crafting beauty from logic. Now, dive into the problem with enthusiasm and insight—your solution awaits!

Example — output prompt (paraphrased)

Begin your exploration of mathematical word problems with a keen focus on the context provided. Start by identifying the essential entities, relationships, and quantities presented in the scenario. Visualize the situation to enhance your understanding of the problem.

Next, analyze the mathematical concepts that apply to this context. Determine which operations—addition, subtraction, multiplication, or division—are relevant, and evaluate the relationships among the numbers, such as ratios, percentages, or sequences.

Create a clear and logical strategy to approach the problem. Take a moment to gather your thoughts and ensure you have a solid grasp of the requirements. As you perform your calculations, proceed with careful and intentional steps, transforming abstract information into concrete results.

Once you reach your final answer, take a moment to reflect on the process you undertook. Recognize the importance of comprehension and the fulfillment that comes from solving the challenge. Trust in your skills to navigate through the problem effectively. Now, tackle the word problem with focus and determination—your solution is within reach!

H.11 ZeroOrder

Single LLM call: generates a new prompt from the task description alone, with plain (non-creative)

framing (contrast with CreativeZeroOrder, Section H.3).

Meta-prompt template

You are an expert in the domain of optimization prompts. Your task is to design prompts that can effectively solve optimization problems. Here is the description of your problem: {PROBLEM_DESCRIPTION}

Write the best prompt/instruction in order to solve that problem in the most effective way. Remember to pay attention to all details provided in the description (i.e. constraints, restrictions, input-output formats and etc.)
Output prompt only.
Bracket the final prompt with <prompt> </prompt>.

Example — input (task description)

You will be given a natural-language narrative that describes a real-world scenario involving people (often two or more) and quantities such as counts of items, times, rates, and relationships between quantities (for example, twice as many, half as many, per hour, per minute). The narrative ends with a question asking for a final numeric quantity (e.g., total number of items, total actions completed in a period, or a derived amount). Your task is to compute the correct answer from the information and relationships stated in the story. The problems typically require basic arithmetic operations (addition, subtraction, multiplication, division) and may involve more than one step or subproblem that must be combined to obtain the final result. Some scenarios provide rates (things done per unit of time) and time spans, so you may need to convert units to a common basis before multiplying or summing. Other scenarios describe distributions or sequences over time (for instance, changing rates from one hour to the next, or a quantity in the first period followed by quantities in subsequent periods based on a defined rule). The final answer should be a non-negative integer representing the quantity requested by the question. Problems may present extraneous details or multiple clauses that require careful parsing to identify the relevant data and the relationships that drive the computation. You should extract the essential numbers, set up the necessary equations or stepwise calculations, perform the computations accurately, and return the single numeric result that answers the question posed in the narrative.

Example — output prompt (generated from scratch)

Analyze the following mathematical word problem carefully.
Extract all relevant numerical information and operations presented in the scenario. Based on your understanding of the narrative, perform the necessary arithmetic calculations to determine the numeric answer to the specific question asked in the text. Your output should be a single numeric value that represents the solution to the problem. Ensure that your computations are accurate and reflect

the context provided.
Here is the problem: [Insert word problem here]

Reproducibility Notes

Default hyperparameters for all three BRAVE presets are as listed in Table 8 unless a preset explicitly overrides a value (e.g., BRAVE-Light and BRAVE-Ultra-Light use smaller budgets and correspondingly adjusted population/step settings, reflected directly in their measured actual-token spends in Section 4.3). Dataset splits follow a stratified train-batch-size(pool-size)/validation-size/test-size convention (e.g., 50(900)/100/300) as used throughout our experiments.