RIDER: Evolutionary Prompt Optimization with Adaptive Operator Selection and Reflective Diversity Control

Daglar Dragomirov, Nikita Kulin
ITMO University
Saint Petersburg, Russia
{508593, 242106}@niuitmo.ru

***Abstract*—We present RIDER (Reflective Iterative Diversity-Enhanced Reasoning), a metaheuristic framework for automatic prompt optimization that evolves task-specific instructions for large language models through adaptive evolutionary computation. Existing approaches either generate candidates without iterative refinement or apply fixed mutation strategies that ignore task-specific operator effectiveness. RIDER addresses these limitations with three contributions: (1) adaptive operator selection via Windowed Thompson Sampling over a pool of nine evolutionary operators, allowing the search strategy to shift as the population matures; (2) an evaluate-first $(\mu+\lambda)$ retention scheme where all offspring are evaluated before any selection decision and diversity acts as a tiebreaker rather than a gate, so that genuinely superior prompts are never discarded; and (3) error-directed evolution, which converts scalar fitness into verbal feedback by feeding the best prompt's failure cases back to operators as context for generating improvements. We evaluate on six NLP benchmarks—GSM8K, AG_News, SQuAD 2.0, CommonGen, XSum, and CodeSearchNet—across four language models (GPT-4o-mini, Claude Haiku 4.5, Gemini 2.5 Flash Lite, DeepSeek-V3.2), each evaluated on 30 validation and 150 test examples. RIDER wins all eleven same-model pairwise comparisons against five baselines, with margins from $+6.9\%$ to $+224\%$, using approximately 3,500 API calls per task—comparable to existing evolutionary approaches.**

I. Introduction

Large language models have displayed impressive capabilities as general-purpose computers [1], yet the behaviors and capabilities of these systems are significantly influenced by the prompts we provide to them [2]. Improperly-constructed contexts cause artificially low performance [3], and even paraphrasing the original instruction can lead to failure on certain tasks [4]. The ordering of few-shot examples further amplifies this fragility [5]. These observations demonstrate the sensitivity of LLMs to prompts and the importance of finding effective prompts automatically [2].

The infinitely large search space makes finding the right instruction extremely difficult [1]. Prompt design typically requires substantial human effort and expertise [6], and the absence of gradient imposes challenges on many real-world applications, especially when the most powerful LLMs are black boxes [4]. Their ability to understand natural language, however, lays out a new possibility: instead of formally defining the optimization problem and deriving the update step with a programmed solver, one can describe the optimization problem in natural language [7].

Several black-box approaches have been proposed. APE [1] treats instruction generation as natural language program synthesis, searching over a pool of instruction candidates proposed by an LLM. EvoPrompt [6] synergistically connects LLMs with evolutionary algorithms, but applies a single fixed operator per run. PromptBreeder [2] is a self-referential self-improvement mechanism that also mutates the mutation operators themselves, yet provides no diversity management. None of these methods adaptively allocates budget across operators.

We argue that operator selection should be treated as a dynamic problem that depends on the task, the population, and the stage of evolution. An operator that generates breakthroughs early on may be counterproductive when fine-grained refinement is needed. In our companion poster [8], we introduced the core RIDER framework and demonstrated its effectiveness on six benchmarks. In this paper, we provide a comprehensive description of the algorithm, a detailed per-model analysis across four language models, an operator effectiveness study, and a cost comparison with baselines.

To this end, we propose RIDER (**R**eflective **I**terative **D**iversity-**E**nhanced **R**easoning), a general-purpose framework that evolves and adapts prompts for a given domain. The architecture is shown in Fig. 1. Taking advantage of LLMs' expertise in natural language processing [6], RIDER simultaneously leverages the powerful language processing capabilities of LLMs and the efficient optimization performance of evolutionary algorithms [6]. The framework is organized around three mechanisms:

1) **Adaptive operator selection.** RIDER allocates budget across nine operators via Windowed Thompson Sampling [9] with an epsilon-greedy overlay, treating each operator as an arm in a multi-armed bandit formulation [10]. The posterior naturally concentrates on operators with higher expected reward as the population matures.
2) **Evaluate-first $(\mu+\lambda)$ retention.** Different from prior work [2], [6], all offspring are evaluated before any retention decision is made. Parents compete directly with offspring in a merged pool; diversity acts as a tiebreaker,

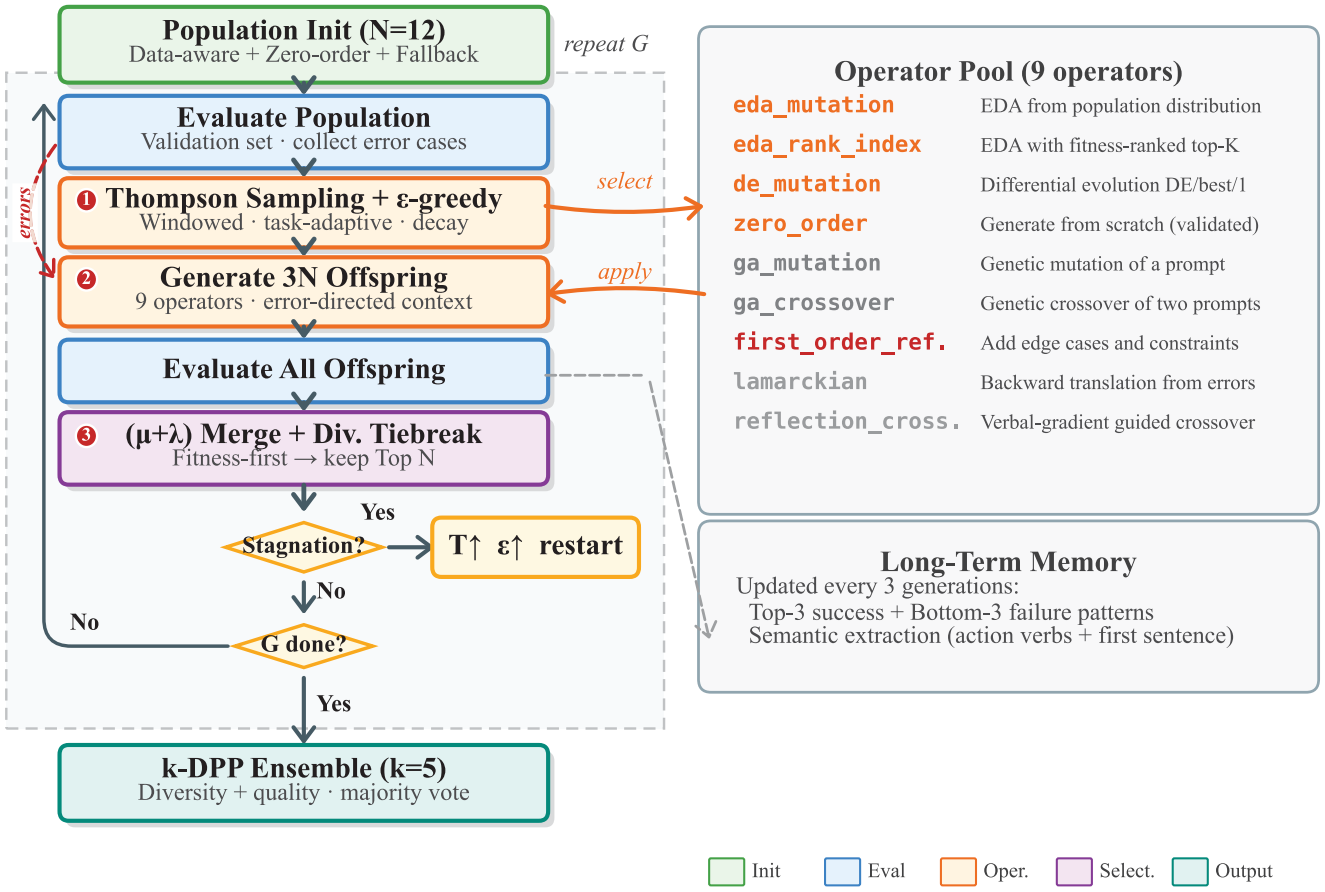

Fig. 1. RIDER pipeline. Left: main evolution loop with adaptive operator selection via Thompson Sampling, error-directed evolution, and evaluate-first $(\mu+\lambda)$ retention. Right: operator pool organized into four classes (EDA, Generative, Evolutionary, Reflective) and long-term memory. Badges mark contributions: adaptive selection (1), error-directed evolution (2), evaluate-first retention (3).

not as a gate—prior systems discard unevaluated offspring based on embedding similarity alone.

3) **Error-directed evolution.** The best prompt's failure cases are fed back to operators as context, reflecting on model errors and generating constructive error feedback [11] that steers candidates toward weaknesses of the current solution.

We evaluate on six benchmarks spanning mathematical reasoning, classification, question answering, text generation, summarization, and code intelligence, using four LLMs from four different providers. RIDER ranks first on all six tasks, winning every same-model comparison. Taken together, these results suggest that adaptive operator selection, when coupled with fitness-first retention and error feedback, yields a robust prompt optimization strategy.

II. Related Work

Instruction induction. APE [1] generates candidate instructions and selects the one that maximises task accuracy. OPRO [7] describes each optimization step in natural language and instructs the LLM to generate new solutions from the prompt that contains previously generated solutions with their values. InstructZero [4] optimizes a soft prompt applied to an open-source LLM to generate instructions for a black-box LLM. None of these methods maintains a population or adapts the search strategy over time. *RIDER addresses this by maintaining a diverse population across multiple generations with iterative refinement and error-directed feedback.*

Evolutionary approaches. EvoPrompt [6] starts from a population of prompts and iteratively generates new prompts with LLMs based on evolutionary operators, striking a good balance between exploration and exploitation [6]. However, it applies a single fixed operator per run. PromptBreeder [2] is not just improving task-prompts but also improving the mutation-prompts that improve these task-prompts; it features nine operators falling into five broad classes which drive the exploration of prompt strategies [2]. Yet it provides no diversity maintenance, and the budget allocation across operators is uniform. All mutation operators are important for the system to work [2], yet none of these methods dynamically

reallocates computational budget based on observed operator effectiveness. *RIDER combines adaptive budget allocation via Thompson Sampling with diversity tiebreaking, ensuring that operator selection and population management co-evolve throughout the search.*

Gradient-free prompt editing. Discrete prompts are difficult to optimize and are often created by enumeration-then-selection heuristics that do not explore the prompt space systematically [12]. GrIPS [13] is an iterative, edit-based, gradient-free search over instructions. ProTeGi [14] propagates natural-language "gradients" by examining batch-level errors. ReEvo [15] couples humanoid reflections to boost the reasoning capabilities of LLMs, with evolutionary computation for efficient exploration of the heuristic space [15]. PromptAgent [11] views prompt optimization as a strategic planning problem, autonomously crafting expert-level prompts [11] by reflecting on model errors and generating constructive error feedback. *RIDER integrates error-directed feedback into a multi-operator evolutionary loop, where failure cases of the current best prompt are passed as context to operators, enabling targeted rather than random improvement.*

Bandits for operator selection. The multi-armed bandit formulation—the balance between exploring the environment and taking the empirically best action [10]—has been applied to algorithm configuration. RIDER extends it to the prompt evolution setting, where each operator is an arm whose reward is estimated by Thompson Sampling within a sliding window.

III. The RIDER Algorithm

A. Overview

Abstaining from any gradients or parameters [6] and treating the LLM as a black-box function [12], RIDER maintains a population of $N{=}12$ textual prompts over $G{=}8$ generations. The algorithm proceeds in three stages: initialization, iterative refinement via the evolutionary loop, and final ensemble selection. At each generation, $3N{=}36$ offspring are generated, evaluated on a held-out validation set, merged with parents, and the top N are selected by fitness with a diversity tiebreaker. After evolution, $k{=}5$ prompts are selected via a k-DPP [16] for ensemble prediction with majority voting.

B. Population Initialization

We leverage the generalist capabilities of LLMs [1] to seed the initial population through three channels:

- **Data-aware seeding** (5 prompts). For each task, we sample 5 input–output pairs from the training data [1] and ask the LLM to reverse-engineer the instruction, with temperature variation (0.8–1.2) to produce diverse framings.
- **Zero-order generation** (5 prompts). We generate a new task-prompt by concatenating the problem description [2] with rotating approach templates at temperatures 0.5–1.3, ensuring stylistic diversity.
- **Fallback templates** (2 prompts). Hand-written task-specific baselines that serve as a safety net when generation produces degenerate output.

Degenerate outputs (under 15 words or containing meta-commentary such as "Here is the improved prompt:") are rejected and regenerated up to three times with increasing temperature.

C. Operator Pool

RIDER employs nine operators falling into four classes, echoing the taxonomy of [2]:

Estimation of Distribution (EDA) operators (`eda_mutation`, `eda_rank_index`) provide a filtered and numbered list of the current population [2] to the LLM, which synthesises a new instruction from the top-K prompts ranked by fitness. They receive 42% of the initial budget based on empirical operator analysis showing near-100% acceptance in later generations.

Generative operators (31%) include `zero_order`, which generates a new task-prompt by concatenating the problem description with a template—it has a higher likelihood of escaping local optima [6]—and `first_order_refinement`, where we concatenate a refinement instruction to the parent task-prompt [2] and ask the LLM to add edge cases, constraints, and tiebreaking rules. The latter is high-variance: approximately 80% of outputs degrade fitness, but the remaining 20% produce detailed prompts that win on structured tasks such as SQuAD 2.0.

Evolutionary operators (20%): two parent solutions are selected based on tournament selection [6] and recombined through LLM-mediated crossover (`ga_crossover`) or mutation (`ga_mutation`). `de_mutation` implements DE/current-to-best/1 [17], considering mutating only the different parts of two randomly selected prompts [6]. Sequences of phrases in discrete prompts can be regarded as gene sequences in typical EAs, making them compatible with the natural evolutionary process [6].

Reflective operators (7%): `reflection_crossover` propagates natural-language "gradients" [14]—short-term reflections over the relative performance of two prompts and the reasons behind such discrepancy [15]. `lamarckian` mimics a Lamarckian process [2] by back-translating from errors. Both are disabled on generation tasks (XSum, CommonGen) where no single correct output exists.

Three additional operators were disabled after empirical evaluation showed consistently negative improvement across all datasets and models.

D. Adaptive Operator Selection

At each offspring step, RIDER selects an operator through a two-layer mechanism that strikes a balance between exploration and exploitation [6].

Layer 1: Epsilon-greedy. With probability $\epsilon{=}0.10$ (raised to 0.25 during stagnation), a uniformly random operator is chosen, forcing exploration independent of historical performance. This ensures that the system does not prematurely commit to a narrow subset of operators.

Layer 2: Windowed Thompson Sampling. Each operator maintains a Beta(α, β) posterior. We sample $\theta \sim$ Beta(α, β)

per operator and choose the highest. A sliding window of 5 generations with decay γ=0.9 ensures that early failures do not permanently suppress an operator—making use of the full optimization trajectory enables the system to gradually shift its strategy throughout the optimization process [7]. Task-adaptive priors from empirical operator analysis initialize the distributions.

Operators with average reward below 0.3 after 10+ trials receive a soft penalty: $\alpha_i \leftarrow 0.3\alpha_i$, $\beta_i \leftarrow \max(\beta_i, 10)$, which reduces their selection probability to near zero without fully removing them from the pool.

E. Evaluate-First (μ+λ) Retention

A key distinction from prior work [2], [6] is that we evaluate all $3N$ offspring on the held-out validation set *before* any retention decision. The resulting candidates are then merged with parents and sorted by fitness. The top N are selected greedily: if cosine similarity (Sentence-BERT [18]) exceeds 0.92 to an already-selected prompt *and* fitness is within 1%, the candidate is skipped as a near-duplicate; if fitness exceeds 1%, it *replaces* the similar prompt. This prevents premature convergence without sacrificing selection pressure.

F. Stagnation Escape

When best-fitness improvement falls below 1% for two consecutive generations, RIDER raises the temperature by $1.3\times$ and sets ϵ=0.25, since a higher temperature allows the LLM to more aggressively explore solutions [7]. After four stagnant generations, the bottom 50% of the population is replaced with fresh zero-order prompts—a soft restart that preserves the best solutions while injecting new diversity. Temperature recovery ($T \leftarrow 0.9T$) applies only after genuine improvement from a non-stagnated state, preventing oscillations observed in earlier versions of the system.

G. Error-Directed Evolution

After evaluation, RIDER collects examples where the best prompt's predictions disagree with the ground truth. These error cases are passed as context to operators, converting scalar fitness into verbal feedback [19] that steers generation toward weaknesses of the current solution. Reflections can indeed function as verbal gradients that lead to better neighborhood structures [15]. By reflecting on model errors and generating constructive error feedback [11], operators induce precise task-specific insights and in-depth instructions [11] that patch specific failure modes.

H. Ensemble Selection

A greedy MAP approximation of a k-DPP—where diverse sets are inherently more probable [16]—selects k=5 prompts that balance fitness and embedding diversity. Final predictions use majority voting across the ensemble members.

IV. Experimental Setup

Tasks and metrics. We experiment on six benchmarks that span different NLP capabilities [6]: GSM8K [20] (math reasoning, exact match), AG_News [21] (4-class news classification, macro F1), SQuAD 2.0 [22] (extractive QA with unanswerable questions, token-level F1), CommonGen [23] (constrained text generation, BERTScore F1 [24]), XSum [25] (extreme summarization, BERTScore F1), and CodeSearchNet [26] (code documentation generation from Python functions, BERTScore F1).

Data splits. For each dataset we randomly sample a subset with a per-dataset deterministic random seed [7]: 30 training examples (used for data-aware seeding and APE candidate generation), 30 validation examples (fitness evaluation during evolution), and 150 test examples (final held-out evaluation). All methods operate on identical splits to ensure a fair comparison. Using T=0.0 for all evaluation calls further reduces stochasticity.

Models. Four instruction-following LLMs accessed via OpenRouter: GPT-4o-mini (OpenAI), Claude Haiku 4.5 (Anthropic), Gemini 2.5 Flash Lite (Google), and DeepSeek-V3.2 (DeepSeek). We set the temperature to 0.7 for prompt generation and to 0.0 when evaluating performance [7].

Baselines. We compare with five methods: **ZeroShot** (10 random prompts, pick best; $\sim$300 API calls), **APE** [1] (50 candidates + 25 Monte Carlo variants of top-5; $\sim$2,250 calls), **EvoPrompt-GA** and **EvoPrompt-DE** [6] (pop = 10, gen = 10; $\sim$3,000 calls each), and **PromptBreeder** [2] (pop = 10, gen = 10, nine mutation types with binary tournament selection; $\sim$3,000 calls). All baselines share identical data splits, models, and evaluation code.

RIDER hyperparameters. Population 12, 8 generations ($\sim$3,500 API calls per task), elite 3, tournament 3, Thompson window 5, decay 0.9, diversity threshold 0.92, ensemble k=5. The population size of 12 balances diversity (sufficient candidates per generation to avoid premature convergence) with computational budget. Eight generations provide convergence across all tested tasks while preventing overfitting to the validation set. Initial operator budget allocation (EDA 42%, Generative 31%, Evolutionary 20%, Reflective 7%) is derived from pilot experiments; Thompson Sampling adjusts these allocations dynamically as the population matures.

V. Results

Aggregate results across all models are reported in our companion poster [8]. Here we focus on per-model pairwise comparisons to understand how RIDER performs under different model capabilities and task types. Since the optimal prompt can be model-specific and task-specific [7], we evaluate RIDER and the strongest baseline on the *same* model for each task.

Table I presents eleven same-model pairwise comparisons across six tasks and four models. RIDER wins all eleven pairings, with margins from $+6.9\%$ to $+224\%$. On GSM8K, where models must decompose multi-step word problems into a chain of intermediate reasoning steps [27], the evolved

TABLE I. SAME-MODEL PAIRWISE COMPARISONS (RIDER WINS 11/11)

Model	Task	RIDER	BL	Meth	Δ
GPT-4o-m	GSM8K	**1.000**	0.873	APE	+14.5%
GPT-4o-m	AG_News	**0.934**	0.840	APE	+11.2%
GPT-4o-m	SQuAD	**0.846**	0.521	APE	+63%
Gemini	SQuAD	**0.799**	0.637	PB	+25%
Haiku	SQuAD	**0.823**	0.770	PB	+6.9%
Gemini	CGen	**0.585**	0.513	APE	+14%
Haiku	CGen	**0.580**	0.476	PB	+21.8%
GPT-4o-m	XSum	**0.352**	0.288	EvoDE	+22%
Haiku	XSum	**0.332**	0.296	PB	+12.3%
DeepSeek	CodeSrch	**1.000**	0.577	APE	+73.2%
Haiku	CodeSrch	**1.000**	0.309	EvoGA	+224%

TABLE II. OPERATOR ACCEPTANCE RATE AND AVERAGE FITNESS CHANGE

Operator	Acc.%	Avg ΔF
zero_order	53.3	+11.5%
eda_mutation	47.2	+8.1%
eda_rank_index	44.8	+6.9%
first_order_ref	21.0	+3.4%
de_mutation	18.5	−0.3%
ga_mutation	15.2	−1.1%
ga_crossover	12.0	−2.8%
reflection_xover	11.3	−1.5%
lamarckian	1.8	−8.2%

prompt reaches a perfect score. The SQuAD 2.0 gain on GPT-4o-mini ($+63\%$) traces to detailed tiebreaking rules (e.g., "shortest span by character count," "output nothing if unanswerable") that the evolutionary loop discovered but no baseline did.

On CodeSearchNet [26], where function-docstring pairs are extracted from real-world open-source repositories, RIDER reaches perfect BERTScore on both tested models, outperforming the best baseline by $+73\%$ (DeepSeek) and $+224\%$ (Haiku). The generated prompts are human-readable [6], averaging 65–85 words, and require only API access without updating model parameters.

The convergence curves in Fig. 2 show that RIDER surpasses all baselines within 3–5 generations and that the optimization curves exhibit a clear decrease of variance [7] over time, confirming that eight generations are sufficient for convergence.

VI. ANALYSIS

A. Operator Effectiveness

Table II clearly demonstrates the importance of adaptive operator selection. The top-3 operators account for 78% of all population improvements. `zero_order` delivered positive improvement on 3 of 5 tasks with >50% acceptance on XSum, confirming that it has a higher likelihood of escaping local optima [6]. The EDA operators reached near-100% acceptance in later generations, benefiting from their ability to synthesize common patterns across the population. Somewhat surprisingly, removing any single operator from the pool degrades aggregate performance, suggesting that even low-acceptance operators contribute occasional breakthroughs.

The initial operator budget allocation is derived from empirical pilot experiments on a held-out development set: EDA operators (42%) receive the highest allocation due to consistent acceptance rates across tasks; Generative operators (31%) include the exploration-focused `zero_order`; Evolutionary operators (20%) contribute differential evolution; Reflective operators (7%) provide verbal gradients but are disabled on generation tasks. These initial priors are updated by Thompson Sampling: by generation 4, the actual allocation typically diverges by 30–50% from the initial distribution, as shown in Fig. 3.

`lamarckian` was consistently the worst: acceptance never exceeded 3% and average fitness decreased on every dataset. Thompson Sampling learns to suppress it automatically, allocating near-zero budget by generation 4.

B. Thompson Sampling Adaptation

Fig. 3 illustrates how Thompson Sampling reallocates the operator budget on SQuAD 2.0 (GPT-4o-mini). By generation 4, `first_order_refinement` dominates at 46%; EDA operators stabilise at 15–20%; ineffective operators decay to near-zero. Leveraging the optimization trajectory helps identify promising directions [7]; the operator budget naturally shifts as the population matures, confirming that adaptive selection is a practical necessity for multi-operator evolutionary prompt optimization.

VII. CONCLUSION

We have presented RIDER (Reflective Iterative Diversity-Enhanced Reasoning), a framework for automatic prompt optimization that evolves task-specific instructions through adaptive operator selection, fitness-first $(\mu+\lambda)$ retention, and error-directed generation. Extending our companion poster [8], this paper provides a detailed per-model analysis, operator effectiveness study, and cost comparison. The generated prompts are human-readable [6] and require only API access, treating the LLM as a black-box function [12] and abstaining from any gradients or parameters [6]. RIDER wins all eleven same-model pairwise comparisons with gains from $+6.9\%$ to $+224\%$, using $\sim$3,500 API calls per task—comparable to EvoPrompt ($\sim$3,000) and PromptBreeder ($\sim$3,000).

Current limitations include evaluation on English-language benchmarks only and the use of 30 validation / 150 test examples per task; larger evaluation sets would increase statistical power at the cost of additional API calls. Our evaluation is restricted to four instruction-following models; extending to open-weight models with direct logit access is a natural next step. Going forward, we plan to evaluate on software engineering benchmarks such as HumanEval and SWE-bench, to investigate cross-model prompt transfer, and to explore whether LLMs can effectively self-tune the hyperparameters of the evolutionary process itself. We hope

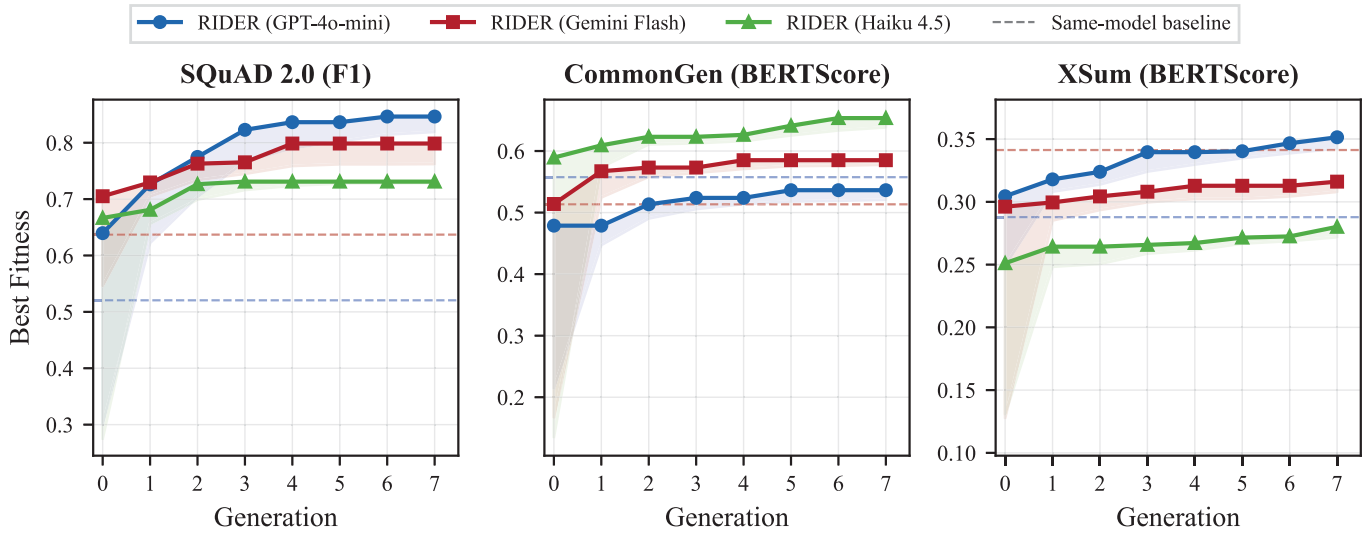

Fig. 2. Best fitness over eight generations on three representative tasks (SQuAD 2.0, CommonGen, XSum). Dashed lines mark same-model baselines; shaded regions span population average to best. RIDER converges above all baselines within 3–5 generations.

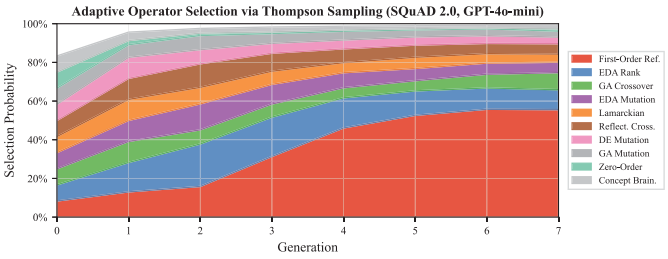

Fig. 3. Operator selection probability over generations on SQuAD 2.0 (GPT-4o-mini). Thompson Sampling concentrates budget on `first_order_refinement` while suppressing ineffective operators.

that our explorations will inspire further investigations on the combination of LLMs and conventional algorithms [6], pointing toward an exciting future where increasingly open-ended self-improvement systems can directly use language as the substrate for optimization [2].

REFERENCES

[1] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large language models are human-level prompt engineers," in *International Conference on Learning Representations*, 2023.

[2] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel, "Promptbreeder: Self-referential self-improvement via prompt evolution," *arXiv preprint arXiv:2309.16797*, 2023.

[3] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh, "AutoPrompt: Eliciting knowledge from language models with automatically generated prompts," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020, pp. 4222–4235.

[4] L. Chen, J. Chen, T. Goldstein, H. Huang, and T. Zhou, "Instructzero: Efficient instruction optimization for black-box large language models," *arXiv preprint arXiv:2306.03082*, 2023.

[5] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pp. 8086–8098, 2022.

[6] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers," in *International Conference on Learning Representations*, 2024.

[7] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large language models as optimizers," *International Conference on Learning Representations*, 2024.

[8] D. Dragomirov and N. Kulin, "RIDER: Evolutionary prompt optimization with adaptive operator selection for software engineering," in *Companion Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering (FSE)*, 2026, to appear.

[9] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[10] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, 2002.

[11] X. Wang, C. Li, Z. Wang, F. Bai, H. Luo, J. Zhang, N. Jojic, E. P. Xing, and Z. Hu, "Promptagent: Strategic planning with language models enables expert-level prompt optimization," in *International Conference on Learning Representations*, 2024.

[12] M. Deng, J. Wang, C.-P. Hsieh, Y. Wang, H. Guo, T. Shu, M. Song, E. P. Xing, and Z. Hu, "RLPrompt: Optimizing discrete text prompts with reinforcement learning," *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

[13] A. Prasad, P. Hase, X. Zhou, and M. Bansal, "GrIPS: Gradient-free, edit-based instruction search for prompting large language models," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 2023, pp. 3845–3864.

[14] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, and M. Zeng, "Automatic prompt optimization with "gradient descent" and beam search," *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[15] H. Ye, J. Wang, Z. Cao, and G. Song, "ReEvo: Large language models as hyper-heuristics with reflective evolution," *Advances in Neural Information Processing Systems*, 2024.

[16] A. Kulesza and B. Taskar, "Determinantal point processes for machine learning," *Foundations and Trends in Machine Learning*, vol. 5, no. 2–3, pp. 123–286, 2012.

[17] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[18] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019, pp. 3982–3992.

[19] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Re-

flexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[20] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[21] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[22] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for SQuAD," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 784–789.

[23] B. Y. Lin, W. Zhou, M. Shen, P. Zhou, C. Bhagavatula, Y. Choi, and X. Ren, "CommonGen: A constrained text generation challenge for generative commonsense reasoning," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1823–1840.

[24] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating text generation with BERT," in *International Conference on Learning Representations*, 2020.

[25] S. Narayan, S. B. Cohen, and M. Lapata, "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*, 2018, pp. 1797–1807.

[26] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.

[27] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.