Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

APRA: Adaptive Prompt Refinement with Early Stopping for Efficient Automatic Prompt Optimization in Generative AI

Varshan Karthik R, Pranish M, Rohinth V, Mohamed Sharrjun M, Mr. N.Ravi
PSG Polytechnic College, Coimbatore, Tamil Nadu

Abstract*—The automatic prompt optimization (APO) research area moved beyond prompt editing into independent research direction in which the language models generate the criticism, candidates, or reflections to optimize the prompt to maximize the task performance. Current approaches include such techniques as textual gradients and beam search, evolutionary and reflective optimizers. Though these algorithms optimize prompt without changing parameters of the model, their search strategy is expensive in terms of computation and is not task-dependent. This work suggests a light-weight Adaptive Prompt Refinement Algorithm (APRA), which implements three components – best prompt tracking, improvement threshold, and patience-based stopping criteria into the single-candidate refinement procedure. The research question is whether the task-dependent adaptive allocation of inference budget is possible based on feedback from the development set. We state the problem, provide the formal cost modeling, compare APRA with other APO methods and design reproducible small-scale experiment based on the classification task. The particular attention will be paid to the validity threats since one-hour pilot is not enough to confirm superiority of our approach over other APO approaches. No experimental results were fabricated in this manuscript: the tables are designed to be filled in based on the actual run of the experiments. Thus, the novelty of this manuscript lies in the proposed optimization policy rather than superiority over other APO systems.

***Index Terms*—Generative AI, large language models, automatic prompt optimization, prompt refinement, prompt engineering, early stopping, adaptive optimization, inference efficiency.**

I. INTRODUCTION

In most cases, language models (LLMs) are adapted to the task with natural-language prompts instead of parameters change. Deployment of the language models usually involves the prompts tuning through the examination of failures; however, the process is expensive in terms of time and requires a lot of expertise. Automatic prompt optimization (APO) aims at solving the above-mentioned issue by considering the prompt as a target for evaluation, criticism, rewriting and optimization. Recent studies have progressed from prompt editing to the structured optimization algorithms. For instance, ProTeGi considers the natural-language criticism as textual gradients and applies beam search and bandit algorithm to select the best candidates. Optimization by PROmpting (OPRO) approach relies on using language model to generate new prompt candidates based on previous prompts and their scores. Promptbreeder uses evolutionary algorithm to co-optimize task prompts and mutation prompts .Existing approaches to the APO expand the search space even more. RiOT algorithm uses residual optimization tree and minimize the semantic drift of the prompt due to the principle of semantic diversity and residual connection.

The Promptomatix project is focused on prompt optimization and cost reduction in optimal way, including light-weight meta-prompt optimizer . ZERA applies structured techniques to optimize in relatively small cycles systems and user's instructions together. GEPA utilizes language reflection and evolution prompts update techniques and demonstrates that reflective optimizer is able to achieve high results using significantly fewer rollouts comparing with some baselines of reinforcement learning. The space of APO is defined in 2025 survey by the optimized object, scoring signal, generation operator, and search 33method.

This is the emergence of the design problem. With the objective being the generation of an optimal prompt within the scope of a finite budget, how can the necessity arise for an optimizer to do so through the same number of iterations? The concept of finite budget is simple but inefficient in the case where the prompt is already saturated. In this way, the early termination of an optimizer may happen prematurely before any improvements start becoming apparent. The optimal trade-off will vary depending on the use case, the model, the development set, and the refinement operator. The performance of an optimizer has also been shown to depend on the capacity of the model and computation in a recent work by .

The problem becomes more pressing when prompt optimization is considered a black box control loop, where every additional iteration might require more model calls as well as evaluations of candidate prompts. While both RiOT and evolutionary methods utilize large search spaces because exploration is necessary, APRA seeks to answer the question of how long to continue iterations after the search process identifies a meaningful direction to go. Thus, the proposed solution is not aimed at replacing beam search, tree search, and evolutionary generation of prompts. Instead, it introduces a controller that wraps a single candidate refinement operator.

The primary contribution is the Adaptive Prompt Refinement Algorithm (APRA). In each iteration of APRA the current prompt is evaluated on a small development set, failures are extracted,one critique and one revision of the prompt are solicited,the best prompt encountered so far is stored, and the algorithm stops running once the measured improvement falls below a predefined threshold over a given number of consecutive iterations. This algorithm is purposely designed to be simple enough to replicate in the classroom, laboratory, or project setting with limited resources.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

How does the paper contribute?

1) The paper provides a definition of APRA as a controller for single candidate prompt tuning through separating the stop condition from the prompting modification process.

2) The paper offers an optimization function as well as a cost function based on the budgeting of calls/tokens.

3) The paper provides a reproducible pilot protocol with an unoptimized baseline and fixed rounds comparator to separate the adaptive aspect of the approach.

4) The paper identifies the limits of applicability of a small one-hour study

The work ProTeGi presents the way how LLMs feedback can be used for prompt tuning. In its turn, ProTeGi gives an opportunity to know what are the natural language errors and to tune the prompt semantics appropriately. Several prompts are analyzed using the beam search and bandit methods . The main advantage of the ProTeGi method is that it can optimize the prompts when the gradient information about the model is not known. Prompt optimization and candidate optimization are the two phases of the process.

A. LLMs as Optimizers

OPRO is a language optimizer that uses LLMs to create and evaluate candidates . It should be noted that the optimizer model is of great importance for the algorithm since, based on many scientific works, the performance of the algorithm OPRO is highly dependent on the capacity of the optimizer model. In the case of APRA, there is an upper limit of optimization steps with adaptive stopping criterion.

B. Evolutionary and Reflective Optimization

Regarding the promptbreeder, prompt evolution is the recursive self-evolutionary process, which involves optimization of task prompts via mutations, which are also optimized . Reflection is applied in GEPA for extracting rules and optimizing prompts. It is important to note that the use of natural language feedback is a highly efficient optimization signal. This is when we have the idea for APRA, since they ask how optimization works in case of learning failures.

C. Structured and Cost-Aware Systems

The cost-aware system with prompt generation and optimization of meta-prompt is developed in Promptomatix . The cost-aware structured optimization with structured iterations is implemented in ZERA. Prompt diversity and semantic drift optimization are done via residual optimization tree and various iterations in RiOT. The point is that optimizer selection and its cost are crucial aspects. APRA solves this issue via adaptive iterations.

D. Recent Evidence about the Efficiency and Failures of Optimization

There are four main properties determined in the recent researches of APOs which are efficiency, robustness, search operators and evaluation. As a rule, reflective optimization does not provide a steady improvement as well. From the findings provided by Liu et al., it was revealed that there was one situation when the reflective optimizer was worse than standard optimizer due to a wrong prompt. It should be emphasized that there is another weakness of APRA in that the early stopping does not mean successful optimization. Although the controller is able to reduce the amount of computations, wrong development set feedback could make it stop before finding a suitable prompt.

E. APRA Positioning

APRA can be regarded as a budget controller but not a new refinement operator. The base refinement operator is able to work with textual criticism, meta-prompt editing or even specific instructions on prompt rewriting. APRA relies on development set improvement only. This means that Base performs initial prompt testing, Fixed-R checks a certain number of refinement iterations while APRA determines if refinement is required.

I. Research Gap and Motivation

It should be noted that the research gap should be formulated with care as the notions of efficiency differ a lot. the papers on cost aware or efficient optimization algorithms are mentioned and on overhead-free approach. it means that the field has paid attention to efficiency but in this paper we are talking about something else. the research gap lies in the absence of any testable budget controller that can be included into a single-candidate loop refinement and tested together with this loop within the fixed-budget setting.

TABLE I
COMPARISON OF THE CONCEPTS OF RELEVANT METHODS FOR AUTOMATIC PROMPT OPTIMIZATION

Approach	Criterion	Candidates	Relevance to APRA
ProTeGi	Text Gradient Analysis	Beam & bandit Search	APRA can employ one critic for one candidate
OPRO	Calculation of Candidates' Scores	Iterative Candidates Pool	APRA employs rounds budget and does not employ search space description
Promptbreeder	Evolutionary Modification of Candidates	Population	APRA does not employ any population
RiOT	Text Gradient & Residual	Optimization Tree	APRA employs single candidate
Promptomatix	Meta-prompt Compiler	Modular Optimization of Candidates	APRA is as cost-effective as Promptomatix
ZERA	Principles of Generation	System & User Prompt	APRA is as light as ZERA
GEPA	Linguistic Reflection	Pareto Evolutionary Search	APRA introduces stop control
APRA	Criterion of Improvement	Single Candidate	APRA focuses on stopping

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

This framing matters. If two optimizers use different search operators, then a direct accuracy comparison mixes several variables. APRA instead holds the prompt-generation mechanism constant and varies only the continuation policy. That makes the causal interpretation cleaner for a small study. The controller is based on the marginal development-set improvement

$$\Delta_r = A(p_r; D_{dev}) - A(p_{r-1}; D_{dev}),$$

where A is the development accuracy after deterministic output normalization.

The intuition is simple. If improvements remain large, another refinement round has evidence in its favor. If improve-ments repeatedly fall below a minimum meaningful threshold δ, the expected benefit of another round is small relative to its cost. This is a heuristic, not a statistical guarantee. The controller therefore has a hard maximum number of rounds and returns the best prompt observed, which limits the consequences of a poor later rewrite.

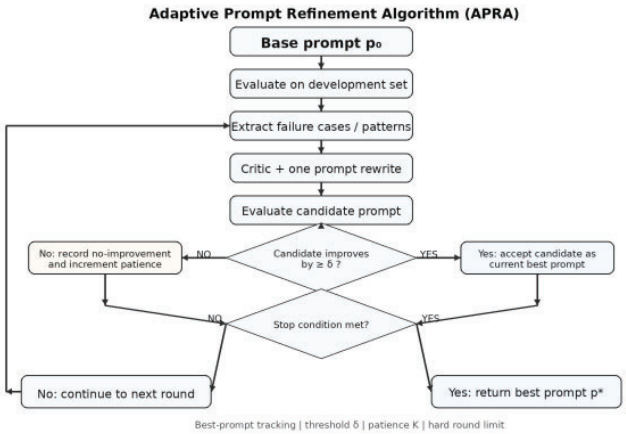


Figure 1. Now, there are two decision-making points in the process of loop refinement: one concerned with possible improvement and another one concerned with loop termination. This arrow coming back to the starting prompt from the controller had a bit vague starting point in the previous figure.

I. Problem Formulation

Let $D_{dev} = \{(x_i, y_i)\}_{i=1}^{N_d}$ be a labeled development set and D_{test} a disjoint held-out test set. Let p denote a natural-language prompt and $M(p, x)$ the deterministic model output for example x. After task-specific normalization, define

$$A(p; D) = \frac{1}{|D|} \sum_{(x,y)\in D} \mathbf{1}[M(p, x) = y].$$

Let R be a refinement operator such that

$$p_r = R(p_{r-1}, F_{r-1}),$$

where F_{r-1} is a failure summary derived from development examples. The standard optimization objective is to maximize test-set quality, but test labels must not be used during optimization. A budget-aware formulation is

$$\max_{p_r} A(p_r; D_{test}) \quad \text{subject to} \quad C(p_r) \le B,$$

II. Proposed APRA Approach

A. Need for Design

APRA is designed based on the following five conditions:

1) The controller must not require model gradients or fine-tuning.
2) It must work with one active prompt rather than a population or tree.
3) It must use only development-set feedback during optimization.
4) It must have a hard upper bound on the number of refinement rounds.
5) It must never discard the best observed prompt solely because a later candidate is generated.

Algorithm 1 Adaptive Prompt Refinement Algorithm (APRA)

Require: $p_0, D_{dev}, \delta, K, R_{max}$
$p^* \leftarrow p_0$; $a^* \leftarrow A(p_0; D_{dev})$; $q \leftarrow 0$
for $r = 1, \ldots, R_{max}$ **do**
  Build failure summary F_{r-1} from (p^*, D_{dev})
  $p' \leftarrow R(p^*, F_{r-1})$
  $a' \leftarrow A(p'; D_{dev})$
  $\Delta \leftarrow a' - a^*$
  if $a' \geq a^*$ **then**
    $p^* \leftarrow p'$; $a^* \leftarrow a'$
  end if
  if $\Delta < \delta$ **then**
    $q \leftarrow q + 1$
  else
    $q \leftarrow 0$
  end if
  if $a^* = 1$ **or** $q \geq K$ **then**
    break
  end if
end for
return p^*

Figure 1. Algorithm of APRA. After each rewriting cycle, the candidate prompt is being rated, the optimal one is selected, and a patience controller determines whether another cycle should be conducted or not.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

A. Threshold Sensitivity

As the threshold value of δ becomes smaller, the process of optimization will be more patient towards small differences, so it will take longer time. However, if the values of δ are bigger, then the process of optimization will be careful and will stop at the first slight change. K is the stability coefficient and it serves as another coefficient of stabilization which guarantees that noise of just one iteration will not interrupt the whole process of optimization. In general, δ, K and the size of the development set will be one of the dependent variables in this experiment.

III. EXPERIMENTAL METHODOLOGY

Research Questions

There are four research questions to be solved in this experiment.

- RQ1: Does APRA preserve the performance of the held-out data better than the unoptimized prompt?
- RQ2: Is APRA able to make less number of iterations compared to fixed-round approach?
- RQ3: Can the quality/effort trade-off provided by APRA be higher compared to fixed-round refinement in the case of the pilot task?
- RQ4: How sensitive are the results of the experiments to the values of parameters?

All of those hypotheses are rather conservative. According to H1, APRA can work at least as well as the benchmark algorithm, and even perform better than it. H2 provides a smaller number of iterations for refinement in some cases. H3 provides a smaller number of optimization iterations but still provides the same quality compared to fixed-round refinement. H4 states that the results of experiments using medium threshold values are more sensitive compared to the aggressive ones.

A. Task and Dataset

First experiment is related to the classification task, which provides an opportunity to evaluate the quality of the proposed algorithm with a relatively small effort. The pilot task, used for the current research is SST-2 classification when a binary classifier is built based on a small stratified sample of 20 training and 50 test samples**.**

The small-sample setup is merely a pilot experiment and should not be used as a benchmark. A more extensive replication study should involve repeating the entire process with multiple random seeds and at least one different task with dissimilar output properties. Reasoning or code generation task can be added later on, but the scoring function needs to be developed independently.

A. Baseline and Comparators

Three experimental conditions allow the isolation of the adaptive component:

1) Control condition: employ the initial prompt p0 without any optimization.

3) APRA: use the same procedure but enable early stopping as described by Algorithm 1.The key difference between Fixed-3 and APRA is the condition for continuation of the search; if you want to evaluate the effect of early stopping on the optimization process, it would be more reasonable to compare APRA with Fixed-3 instead of some other optimization method.

D. Prompt Templates

The minimal instruction for this task will be: "Classify the sentiment of the sentence as POSITIVE or NEGATIVE. Output exactly one label." The instructions for the critic will be the same as for the original task but with addition of the list of the failure modes and the prompt template requirements. Critic will be explicitly prohibited from using the test data, changing the output format and creating multiple prompt templates at once.This decreases the impact of differences in format on the result.

E. Evaluation Metrics

The key metric of success will be the exact match test accuracy

$$Acc = \frac{}{|D_{test}|}.$$

E. Execution Parameters

The same base model, configuration, decoding parameters, maximum number of outputs, and parsing parameters have to be used in all approaches. If deterministic decoding is used, temperature should be set to zero. Model name, release date, if available, provider, API version, and execution date have to be included in the report. Minimum of three seeds has to be used if stochastic decoding is applied.

F. Ablation Study

In order to perform a minimal pilot experiment, the following values for the threshold \(\delta\) could be used: \(\delta \in \{0.00, 0.05, 0.10\}\), \(K=2\), \(R_{max}=3\). Other possible values are \(K \in \{1,2\}\). The goal of the ablation study is not to explore the full parameter space but to investigate the impact of certain parameter values. ANALYSIS PLAN AND REPORTING. The analysis should present both the final test result and the path taken to obtain it. A single final accuracy number hides whether the controller stopped after one round or consumed its entire budget. Therefore, each experiment should log the development score after every round, the candidate prompt length, the number of critic calls, and the stopping reason.

For statistical analysis, the study should report absolute differences as well as an uncertainty interval. With a 50-example test set, a bootstrap interval can be used to show

TABLE II
EXPERIMENTAL RESULT TABLE TO BE POPULATED FROM AN ACTUAL RUN.

Method	Test Acc.	Critic	Eval.	Tokens	Time (s)	Stop
Base	–	0	1	–	–	N/A
Fixed-3	–	3	4	–	–	3
APRA	–	–	–	–	–	–

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

The accuracy of the estimation depends on the examples that are chosen for testing. In case the same examples are chosen for two methods, the paired analysis could be performed. The aim of this tiny investigation is descriptive and diagnostic, thus, no statistical conclusions about superiority could be made.

An interesting graph is the trajectory of optimization. The development accuracy is shown on the vertical axis, the number of refinement round – on the horizontal one. APRA indicates the round of termination, while Fixed-3 keeps running until the budget limit is reached. An additional graph may show the number of calls/token usage vs. test accuracy. Thus, it will become clear how budget affects performance without large tables.

A. Example Calculation

In case of a fixed budget equal to 3 rounds of refinement, when APRA terminates after the first round, it utilizes \(1/3\) of the critic-call budget and offers 66.7% saving. The saving becomes 33.3% when APRA terminates after the second round. These numbers are deterministic and are dependent on the stopping round. They are not experimental results. The real stopping round should be indicated in the experiment..

III. Reproducibility Protocol

The compressed research artifact must contain six components: identifiers of the data split, baseline prompt, critic prompt, APRA execution, run log in JSON or CSV format, and analysis script. The log should contain timestamp, model, config, prompt version, development score, stop criteria, tokens used, and latency time. Raw model outputs can be kept for auditing purposes and identifying test set contamination.Data split must be fixed before optimization. In case of SST-2 and any other publicly available dataset, it is better to keep only identifiers of the samples to prevent redistribution problems. Manuscript should contain information about the sampling strategy, random seed, and filters used in order to make an evaluation reproducible by other researchers.Code must differentiate data on which the optimization is happening and data kept for evaluation. For example, the function for generating failure summary should only access the development iterator. The test iterator must not be available until optimization is finished. This makes possible programmatic protection against unintentional data leaks..

IV. Threats to Validity

A. Internal Validity

The key internal threat is variance in the behavior of the model. Despite the use of deterministic parameters in API requests, variance might appear due to the updates at the backend level or at the service level. Consequently, both model identification and the date of its execution must be kept. Another threat to internal validity is prompt contamination. If the critic gets access to test examples, then the evaluation will become biased. Therefore, such separation needs to be implemented programmatically, rather than depending on researcher's self-control.

B. External Validity

The limitation to just one model, little data, and at most three refinements restricts the external validity of our results. Different LLMs will produce critic of different qualities, and different tasks with prompts will require additional development examples. Additionally, the methodology will perform differently in the long-form generation where the exact match accuracy will not be a proper metric for evaluating the semantic quality.

C. Novelty Validity

Prompt optimization is a well-explored task. There are several existing automated ways of prompt improvement: ProTeGi, OPRO, Promptbreeder, RiOT, Promptomatix, ZERA, and GEPA . Thus, novelty of the current work is intentionally restricted to evaluating the simple adaptive stopping controller with single-candidate refinement loop. The APRA method cannot be treated as the first prompt optimizer nor as a better solution than others in any conditions.Novelty Validity

Prompt optimization is an established research area. ProTeGi, OPRO, Promptbreeder, RiOT, Promptomatix, ZERA, and GEPA all provide prior mechanisms for automated prompt improvement . The novelty claim here is intentionally limited to the evaluation of a simple adaptive stopping controller around a single-candidate refinement loop. The work should not be described as the first prompt optimizer or as universally superior to existing systems.

D.Statistical Validity

With 50 examples in the test dataset, there is a rather coarse resolution of the accuracy — every example will shift the accuracy metric by two percentage points. Therefore, there should be an explanation of how thresholds were set and uncertainties assigned..

V. Discussion

The motivation behind APRA stems from the simple fact that optimization is an inference problem in and of itself. In many APO pipelines, search algorithm, critic, candidate generator, and evaluator are all distinct entities, whereas the continuation choice is predetermined via some fixed number of rounds. The decision is made explicit by APRA. This does not mean that fixed budgets are bad choices; they can still be employed when benchmark protocol calls for equal amount of computations or if a fixed depth of search is desired. APRA merely presents an alternative choice that carries low overhead and can be analyzed independently.

There is a fundamental difference between optimization quality and optimization efficiency. An algorithm can yield better results, but consume far more inference power compared to another, or vice versa, yield poorer results, but use far less computational resources. Hence, measuring either one of the two aspects alone does not give a full account of the optimization process. APRA analyzes both aspects through looking at quality obtained and the amount of computational power necessary to obtain that quality.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

It follows that the accuracy-cost couple is the main object to be analyzed. Such an approach is in line with recent research where cost, size of the sample, or the number of rollouts is taken into account while optimizing prompts .

The technique is intentionally conservative about later prompt rewrites. It preserves the best development prompt in order not to replace the latter with some bad rewrite. However, the tracking of the best prompt cannot stop overfitting on the development set. In case the development set is relatively small, it is possible that the prompt will become better simply by specializing in the failures chosen as samples. The optimization technique becomes more complicated and, therefore, a separate validation level is needed.

Reflective prompt optimization is also a subject of recent studies. Liu et al. prove that reflective optimization may fail if a defective prompt is used as a starting point and put forward the corresponding hypotheses and explore-exploit approaches . If APRA terminates without improvements over Base, then the stopping strategy was not helpful in this particular case.

One possible extension of the algorithm would be to use a different threshold criterion that takes into account the degree of uncertainty. It might calculate a confidence interval for the difference in development scores and terminate if the likelihood of significant improvement becomes lower than some predefined value. One can also extend the APRA algorithm by making the threshold cost-sensitive and thus requiring more evidence before an expensive iteration.

.

VI. Conclusion

In the current paper, we presented APRA, a continuation controller for adaptive automatic prompt refinement of Generative AI models. APRA works with one candidate at a time, generates one prompt refinement in each iteration based on failures data in the development set, memorizes the best candidate prompt found so far and stops when the marginal improvement is smaller than the threshold for a certain number of patience iterations. APRA is a straightforward controller that can be utilized above any single-candidate based prompt refinement approach.

The proposed evaluation methodology is aimed to address an empirical question of whether adaptive continuation can decrease the effort spent on prompt tuning without harming the performance on the hold-out task. That is why the experiments are consistent with the development/test split, have zero-shot baselines, consider the fixed-rounds alternative, are performed deterministically if possible, are focused on accuracy, number of calls made, number of tokens generated and runtime. As there are no artificial experimental results.

Acknowledgment

The author would like to sincerely thank all the researchers, academicians, developers, and open-source communities whose research and contributions have helped in the making of this project. The research done and innovations in automatic prompt optimization, reflective reasoning, iteration, and fast inference in Large Language Models (LLMs) have been highly informative and instructive throughout this project.

All the research papers, technical articles, documentation, and open-source projects made freely available by them have helped in learning the concepts related to this project and experimenting with various methods for improving the efficiency and performance of language models.

Apart from that, the author would like to acknowledge the developers and the contributors who keep updating the open source software toolkits and frameworks that play an important role in the implementation of new techniques.

The existence of such a vast amount of information has made it possible for the students, researchers and the developers to explore, experiment and implement new techniques in their research.

This research has been carried out based on the knowledge gained and learned from the work done by the research community. The author is grateful to the research community for its efforts in the development of this work.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

REFERENCES

[1] R. Pryzant, D. Iter, J. Li, Y. Lee, C. Zhu, and M. Zeng, "Automatic Prompt Optimization with 'Gradient Descent' and Beam Search," in *Proc. EMNLP*, pp. 7957–7968, 2023.

[2] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large Language Models as Optimizers," arXiv:2309.03409, 2023.

[3] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rocktaschel, "Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution," arXiv:2309.16797, 2023.

[4] K. Shum, S. Diao, and T. Zhang, "Automatic Prompt Augmentation and Selection with Chain-of-Thought from Labeled Data," in *Findings of EMNLP*, pp. 12113–12139, 2023.

[5] T. Zhang, J. Yuan, and S. Avestimehr, "Revisiting OPRO: The Limitations of Small-Scale LLMs as Optimizers," in *Findings of ACL*, pp. 1727–1735, 2024.

[6] W. Hu, Y. Shu, Z. Yu, Z. Wu, X. Lin, Z. Dai, and B. K. H. Low, "Localized Zeroth-Order Prompt Optimization," arXiv:2403.02993, 2024.

[7] C. Zhou, Z. Shi, Y. Yao, L. Liang, H. Chen, and Q. Zhang, "RiOT: Efficient Prompt Refinement with Residual Optimization Tree," in *Proc. ACL*, pp. 22307–22323, 2025.

[8] R. Murthy, M. Zhu, L. Yang, J. Qiu, J. Tan, S. Heinecke, C. Xiong, S. Savarese, and H. Wang, "Promptomatix: An Automatic Prompt Optimization Framework for Large Language Models," arXiv:2507.14241, 2025.

[9] S. Yi, M. Khang, and S. Park, "ZERA: Zero-init Instruction Evolving Refinement Agent – From Zero Instructions to Structured Prompts via Principle-based Optimization," in *Proc. EMNLP*, pp. 23323–23337, 2025.

[10] L. A. Agrawal, S. Tan, D. Soylu, N. Ziems, R. Khare, K. Opsahl-Ong, A. Singhvi, H. Shandilya, M. J. Ryan, M. Jiang, C. Potts, K. Sen, A. G. Dimakis, I. Stoica, D. Klein, M. Zaharia, and O. Khattab, "GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning," arXiv:2507.19457, 2025.

[11] K. Ramnath, K. Zhou, S. Guan, S. S. Mishra, X. Qi, Z. Shen, S. Wang, S. Woo, S. Jeoung, Y. Wang, H. Wang, H. Ding, Y. Lu, Z. Xu, Y. Zhou, B. Srinivasan, Q. Yan, Y. Chen, H. Ding, P. Xu, and L. L. Cheong, "A Systematic Survey of Automatic Prompt Optimization Techniques," in *Proc. EMNLP*, pp. 33078–33110, 2025.

[12] W. Cui, J. Zhang, Z. Li, H. Sun, D. Lopez, K. Das, B. A. Malin, and S. Kumar, "Automatic Prompt Optimization via Heuristic Search: A Survey," arXiv:2502.18746, 2025.

[13] S. Liu, Q. Xia, Q. Xia, Y. Liu, X. Yu, and R. Qu, "Reflection in the Dark: Exposing and Escaping the Black Box in Reflective Prompt Optimization," in *Proc. ACL Student Research Workshop*, pp. 76–109, 2026.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)