Controlling the Mutation in Large Language Models for the Efficient Evolution of Algorithms

Haoran Yin¹, Anna V. Kononova¹, Thomas Bäck¹, and Niki van Stein¹,

LIACS, Leiden University, The Netherlands h.yin@liacs.leidenuniv.nl

Abstract. The integration of Large Language Models (LLMs) with evolutionary computation (EC) has introduced a promising paradigm for automating the design of metaheuristic algorithms. However, existing frameworks, such as the Large Language Model Evolutionary Algorithm (LLaMEA), often lack precise control over mutation mechanisms, leading to inefficiencies in solution space exploration and potentially suboptimal convergence. This paper introduces a novel approach to mutation control within LLM-driven evolutionary frameworks, inspired by theory of genetic algorithms. Specifically, we propose dynamic mutation prompts that adaptively regulate mutation rates, leveraging a heavy-tailed powerlaw distribution to balance exploration and exploitation. Experiments using GPT-3.5-turbo and GPT-40 models demonstrate that GPT-3.5turbo fails to adhere to the specific mutation instructions, while GPT-40 is able to adapt its mutation based on the prompt engineered dynamic prompts. Further experiments show that the introduction of these dynamic rates can improve the convergence speed and adaptability of LLaMEA, when using GPT-40. This work sets the starting point for better controlled LLM-based mutations in code optimization tasks, paving the way for further advancements in automated metaheuristic design.

Keywords: Metaheuristic Algorithms, Algorithm Evolution, Mutation Control, Large Language Models, Evolutionary Computation.

1 Introduction

The development of metaheuristic algorithms [3] has been foundational for tack-ling complex optimization problems across domains such as engineering [22], artificial intelligence [16], and computational biology [17]. Traditionally, these algorithms have relied on expert-crafted designs and manual tuning, a process that, while effective, is labor-intensive and constrained in scalability. Recent advances in Large Language Models (LLMs) [6,23] have brought new opportunities to this field, enabling automated generation and refinement of metaheuristics through natural language processing capabilities. The synergy between LLMs and evolutionary computation (EC) [21] represents a transformative shift, leveraging LLMs for generating and enhancing code, while EC methods provide global search and iterative improvement capabilities.

A variety of methodologies exemplify this integration. The Large Language Model Evolutionary Algorithm (LLaMEA) [18] employs LLMs to iteratively generate, mutate, and select optimization algorithms, outperforming state-of-the-art methods in benchmark tasks. Similarly, the Algorithm Evolution using Large Language Models (AEL) framework and the Evolution of Heuristics approach [13,14] demonstrates how LLMs can autonomously evolve heuristics using incontext learning and feedback loops. Further, approaches such as Evolutionary Optimization with LLMs (EvoLLM) [12] show how prompt engineering and mutation operators enhance LLMs' ability to navigate search spaces effectively. These studies illustrate the potential for LLM-driven algorithmic frameworks to automate tasks traditionally requiring significant domain expertise.

Despite these advancements, existing frameworks often exhibit inefficiencies in exploring the solution space. A key limitation lies in the lack of precise control over mutation mechanisms during algorithm evolution, which can lead to redundant evaluations and suboptimal convergence. Addressing this gap, this paper introduces a novel mutation control mechanism for LLM-driven evolutionary frameworks. Drawing inspiration from genetic algorithms where a mutation rate sampled from a heavy-tailed power-law distribution is most beneficial [7], we propose dynamic mutation prompts that adaptively modulate the mutation rate to balance exploration and exploitation, aiming to enhance the convergence speed of LLaMEA.

2 Related Work

The generation and optimization of metaheuristic algorithms using LLMs has become a research hotspot in the field of research that explores the integration of large-scale language models with evolutionary computation, including EoH and LLaMEA [18,14,21,19]. LLaMEA is the basis of our work as it provides an open-source modular framework that we can easily expend on. It is designed to utilize LLMs to automatically generate and improve optimization algorithms. The core idea is to achieve steady improvement of algorithm performance through an automated evolutionary process driven by language models, and an evolutionary strategy (1, 1) or (1 + 1), including algorithm generation, mutation and selection [4]. Specifically, the LLaMEA process consists of the following steps:

- Generation: The LLM models generate the algorithms based on the generation prompt. This prompt can include problem descriptions, constraints, and a framework of expected solutions. By adjusting the prompt, the model can be guided to generate more targeted algorithmic candidates.
- Mutation: When LLMs are tasked with generating a new algorithm, they receive not only the initial generation prompt, but also mutation prompts simultaneously. These mutation prompts guide the model to either make slight refinements to the existing algorithm or completely redesign it. The latter approach is analogue to a restart in Evolutionary Algorithm optimization, thereby enhancing the model's exploration capabilities.

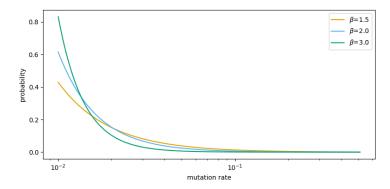


Fig. 1: The distribution of Equation 1. The area under the curves are all 1.

— Selection: For the best current and newly generated algorithms, LLaMEA evaluates them based on performance metrics. Only the algorithm that excels in multidimensional metrics is allowed to proceed to the next iteration. This performance feedback-based selection mechanism enables LLaMEA to continuously optimize the quality of the generated algorithms.

Through this iterative process, LLaMEA enables the algorithms to evolve themselves, demonstrating greater efficiency and flexibility in solving complex problems [18]. This framework not only reduces development time, but also generates algorithmic solutions with high performance on specific tasks, which has strong potential for practical applications. However, research in LLaMEA so far has mainly focused on the automatic generation of algorithms, while careful control of the mutation behavior during their evolution has been neglected, which makes the results of generation uncontrollable and leads to a potential waste of computational resources and money.

To explore the potential improvement of the mutation mechanism in LLaMEA, the choice of mutation rate is crucial. Traditional genetic algorithms recommend a fixed mutation rate, which is generally effective enough in simple unimodal optimization problems [10,3]. However, for complex and multimodal optimization problems, dynamically adjusting the mutation rate has been shown to provide significant performance gains [11,5]. Therefore, we refer to the fast mutation operator $fmut_{\beta}$ from B. Doerr et al. [7], to help improve the convergence speed of LLaMEA. $fmut_{\beta}$ is a heavy-tailed mutation operator for mutation steps, which is specifically beneficial for dealing with multimodal functions, where static mutation rates may not be efficient [9,2]. It employs a randomized mutation rate α/n , where α is an integer that follows a power-law distribution that does not concentrate strongly around its mean, and n represents the length of the bit string. This allows for a greater range of mutation strengths, potentially improving the algorithm's ability to escape local optima and explore the fitness landscape more effectively. This power-law probability distribution is represented

as Equation 1, and its distribution is illustrated in Figure 1:

$$Pr(\alpha/n) = \left(\sum_{i=1}^{\lfloor n/2\rfloor} i^{-\beta}\right)^{-1} \alpha^{-\beta} \tag{1}$$

Studies show that using a heavy-tailed mutation rate significantly reduces the optimization time for complex problems compared to a fixed mutation rate, and this approach has been proven to optimize functions more efficiently, particularly for complex problems with multiple local optima [7].

3 LLaMEA with Dynamic Mutation Rate

By integrating dynamic mutation rates from discrete power-law distributions into the LLaMEA framework, we can simulate the mutation process in genetic algorithms with a view to improving the performance and adaptability of the algorithms to complex and multimodal optimization problems.

The effort of introducing a dynamic mutation rate to LLaMEA starts with the refinement of the existing mutation prompt. In the LLaMEA framework, each generated algorithm is produced by a LLM using the parent algorithm and a mutation prompt, but this mutation prompt is simply 'Either refine or redesign to improve the algorithm'. To make this prompt more specific, we modified it to include the information about the mutation rate we want. The question of whether LLMs can correctly understand and accurately execute the mutation prompt needs to be investigated. Once this is established, mutation rate should be probabilistic and follow a given distribution, inspired by $fmut_{\beta}$.

We start with experiments to establish whether LLMs can achieve a fixed mutation rate when given a mutation prompt. We measure the code mutation effect by: first computing the logarithm of the ratio of the percentage of code modification of the actual mutated code (child) to that of the initial code (parent) to the requested mutation rate and then calculating the mean squared error (MSE) as shown in Equation 2:

$$MSE(x_i) = \frac{\sum_{j=2}^{N} \left(\log \frac{x_i^j}{x_i} - \log \frac{x_i}{x_i} \right)^2}{N-1} = \frac{\sum_{j=2}^{N} \log^2 \frac{x_i^j}{x_i}}{N-1}$$
(2)

where x_i is requested mutation rate and x_i^j is delivered code difference by LLaMEA of j-th algorithm code, N is the number of generated algorithm codes. To account for the distribution of mutation rates, we define target-distribution-weighted score (TDW-score) to measure the overall effect of different prompts as shown in Equation 3:

$$TDW-score(prompt) = \frac{\sum_{i=1}^{M} w_i MSE(x_i)}{M}$$
 (3)

where x_i is *i*-th mutation rate we experiment with, $w_i = \frac{Pr(x_i)}{\sum_{j=1}^{M} Pr(x_j)}$ is the weight of $MSE(x_i)$ and M is the number of experimented mutation rates. With

the MSE and the TDW-score, we can visualize the difference between the effects of different investigated prompts more intuitively.

We can then explore the impact of mutation prompts on the LLaMEA generation algorithm when combined with the dynamic mutation rate. We set the number of lines of the parent code to n in Equation 1, and the mutation rate obtained by sampling from Equation 1 at each mutation operation replaces x in the mutation prompt, and we use the same performance metric as LLaMEA [18], Area Over the Convergence Curve (AOCC), as introduced in [20].

3.1 Manually Constructed Prompts

To explore how to make mutation prompts effective, we start with the most basic and intuitive **prompt 1**, which contains only a description of the mutation task and a indication of the mutation rate. Then, step by step, we continue to add new content, including mandatory clarification (**prompt 2**), strict limitations (**prompt 3**), example specification (**prompt 4**), and context-specific requirements (**prompt 5**). The prompts are shown below:

Manual prompts

- **Prompt 1** Now, refine the strategy of the selected solution to improve it. Make sure that you only change x% of the code.
- **Prompt 2** Now, refine the strategy of the selected solution to improve it. Make sure that you only change x% of the code. This changing rate x% is the mandatory requirement.
- **Prompt 3** Now, refine the strategy of the selected solution to improve it. Make sure that you only change x% of the code. This changing rate x% is the mandatory requirement, you cannot change more or less than this rate.
- **Prompt 4** Now, refine the strategy of the selected solution to improve it. Make sure that you only change x% of the code, which means if the code has 100 lines, you can only change x lines, and the rest lines should remain the same. This changing rate x% is the mandatory requirement, you cannot change more or less than this rate.
- **Prompt 5** Now, refine the strategy of the selected solution to improve it. Make sure that you only change x% of the code, which means if the code has 100 lines, you can only change x lines, and the rest lines should remain the same. For this code, it has n lines, so you can only change $\lfloor n \times x/100 \rfloor$ lines, the rest $n \lfloor n \times x/100 \rfloor$ lines should remain the same. This changing rate x% is the mandatory requirement, you cannot change more or less than this rate.

3.2 Automatic Generated Prompts

Popular research at the forefront has shown that LLMs themselves have capabilities comparable to human prompt engineers [24]. Therefore, we also briefly

explore initially the effects of LLM-generated mutation prompts. We passed the following meta-prompt to ChatGPT 4 [1]:

Meta prompt

Now, imagine yourself as a prompt engineer, you give $\langle \text{LLM} \rangle$ a piece of optimization algorithm code, and you want $\langle \text{LLM} \rangle$ to modify it by x% (where x is a predefined number from 2 to 40, and indicated the code difference between the new one and the old one) exactly to improve the algorithm performance (meaning optimization convergence speed, not code efficiency, for example, by trying different mutation, selection, etc. strategies), what prompt would you give? Please give me at least 3 examples. Do not propose any specific directions or elements to change, since we want to cover the whole algorithm search space.

where <LLM> should be replaced with the specific model name when using this meta-prompt.

4 Experimental Setup

Large Language Models

This study uses two different large language models, GPT-40 and GPT-3.5-turbo, which have advanced capabilities and performance in generating metaheuristic optimization algorithms.

- GPT-40-2024-08-06: this is OpenAI's latest large-scale multi-modal language model, known for its powerful ability to understand and generate complex code.
- GPT-3.5-turbo-0125: an earlier version of the GPT model, which lags behind the newer models but still shows good performance when dealing with complex tasks.

We choose these two models due to their popularity and ease of implementation. In future work it would be interesting to extend the experimental setup with additional (open-weight) models.

Choice of Fixed Mutation Rate

Since we expect to achieve dynamic mutation rates sampled from a heavy-tailed power-law distribution, and with a maximum mutation rate of no more than 50% of the distribution, we choose 2%, 5%, 10%, 20%, and 40% as important references we use to determine how well the LLM is able to mutate the code according to these rates.

Automatically Generated Prompts

Using the meta-prompt of Section 3.2, we obtain **prompt 6** to **prompt 11**, where **prompt 6** to **prompt 8** correspond to the model GPT-3.5-turbo and **prompt 9** to **prompt 11** correspond to the GPT-40 model. These prompts are shown below:

Generated prompts for GPT-3.5-turbo

- **Prompt 6** Adjust the code such that the algorithm's convergence speed is improved, while ensuring that the changes result in an exact difference of x% compared to the original code. This difference should reflect the modification in functionality, not code style or syntax. Feel free to adjust any part of the algorithm (e.g., initialization, selection, mutation, or other components) to achieve faster convergence while maintaining the specified code difference.
- **Prompt 7** Modify the optimization algorithm code to improve its performance in terms of convergence speed. The modification should result in a code difference of exactly x%. Ensure that the changes are meaningful to enhance optimization speed without focusing on code efficiency or readability improvements. Explore any strategy within the algorithm to achieve this, but keep the difference precisely at the specified percentage.
- **Prompt 8** Please enhance the convergence speed of the optimization algorithm given below by modifying it. The modifications should introduce a code difference of precisely x% compared to the original code. Focus on optimizing the algorithm's behavior rather than its implementation efficiency. You are free to explore any area of the algorithm's logic, but ensure that the total code difference remains exactly at x% and is geared toward faster convergence.

Generated prompts for GPT-40

- **Prompt 9** Here's a piece of code for an optimization algorithm. Please modify it by exactly x% to improve the algorithm's performance in terms of optimization convergence speed. Focus on introducing meaningful changes that can potentially enhance its effectiveness, such as exploring alternative strategies or approaches across any aspect of the algorithm. Keep the modifications strictly within the specified x% range for code difference while striving for faster convergence.
- **Prompt 10** Take this code of an optimization algorithm and adjust it by x% to improve convergence speed. Make sure the modifications cover a broad spectrum of possible algorithm adjustments, considering changes across different components without exceeding x% in code difference. Your changes should aim to improve the algorithm's ability to reach optimal solutions more quickly.
- **Prompt 11** Please transform this optimization algorithm code by exactly x% in a way that enhances convergence speed. Keep the code difference precisely at x%, and focus solely on achieving performance improvements through algorithmic adjustments across various elements of the code. Avoid focusing on code efficiency; instead, prioritize exploration of diverse approaches within the allowed modification percentage.

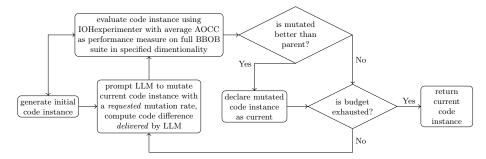


Fig. 2: LLaMEA with controlled mutation via dynamic prompts.

4.1 Experimental design

In our experiments, LLaMEA will be applied to the following scenario: using an evolutionary strategy (1+1) to propose and improve metaheuristic algorithms to solve single-objective continuous optimization problems, which is shown in Fig. 2. In each iteration, LLaMEA mutates the parent metaheuristic algorithm to obtain a child algorithm, and only the better algorithm of them can be the new parent for the next iteration. To determine which algorithm is better, the parent and child will be tested for their effectiveness with the widely used Black-Box Optimization Benchmarking (BBOB) test set [8], as well as with the IOHexperimenter platform [15], with mean AOCC as the measure of algorithm performance [20]. The dimensionality is set to 5 for all BBOB problems, the tests cover 3 instances of each problem, and the generated metaheuristic algorithms are repeated three times to eliminate experimental randomness and ensure reliable results.

The experiments consisted of two parts: First, we explore the reliability of the mutation prompts. The second experiment aims to explore the possible enhancements that the dynamic mutation rate brings to LLaMEA. The following is the setup of the first experiment:

Models GPT-3.5-turbo-0125 and GPT-4o-2024-08-06

Evolutionary strategy (1+1)

Requested mutation rates 2%, 5%, 10%, 20% and 40%

Prompts used a total of 11 different prompts, of which 5 are manually constructed to be generic and automatically generated to be model-specific, i.e., both GPT-3.5-turbo and GPT-40 are experimented with 8 prompts.

Number of independent runs the experiment is repeated 3 times for each combination (model, requested mutation rate, prompt).

Code generation budget 100 code instances are generated for each configuration.

After finishing the first experiment, we select the best-performing manually constructed prompts and automatically generated prompts for GPT-3.5-turbo and

GPT-40, respectively, and then experiment the effect of using a dynamic mutation rate on the basis of these prompts in LLaMEA. The setup for this followup experiment is as follows:

Models GPT-3.5-turbo-0125 and GPT-4o-2024-08-06

Evolutionary strategy (1+1)

Requested mutation rates dynamic mutation rate, with $\beta = 1.5$ in Equation 1

Prompts used the best-performing manually constructed prompts and automatically generated prompts for GPT-3.5-turbo and GPT-40, respectively

Number of independent runs the experiment is repeated 5 times for each combination (model, prompts).

Code generation budget 100 code instances are generated for each configuration.

Through this exhaustive experimental setup, we aim to comprehensively assess the role of dynamic mutation in automated algorithm generation and its benefits.

5 Results

5.1 Reliability of Mutation Prompts

In Fig. 3, the MSE and TDW-scores are shown. x-axis and y-axis represent requested mutation rates and prompts, respectively. For both Figs. 3a and 3b, the first five columns from the left illustrate MSE values, and the last column on the right shows the TDW-score, which is the most important metric for selecting the prompts. The smaller the value, the better the prompt. Thus, we can find that prompt2 is the best manually constructed prompt and prompt7 is the best automatic generated prompt for GPT-3.5-turbo, but for GPT-4o, prompt5 and prompt9 are the best of manually constructed prompts and best automatic generated prompts, respectively.

In addition, there is a very noticeable difference for both models: the MSE and TDW-scores of GPT-3.5-turbo are significantly inferior to those of GPT-4o. In order to explore the details behind these numbers in depth, we show in detail the distribution of code difference of the codes generated with different prompts and requested mutation rates, as well as a scatterplot of the ratio of actual delivered code difference to requested mutation rate for LLaMEA, displayed in Fig. 4.

In Figs. 4a and 4b, the red dashed line represents the requested mutation rate, and the green dotted-dashed line indicates that the delivered code difference of the points on this line is equal to the requested mutation rate. Each column of data contains mutated codes from the 100 codes generated in 3 independent runs using LLaMEA and the requested mutation rate in each prompt.

From Fig. 4a, we can see that progressively more complex and accurate artificially constructed prompts do not improve the performance of LLaMEA with

GPT-3.5-turbo. Moreover, it seems that GPT-3.5-turbo does not understand the mutation rate variation well. The code difference of the code generated by GPT-3.5-turbo always stays high regardless of whether the requested mutation rate is high or low, and the distribution is not concentrated.

From Fig. 4b, we find that the distribution of code difference for GPT-4o is more concentrated compared to the result of GPT-3.5-turbo, and also closer to the requested mutation rates, suggesting that GPT-4o is more powerful. Therefore, GPT-4o is a significantly better choice than GPT-3.5-turbo for LLaMEA when economic conditions allow. Second, as the artificially constructed prompts become more complex (longer), the delivered code difference gets closer to the mutation rate, that is, the scatterplot gets closer and closer to the green dotted dashed line, especially when the mutation rate is 2%, 5%, and 10%. This is encouraging, and it shows that GPT-4o understands the task of mutation prompts, and that prompt engineering is effective for mutation prompts and GPT-4o performs the mutation task quite well.

5.2 Influence of Dynamic Mutation Rate

Fig. 5 compares the convergence speed of the baselines and LLaMEA with the dynamic mutation rate as introduced in Section 3 using the best mutation prompt as identified before. Baselines are raw data directly from [18], which represent the default setting of LLaMEA without dynamic mutation rate. Shaded areas denote the standard deviation of the best-so-far.

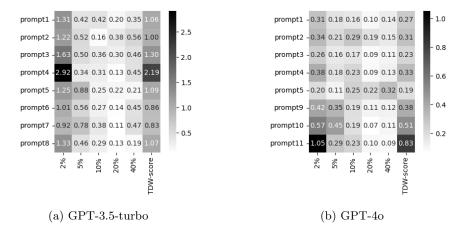


Fig. 3: MSE and TDW-scores of prompts. For both figures, the x-axis is the requested mutation rates, the y-axis is the prompts, the first five columns from left show the corresponding MSE of different prompts with requested mutation rate, and the last column on the right shows the TDW-score of prompts. The smaller the value, the better the prompt.

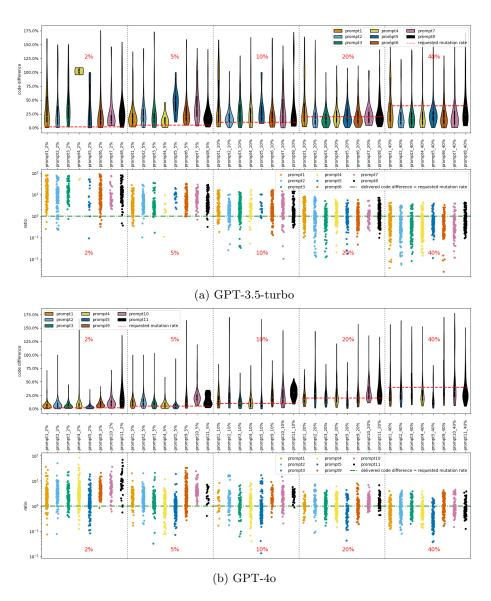


Fig. 4: The distribution of code difference of the codes generated with different prompts and requested mutation rates, and scatterplot of the ratio of actual delivered code difference to requested mutation rate for LLaMEA. Results of different mutation rates are separated by gray dotted lines and marked by red text. The red dashed line represents the requested mutation rate, while the green dotted-dashed line shows that the difference in the delivered code for points on this line equals the requested mutation rate. Each data column contains mutated codes from 100 codes generated over 3 runs.

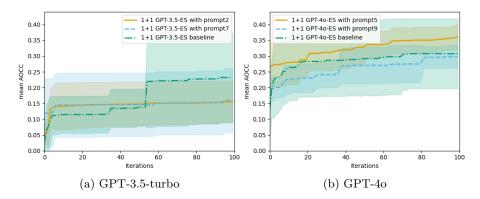


Fig. 5: Mean convergence curves (best-so-far algorithm scores) over the 5 different runs for each selected prompt and LLM. Baselines are the raw data from [18]. Shaded areas denote the standard deviation of the best-so-far.

From Fig. 5, we can observe that the manually constructed mutation prompt with dynamic mutation rate using GPT-40 improves the performance of LLaMEA with the (1+1) evolutionary strategy, but on the other hand, the automatically generated mutation prompt is not much different from the baseline (even slightly worse). Considering that this experiment is relatively simple automatic prompt engineering, there is still room for exploring more efficient automatic prompt generation methods to define better mutation prompts.

For the GPT-3.5-turbo model, the prompt used seems to have little effect on the convergence speed, and both are weaker than the baseline (though not significantly). The overall performance of GPT-3.5-turbo is also significantly weaker than that of GPT-40, which implies that the comprehension capacity of GPT-3.5-turbo becomes a bottleneck in improving the effectiveness of LLaMEA.

Figure 6 shows the change in the code difference of the code generated by LLaMEA after applying the prompts of the dynamic mutation rate. Dynamically required mutation rates are generally small, and compared to GPT-3.5-turbo, GPT-40 is significantly easier to mutate at the required mutation rate. Without specifying a specific value for the mutation rate, both GPT-3.5-turbo and GPT-40 are more likely to carry out extreme mutations, as shown in Figs. 6e and 6f.

6 Discussions and Future Work

In this study, we observe that the design of mutation prompts has a significant impact on the ratio of code mutations applied by the LLM in an algorithm optimization task, and additionally influences the optimization performance. The experimental results show that there is a difference in performance between the manually constructed prompts and the automatically generated prompts, suggesting that the precise design of the prompts is crucial to the optimization

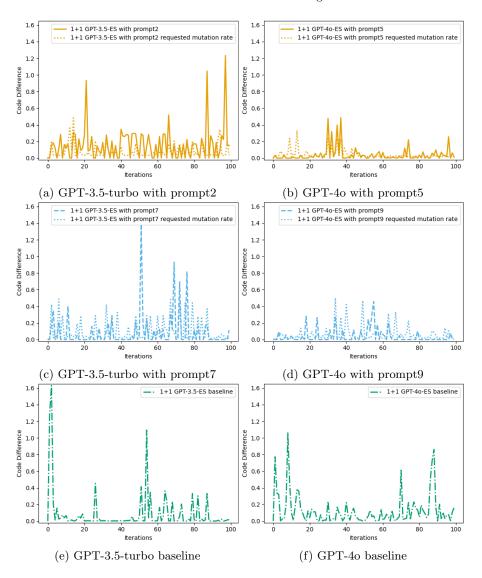


Fig. 6: Example code difference of 1 run out of 5 runs for each selected prompt and LLM. Baselines are the raw data from [18]. For Figs. 6a, 6b, 6c, and 6d, the dotted lines represent requested mutation rate.

effect of the algorithm. For the GPT-3.5-turbo model, the performance is not satisfactory even when the indication of the mutation rate is more explicit, which may be related to the model's ability to understand the mutation task.

The introduction of dynamic mutation rates can improve the adaptability of the LLaMEA algorithm and lead to better generated meta-heuristic algorithms. By experimental comparison, the LLaMEA framework using dynamic mutation rate has better convergence speed and optimization results than the default prompt without mutation rate when using GPT-4o. This finding supports the effectiveness of using dynamic control strategies in evolutionary algorithms. Despite the experimental success, there are still challenges in precisely controlling the mutation rate. In addition, advanced automatic prompt engineering techniques may require more development and testing in practice to produce and optimize more efficient algorithms.

Future research should consider developing more advanced automated cue engineering techniques. Research should also be extended to a wider variety and newer versions of local LLMs to assess the scalability and adaptability of mutation strategies and reduce funding consumption.

References

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
- 2. Antipov, D., Doerr, B.: Runtime analysis of a heavy-tailed genetic algorithm on jump functions. In: International Conference on Parallel Problem Solving from Nature. pp. 545–559. Springer (2020)
- 3. Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. Evolutionary computation 1(1), 1–23 (1993)
- 4. Beyer, H.G., Schwefel, H.P.: Evolution strategies—a comprehensive introduction. Natural computing 1, 3–52 (2002)
- Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the leadingones problem. In: Parallel Problem Solving from Nature, PPSN XI: 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I 11. pp. 1–10. Springer (2010)
- 6. Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., et al.: A survey on evaluation of large language models. ACM Transactions on Intelligent Systems and Technology **15**(3), 1–45 (2024)
- Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proceedings of the genetic and evolutionary computation conference. pp. 777–784 (2017)
- 8. Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Ph.D. thesis, INRIA (2009)
- 9. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary computation 9(2), 159–195 (2001)
- 10. Holland, J.H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)
- 11. Jansen, T., Wegener, I.: On the analysis of a dynamic evolutionary algorithm. Journal of Discrete Algorithms 4(1), 181-199 (2006)
- 12. Lange, R., Tian, Y., Tang, Y.: Large language models as evolution strategies. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 579–582 (2024)
- 13. Liu, F., Tong, X., Yuan, M., Zhang, Q.: Algorithm evolution using large language model. arXiv preprint arXiv:2311.15249 (2023)

- 14. Liu, F., Xialiang, T., Yuan, M., Lin, X., Luo, F., Wang, Z., Lu, Z., Zhang, Q.: Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In: Forty-first International Conference on Machine Learning (2024)
- 15. de Nobel, J., Ye, F., Vermetten, D., Wang, H., Doerr, C., Bäck, T.: IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics. arXiv e-prints:2111.04077 (Nov 2021), https://arxiv.org/abs/2111.04077
- 16. Ojha, V.K., Abraham, A., Snášel, V.: Metaheuristic design of feedforward neural networks: A review of two decades of research. Engineering Applications of Artificial Intelligence **60**, 97–116 (2017)
- 17. Penas, D.R., González, P., Egea, J.A., Banga, J.R., Doallo, R.: Parallel metaheuristics in computational biology: An asynchronous cooperative enhanced scatter search method. Procedia Computer Science **51**, 630–639 (2015)
- 18. van Stein, N., Bäck, T.: Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. IEEE Transactions on Evolutionary Computation pp. 1–1 (2024). https://doi.org/10.1109/TEVC.2024.3497793
- van Stein, N., Vermetten, D., Bäck, T.: In-the-loop hyper-parameter optimization for llm-based automated design of heuristics. arXiv preprint arXiv:2410.16309 (2024)
- van Stein, N., Vermetten, D., Kononova, A.V., Bäck, T.: Explainable benchmarking for iterative optimization heuristics. arXiv preprint arXiv:2401.17842 (2024)
- Wu, X., Wu, S.h., Wu, J., Feng, L., Tan, K.C.: Evolutionary computation in the era of large language model: Survey and roadmap. arXiv preprint arXiv:2401.10034 (2024)
- 22. Yang, X.S.: Optimization and metaheuristic algorithms in engineering. Metaheuristics in water, geotechnical and transport engineering 1, 23 (2013)
- Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al.: A survey of large language models. arXiv preprint arXiv:2303.18223 (2023)
- 24. Zhou, Y., Muresanu, A.I., Han, Z., Paster, K., Pitis, S., Chan, H., Ba, J.: Large language models are human-level prompt engineers. In: NeurIPS 2022 Foundation Models for Decision Making Workshop (2022), https://openreview.net/forum?id=YdqwNaCLCx