

Large Language Models As Evolution Strategies

Robert Tjarko Lange
 TU Berlin, Google DeepMind
 Germany, Japan
 robert.t.lange@tu-berlin.de

Yingtao Tian
 Google DeepMind
 Japan
 alantian@google.com

Yujin Tang
 Google DeepMind
 Japan
 yujintang@google.com

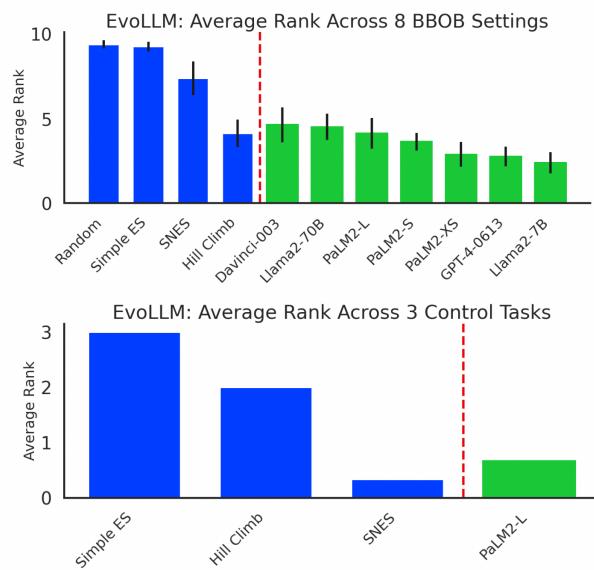
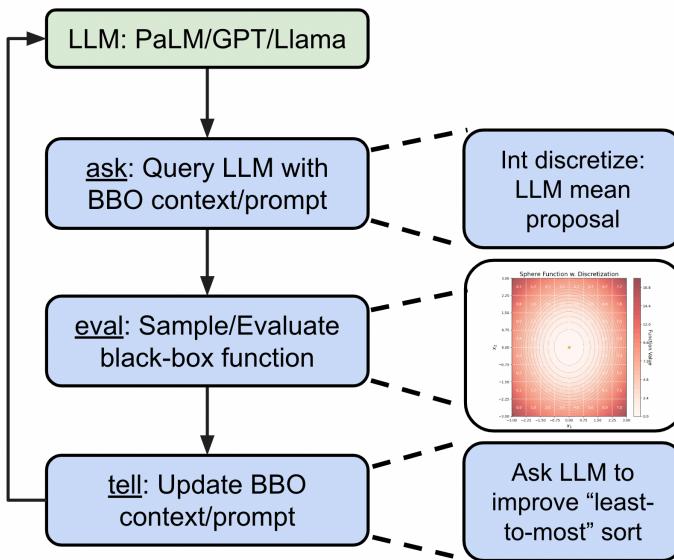


Figure 1: Left. Overview of EvoLLM procedure. An LLM proposes an ES search distribution update using a discretized search space & solutions sorted by their performance (least-to-most). To combat the context length growth in the number of dimensions, we can split the search dimensions into blocks and perform LLM batch queries. **Right:** Aggregated results across 8 BBOB [12] settings, and 3 neuroevolution control problems. The results are averaged over ten and five independent runs, respectively. LLM-based Evolution Strategies (green) outperform traditional baselines (blue).

ABSTRACT

Large Transformer models are capable of implementing a plethora of so-called in-context learning algorithms. These include gradient descent, classification, sequence completion, transformation, and improvement. In this work, we investigate whether large language models (LLMs), which never explicitly encountered the task of black-box optimization, are in principle capable of implementing evolutionary optimization algorithms. While previous works have solely focused on language-based task specification, we move forward and focus on the zero-shot application of LLMs to black-box optimization. We introduce a novel prompting strategy, consisting of least-to-most sorting of discretized population members and querying the LLM to propose an improvement to the mean statistic, i.e. perform a type of black-box recombination operation. Empirically, we find that our setup allows the user to obtain an LLM-based evolution strategy, which we call ‘EvoLLM’, that robustly outperforms baseline algorithms such as random search and Gaussian Hill Climbing on synthetic BBOB functions as well as

small neuroevolution tasks. Hence, LLMs can act as ‘plug-in’ in-context recombination operators. We provide several comparative studies of the LLM’s model size, prompt strategy, and context construction. Finally, we show that one can flexibly improve EvoLLM’s performance by providing teacher algorithm information via instruction fine-tuning on previously collected teacher optimization trajectories.

CCS CONCEPTS

- Computing methodologies → Evolutionary Robotics.

KEYWORDS

evolution strategies, machine learning

ACM Reference Format:

Robert Tjarko Lange, Yingtao Tian, and Yujin Tang. 2024. Large Language Models As Evolution Strategies. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Motivation. Recently, it has been demonstrated that language models trained on large text corpora are capable of impressive in-context learning [2]. For example, given a prompt of pattern demonstrations, LLMs can infer the underlying generating rules and even propose sequence improvements [32]. Unlike fine-tuning this property notably emerges with frozen model weights and only requires online context information. Interestingly, this in-context improvement property appears to be broadly applicable to abstract pattern sequences: In the context of in-silico evolution, LLMs can out-of-the-box act as code-level mutation [3], cross-over operations [31] or be embedded into the context of genetic programming [23]. To what degree can LLMs, trained on text, be generalized to behave like an optimization algorithm? Is it possible to have an LLM train the weights of a neural network as in evolutionary strategies (ES)?

Approach. Here, we leverage the recent advances in the understanding of LLMs as ‘general pattern machines’ [32] to construct a prompt strategy, which turns an LLM into a recombination operator: A purely text-trained LLM processes fitness-sorted sequences of function evaluations and their corresponding candidate solutions. Afterwards, we can simply ‘ask’ the LLM to propose the next mean statistic to sample from (see Figure 1, left). We term the resulting LLM-based Evolution Strategy *EvoLLM*. Concretely, we also propose to adapt in our approach integer-based, search space discretization, least-to-most prompting [47] and decision transformer-style [4] fitness improvement queries.

Results. Our experiments demonstrate that our proposed prompting strategy is sufficient to induce the LLM to conduct a recombination operation. The resulting *EvoLLM* successfully performs black-box optimization (BBO) on both synthetic BBOB [12] test functions and small neuroevolution tasks [1, 19] with various search space dimensions and evaluation budgets (see Figure 1, right). Furthermore, we provide rigorous ablation to the prompt design and provide general construction recommendations promoting better optimization behavior. Interestingly, we observe that among our selection of LLMs, there is a general trend indicating that smaller LLM models tend to outperform larger ones. We also show that choosing a sufficient solution representation is critical for in-context BBO powered by LLMs. Finally, LLMs are capable of leveraging additional information provided by supporting teacher algorithms.

Contributions. Our contributions are summarized as follows:

- (1) We introduce a general prompt approach that induces LLMs to act as ES. The prompt is composed of a discretized solution candidate representation, performance-based least-to-most sorting and a fitness improvement query (Figure 1).
- (2) We use a set of diverse LLMs and establish that language models can robustly perform zero-shot optimization on classic BBO and small neural network control tasks.
- (3) We investigate various prompting strategies and show that a discretized solution space representation greatly outperforms common natural language-based instructions. Furthermore, the *EvoLLM* approach is largely robust to the choice of search space discretization and context information.
- (4) We show that *EvoLLM*’s performance can be improved by fine-tuning the base LLM model on BBO trajectories generated by teacher algorithms.

2 RELATED WORK

In-Context Learning with Transformers. Various recent efforts have investigated the possibility of training large models from scratch to perform in-context learning. These methods rely on the impressive sequence modeling capabilities of the Transformer architecture [41] to model long-range dependencies. For example Kirsch et al. [13] show that Transformers can learn supervised in-context image classification algorithms. Laskin et al. [21] investigated in-context reinforcement learning and Lu et al. [29] considered state space models for large-scale meta-reinforcement learning.

In-Context BBO with Autoregressive Models. Chen et al. [5] trained RNN-based BBO algorithms using privileged access to gradient computations of the fitness function at training time. Furthermore, Chen et al. [6], Dery et al. [7] used large pre-collected datasets to directly train a T5 Encoder-Decoder architecture [35] for BBO, called OptFormer. Krishnamoorthy et al. [15] later on investigated the usage of generating offline data for training autoregressive BBO models. All of these approaches trained large sequence models leveraging large programmatically generated or augmented task spaces to induce in-context learning across long timescales. While this line of works is close to our proposed method, ours differs in that we use text-trained large models and investigate their ability to perform BBO without explicitly being trained on such tasks.

In-Context Learning with LLMs. LLMs are capable of few-shot learning given little text examples in their prompt [2]. Various prompting strategies and automation methods have subsequently been developed to improve task-specific performance. E.g. these include least-to-most sorting [47], chain-of-thought prompting [43] or self-consistency [42]. More recently, Mirchandani et al. [32] have shown that these LLM capabilities can also be elicited outside of the text-based prompting context. Indeed LLMs appear to be able to reason about abstract sequences of integers or even ASCII codes.

LLMs for Optimization. Related to our work, Nie et al. [33], Yang et al. [45], Zhang et al. [46] show that LLMs can be turned into text-based optimizers. Our work extends this line of work and shows that they can also be transformed into a recombination operator for ES and are capable of optimizing small neural networks. In the context of computational evolution, LLMs can out-of-the-box act as code-level or algorithm mutation [3, 27], cross-over operations [31], be embedded into the context of genetic programming [23] or evolve prompt strategies [10]. While preliminary work [26, 28] has started to investigate LLMs for Evolutionary Optimization, our work is the first to consider LLMs for ES, compares different base LLM models and various prompt construction approaches.

Learned Black-Box Optimization. ES-update rules are inherently set of operations, i.e. the order of the population members within a generation should not affect the performed distribution change. Self-attention provides a natural inductive bias for such an operation. Previously, Lange et al. [16, 17] constructed ES and GA algorithms, which used self- and cross-attention to process the information within a single generation. The attention parameters are meta-evolved on a small task distribution of BBO problems. As a comparison, our proposed method does not meta-train a new BBO algorithm but instead asks whether text-trained LLMs are capable of performing optimization with discretized prompt information provided.

3 BACKGROUND

Black-Box Optimization (BBO). Given a function $f(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}$ with unknown functional form, i.e. we cannot compute its derivative (or it is not well behaved or empirically infeasible to compute), BBO seeks to find its global optimum using only function evaluations, without derivatives:

$$\min_{\mathbf{x}} f(\mathbf{x}), \text{s.t. } \mathbf{x}_d \in [l_d, u_d] \subset [-\infty, \infty], \forall d = 1, \dots, D.$$

Throughout, we leverage a set of synthetic benchmark functions (BBOB [12] and classic control tasks [1] to evaluate BBOs.

Evolution Strategies. Evolutionary Optimizers (EO) are a set of BBO algorithms inspired by mechanisms of biological evolution. Roughly speaking, EO algorithms can be grouped into Evolution Strategies [36] and Genetic Algorithms, where the former focus on mutating real-valued parameters and often use self-adaptation, while the latter typically work with a population of binary-coded solutions and rely on crossover and mutation operators. Here, we focus on isotropic Gaussian ES. Given a population size N and a search distribution $\mu \in \mathbb{R}^D, \Sigma = \sigma \mathbf{1}_{D \times D} \in \mathbb{R}^{D \times D}$, ES sample a population of candidate solutions $X = [x_1, \dots, x_N] \in \mathbb{R}^{N \times D}$ at each generation. Afterwards, the performance (or fitness) of each candidate is evaluated on the task of interest and one obtains fitness scores $F = [f_1, \dots, f_N] \in \mathbb{R}^N$ (we denote $[f(x_1), \dots, f(x_N)]$ as $[f_1, \dots, f_N]$ for brevity). The search distribution is then updated to increase the likelihood of sampling well-performing solutions, $\mu', \sigma' \leftarrow \text{UPDATE}(\mu, \sigma, X, F, H)$, where H denotes a set of summary statistics constructed from the search history. There exist various types of ES including estimation-of-distribution [11], natural [38, 44] and finite-difference-based ES [37]. In this work, we investigate whether an LLM can represent a UPDATE operator using a context string constructed from H .

Transformer-Based Language Models. The Transformer [41] stacks blocks of multi-head self-attention (MHSA) operations, feed-forward MLP transformation, dropout, and layer normalization. At its core self-attention is a set operation that projects an input matrix $X \in \mathbb{R}^{T \times D}$ onto D_K -dimensional vectors $Q, K, V \in \mathbb{R}^{T \times D_K}$ called queries, keys, and values, respectively:

$$\begin{aligned} \text{Attention}(X) &= \text{softmax}\left(QK^T / \sqrt{D_K}\right)V \\ &= \text{softmax}\left(XW_Q(XW_K)^T / \sqrt{D_K}\right)XW_V. \end{aligned}$$

Permuting the rows of X will apply the same permutation to $\text{Attention}(X)$ [e.g., 14, 22, 39]. MHSA has quadratic complexity with respect to the context length T . In this work, we leverage large Transformer models trained on text data. Afterwards, we evaluate their ability to implement evolutionary improvement operations when presented with BBO trajectory information, i.e. discretized solution candidates and their fitness or rank. Hence, we ask whether in-context learning can resemble an efficient UPDATE operation.

Prompt Engineering & Vocabulary Tokenization. LLMs can behave sensitively concerning the specific context construction. This has led to terms such as ‘prompt engineering’. Throughout our investigations, we consider various prompt construction approaches to provide ablative insights into the underlying LLM mechanisms. More specifically and motivated by computational resource considerations, we conduct coordinate-wise evaluations in order to

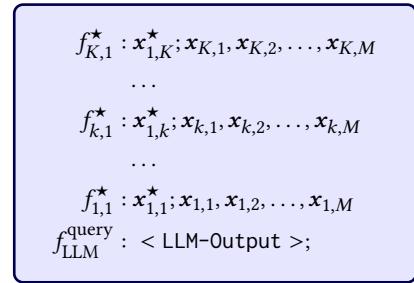
extract a well-performing approach. There exist several automated prompt tuning approaches including gradient-based soft prompt optimization [24], which requires access to LLM weights. Random search-based prompt optimization [30] and evolutionary optimization of prompts [8, 10, 25] have recently shown promising result.

Language models pre-process raw language strings into vector representations using so-called tokenizers. These tokenizers perform a type of compression based on natural language co-statistics. Naturally, raw high-precision floating point numbers used in BBO tend to be underrepresented. We therefore, propose a different integer-based representation approach.

4 TURNING LLMS INTO ES ALGORITHMS

In the following, we outline a prompt strategy that enables LLM-based optimization in the form of an Evolution Strategy. More specifically, we establish a prompt design space used to construct an LLM query, which makes the LLM represent the UPDATE operator for an ES. Afterwards, we show how this procedure can be incorporated into the simple ‘ask-evaluate-tell’ API common to many popular black-box optimization algorithms [18].

High-Level EvoLLM Prompt Design Space. We follow the paradigm outlined in Mirchandani et al. [32] and construct a LLM prompt by representing the solution candidates as integers resulting from a discretized search space with a pre-specified resolution (Figure 1). Note, that we don’t use raw floating point numbers due to the LLM tokenizer potentially returning different numbers of tokens per individual number. This can hinder the LLM from inferring improvement sequences. First, we sort the set of previous population evaluations $H = \{X_g, F_g\}_{g=1}^G$ by their fitness within and across generations. Afterwards, we select the top- K performing generations and top- M solutions within each generation. We let the LLM propose the next mean for a desired fitness level [4], $f_{\text{LLM}}^{\text{query}}$:



where x_k^\star, f_k^\star denotes the best-performing solution and its fitness up to generation k . We observed that LLMs robustly follow the pattern outlined in the prompt above and continue the string format by outputting a new mean x_{LLM} with the delimiter ;. Afterwards, we use the proposed mean to sample a new set of candidates, update the context statistics H and iterate the process. The LLM can thereby in-context adapt to accumulated search information and implement a novel type of recombination.

Detailed EvoLLM API. At each generation, ES samples a set of candidates, evaluates the population on the task at hand and afterward updates the search distribution. Here, we let the LLM perform the search update given text-context information H . The general procedure consists of the following steps (Figure 2):

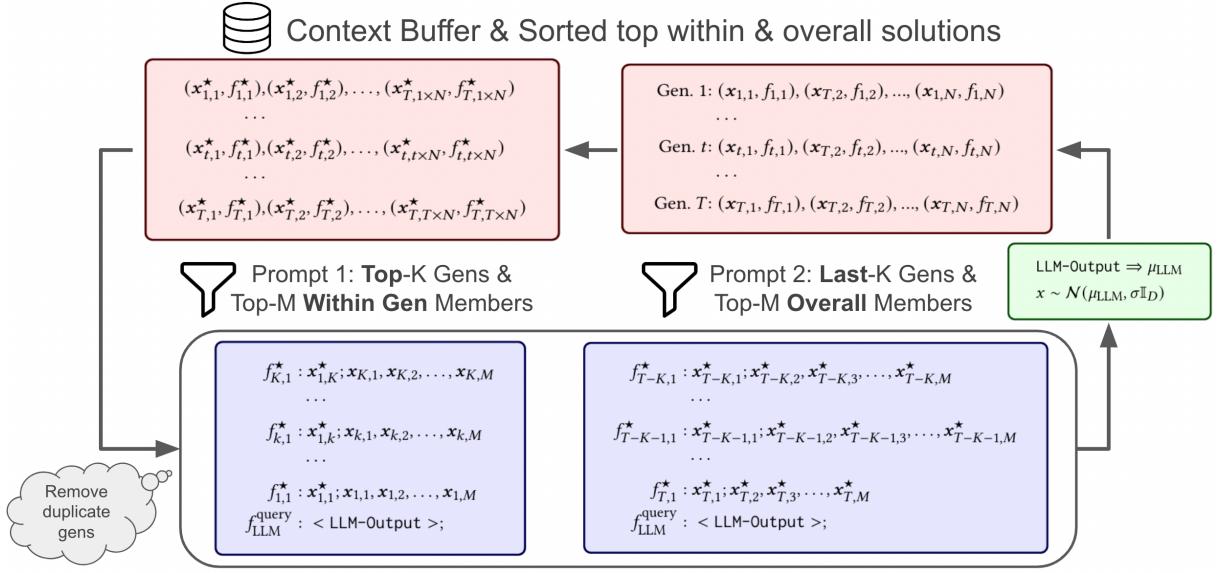


Figure 2: **EvoLLM prompt design space & API.** We track all solution evaluations and their performance in a context buffer. The buffer is used to construct query prompts for the LLM. After parsing the LLM output and sampling, we evaluate the resulting population and add the new information to the buffer. We provide an example of the generated prompts in the appendix.

- (1) **Context Buffer Warm-Up/Seeding.** We use standard BBO algorithm (here: random search) to fill up a context data buffer with an initial set of evaluations.
- (2) **Discretize & Augment Context Buffer.** We represent the solutions as discretized integers with a chosen resolution and track the evaluation candidates and their fitness scores.

Given the initial buffer with discretized solutions, we construct a string representation of the previous evaluations. The K generations and corresponding M candidates are selected and sorted as follows:

- (3) **Select & Sort Context Generations.**
 - (a) '**Random**'. The simplest option is to select previous generations from the buffer uniformly at random.
 - (b) '**Last**'. Alternatively, we select the most recent K generations evaluated on the problem.
 - (c) '**Best**'. Finally, we considered the generations, which led to the best-performing candidate solutions seen so far.
- (4) **Select & Sort Context Candidates.**
 - (a) '**Random**'. From the K selected generations we again select $M < N$ random population members.
 - (b) '**Best-Within-Generation**'. Alternatively, we select the best candidates evaluated within a given generation.
 - (c) '**Best-Up-To-Generation**'. Finally, we consider the entire evaluation history and provide the top- M candidate solutions evaluated up to the k -th generation.

Note, that the exact ordering of the generations and candidates can affect the ease with which the LLM may infer improving directions (e.g. momentum). Each generation of candidates is separated with a line break and each member with a .. Finally, we add a fitness improvement query (ca. 2 times the best previously seen fitness value). After querying the LLM, we parse the LLM-proposed mean update back into a floating point representation:

- (5) **Query LLM for Search Improvement.** We construct the prompt repeatedly at each generation, sample a temperature parameter and query the LLM. We pass the returned integer output back into the mean for the next generation. Occasionally, the parsing may fail. In this case, we use a back-up strategy of sampling around the previous best evaluation.
- (6) **Sample & Evaluate New Candidates.** We perturb the mean with isotropic noise and evaluate all population members. Afterwards, we add the information to the context buffer.

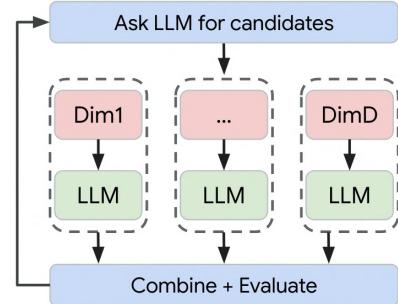


Figure 3: **Dimension-batched Querying of an LLM.** As the search space dimensionality grows, the context length can exceed the feasibilities of the LLM. We split the solution space into blocks and perform multiple LLM queries per update.

Scaling EvoLLMs to Larger Search Spaces. The context length of the used prompt quickly increases with the number of dimensions. Many LLMs have been trained with a context length with limits their applicability to longer horizons. We found that once the

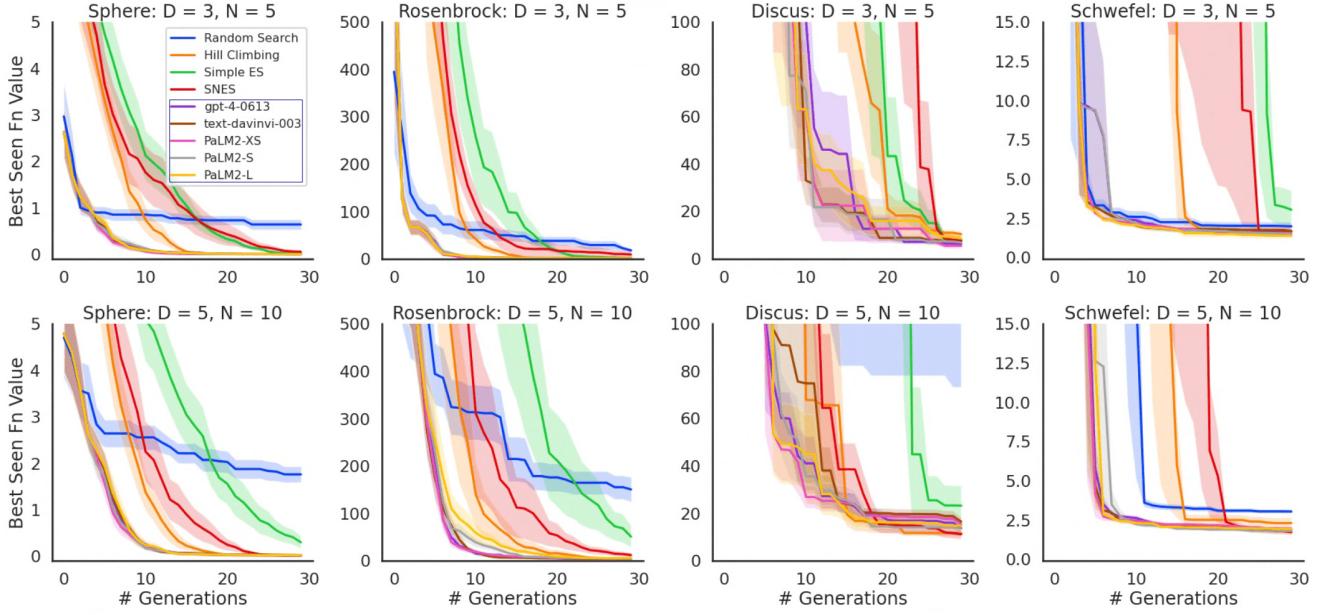


Figure 4: EvoLLM performance (lower is better) on BBOB [12] functions with single LLM query. We compare different LLM base models (marked in the lower box in the legend) and find that the behavior of EvoLLM is robust to the exact choice of LLM. The results are averaged across 10 independent runs.

context becomes too long, the LLM may output non-informative information. This in turn implies that either the number of context generations or the number of considered context population members per generation would have to be reduced to allow for large search spaces. To avoid this limitation we use (block-)independent LLM queries for batches of dimensions. More specifically, we group a set of dimensions that fits into the context of the LLM and perform multiple LLM queries per generation (Figure 3). Hence, we trade off an increased LLM inference time with scalability to a larger number of search dimensions. In the limit, each LLM call processes a single dimension d . Note that as the capabilities of LLMs to model longer-range dependencies increase, EvoLLM will likely benefit from such advances.

5 LLMS ARE ZERO-SHOT EVOLUTION STRATEGIES

5.1 Evaluation on Synthetic BBO Functions

We start by evaluating the performance of the EvoLLM prompt design on various BBOB [12] tasks with different numbers of search dimensions and population sizes. We use the $K = 5$ last generations and $M = 5$ best-seen evaluations throughout the optimization trajectory to construct the context string. We use a set of 4 warm-up generations using random search to seed the context buffer H . Afterwards, we use the LLM proposed mean update to perform BBO and use a fixed perturbation strength, $\sigma = 0.2$. The default prompt construction settings are summarized in Table 1.

To assess the performance of EvoLLM across various settings, we consider four different tasks, which consist of the separable Sphere,

Setting Name	Setting Choice
Context Generations	$K = 5$
Context Members	$M = 5$
Generation Selection	'last'
Candidate Selection	'best-across'
Generation Sorting	'improving'
Candidate Sorting	'improving'
Improvement Indicator	✓
Uniqueness Filtering	False
Improvement Querying	True
Warm-up Generations	4
Warm-up Strategy	Random Search

Table 1: EvoLLM Context Construction Settings.

moderate-condition number Rosenbrock, the high-condition number Discus, and the multi-modal Schwefel function. Figure 4 shows that the LLM-based ES can outperform random search and Gaussian Hill Climbing on these BBOB functions [12] with different search dimensions ($D \in \{3, 5\}$) and population sizes ($N \in \{5, 10\}$). On many of the considered tasks, the LLM-based Strategy is even capable of outperforming diagonal covariance ES such as SNES [38]. Furthermore, EvoLLM outperforms all baselines across all tasks for small budget settings, i.e. less than 10 generations. We consider three different classes of differently sized pre-trained LLM models including three PaLM2 models [9], OpenAI models [34], and the open-source available Llama2 [40] models. We observe (Figure 1, right) that the LLM model size inversely affects the downstream performance of EvoLLM. Larger models (PaLM-L, Llama2-70B) tend to perform

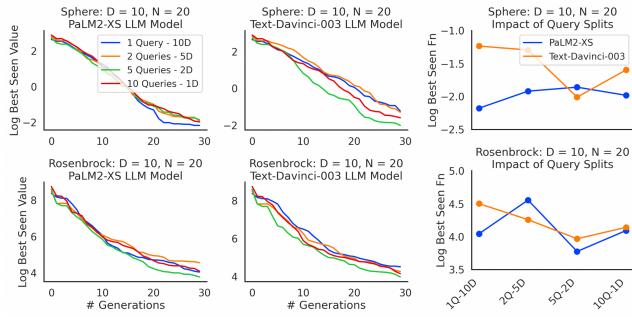


Figure 5: EvoLLM performance (lower is better) on BBOB [12] functions with multi-dimensional LLM query splits. We consider text-davinci-003 and PaLM2-XS as base LLM models and find that performance does not degrade when using splits. Top: 10-dimensional Sphere problem. Bottom: 10-dimensional Rosenbrock problem. Averaged results over 5 independent runs.

worse than smaller models (PaLM-XS, Llama2-7B), negating a scaling law trend. Interestingly, GPT-4 tends to perform similarly to the smaller models potentially pointing towards the size of the individual mixture of expert models.

Next, we scaled EvoLLMs to larger search spaces ($D = 10$) using batched LLM queries for blocks of parameters. More specifically, we considered different numbers of queries and dimension groupings, e.g. 5 times 2-dimensional versus 2 times 5-dimensional LLM queries. Thereby, the resulting optimizer has to perform blocked separable optimization and can not interpolate information from other potentially correlated groups of dimensions. Figure 5 indicates that the performance of the EvoLLM does not significantly decline as we optimize the parameters in groups. Interestingly, this observation holds for both the separable quadratic fitness function as well as the non-separable Rosenbrock function. Furthermore, it robust for two different model classes, PaLM2-XS and text-davinci-003.

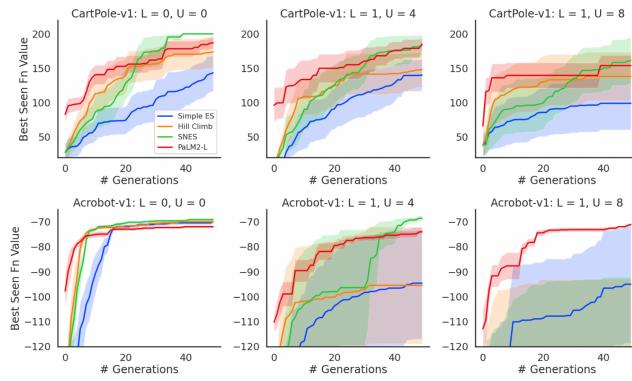


Figure 6: EvoLLM performance (higher is better) on CartPole & Acrobot [1, 19] control task with different neural network architectures. LLM-based ES can optimize small networks and even outperform baselines in the small evaluation budget regime. Averaged results over 5 independent runs.

5.2 Evaluation on Neuroevolution Tasks

The previous BBO results indicate that LLM-based ES are capable of optimizing various classic functions with different characteristics (conditioning, single/multi-modal optima structure, etc.). However recent work has shown that benchmarking on such functions can be of limited relevance for machine learning tasks [20]. Therefore, we wondered whether EvoLLMs can also be applied to neuroevolution tasks. If this is indeed the case, LLMs may be a viable future option for large-scale autonomous optimization which potentially can include various text-based information. We consider the CartPole-v1 and Acrobot-v1 discrete control tasks using a single hidden layer MLP policy. In this case, the EvoLLM has to evolve the parameters of the feedforward network used to output the agent’s action at every episode timestep. The considered networks contain between 16 and 40 weights and biases to be optimized. We again batch the parameters into groups and optimize the neural network parameters using 32 population members and 8 stochastic rollouts for fitness evaluation. Figure 6 provides evidence that EvoLLM can indeed evolve such neural network-parametrized policies for both considered tasks. More specifically, it is again capable of even outperforming competitive baselines in the small budget regime. We note that optimization becomes more challenging as the number of optimized parameters increases. This result is of importance because it provides an intriguing perspective on LLM-based agents: In principle, they can optimize neural network artifacts using a gradient-free implicit optimization procedure implemented by their internal activations.

6 EVOLLM ABLATION STUDIES

After having established that LLMs are capable of acting as improvement operators for ES, we next investigate the importance of the prompt design, discretization resolution, and context length.

6.1 Prompt Strategy Ablations

We consider the following 4 different prompt design decisions:

- (1) **Uniqueness of mean for the selected generations:** We either filter the context generations by the uniqueness of their mean in the sequence or not (Figure 7, left).
- (2) **Selection criteria for context generations/candidates:** We compare the base selection setting (Table 1) with randomly selected generations ('G-R') and members ('C-R').
- (3) **Sorting criteria for selected generations/candidates:** We compare the base sorting setting (Table 1) with randomly sorted generations ('G-R') and members ('C-R').
- (4) **Fitness information provision & improvement request:** We either add the desired fitness query or not.

Figure 7 considers two different BBOB settings (Sphere, Rosenbrock) and two different base LLM models (GPT-4, PaLM-XS). It is not beneficial to filter the selected generations by mean uniqueness. Furthermore, EvoLLM remains largely robust concerning the selection of context candidates and members. The sorting criterion, on the other hand, has strong effects on the downstream performance. Sorting generations and members randomly decreases performance substantially. Furthermore, not providing fitness information decreases GPT-4’s performance. This suggests that different LLM models can behave differently in the context of BBO.

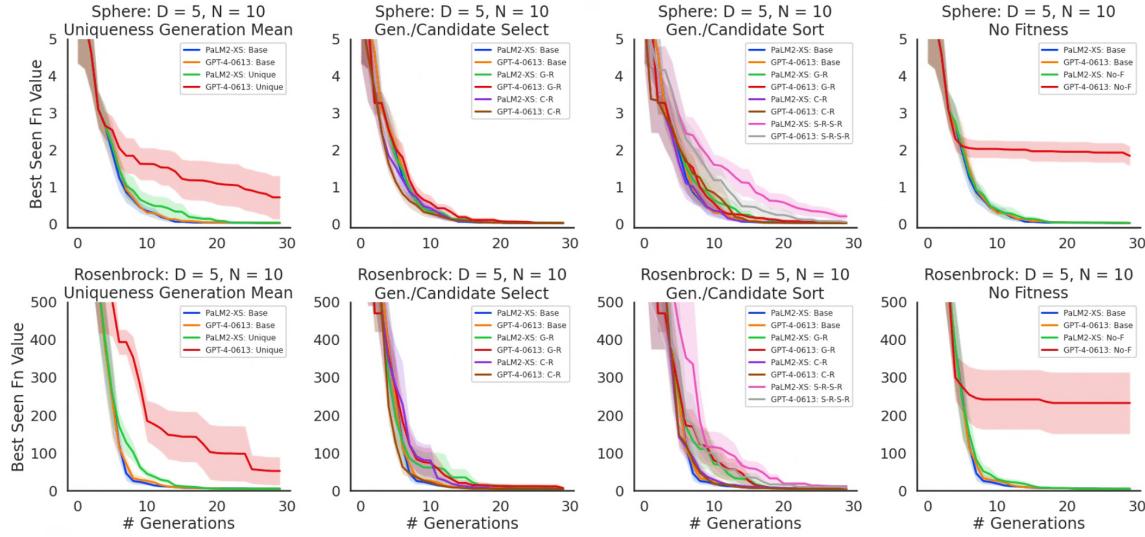


Figure 7: Different Prompt Constructors on two different BBOB problems. Left: Impact of generation uniqueness filtering. Middle: Impact of selection and sorting of generations and candidates. Right: Impact of providing fitness information and improvement query. The EvoLLM prompt is largely robust to all individual choices, but the performance drops if we do not include the fitness information or filter for improving generation sequences. Averaged results over 5 independent runs.

6.2 Raw text vs. discretized representation

One key aspect of the EvoLLM is our usage of discretized integer representations instead of raw floating point number strings. In preliminary experiments, we observed that the vocabulary tokenizer of LLMs struggles with representing high-precision numbers. Different numbers may result in different numbers of tokens (Figure 13), which in turn makes it challenging for the LLM to infer improvements. Most common tokenizers (e.g. SentencePiece) represent integer numbers up to 1000 as a single token. We therefore chose to translate raw solutions into integer representations and map the LLM output back into the corresponding floating point. To illustrate the impact of this choice, we compare the performance of our EvoLLM with the raw text-based prompting strategy outlined in Zhang et al. [46]. Using the same LLM backend model we repeatedly observe that the text-based approach is not capable of ‘zooming into’ optima and instead performs too large perturbations to progress after initial improvements.

6.3 Impact of Resolution & Context Length

Next, we considered how the chosen discretization resolution impacts the EvoLLM performance. Intuitively, a resolution that is too coarse will not allow the optimizer to discover very narrow optima basins, while a too-high resolution will hurt the LLM’s ability to infer improvements from the context. We used the template outlined in Table 1 and discretized an optimization range from -3.0 to 3.0 into $\{50, 100, 1000, 10000\}$ bins. In Figure 11 we indeed observe the hypothesized behavior for two different PaLM2 models: As we increase the resolution up to 1000, performance improves. For 10000, on the other hand, it decreases again. For the Rosenbrock problem, on the other hand, this trend is not clear. This may be due to the optimizer not finding the global optimum but preemptively

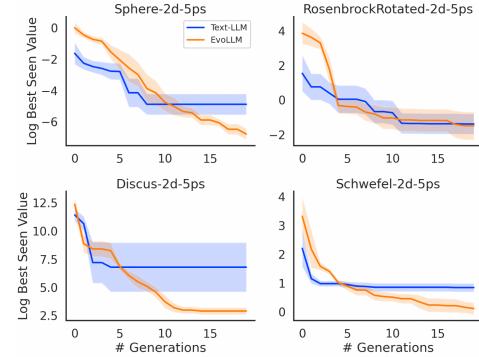


Figure 8: EvoLLM versus text-based optimization prompting performance. Our proposed discretized solution representation is capable of improvements even for longer optimization trajectories. Text-based prompting, on the other hand, quickly saturates. Averaged results over 5 independent runs.

converging in a broader local optimum basin. This indicates that the optimal resolution parameter is indeed problem-specific.

Finally, we wondered how much context information is required for the LLM to infer beneficial updates to the mean statistic. In Figure 12 we considered different amounts of context generations and members. We find that even with extremely limited information, the EvoLLM can make improving updates to the mean. Often more context generations appear to be beneficial compared to more context members. This may be due to the clear improvements in the sequence, which makes it easier for the LLM to infer continuation patterns such as momentum.

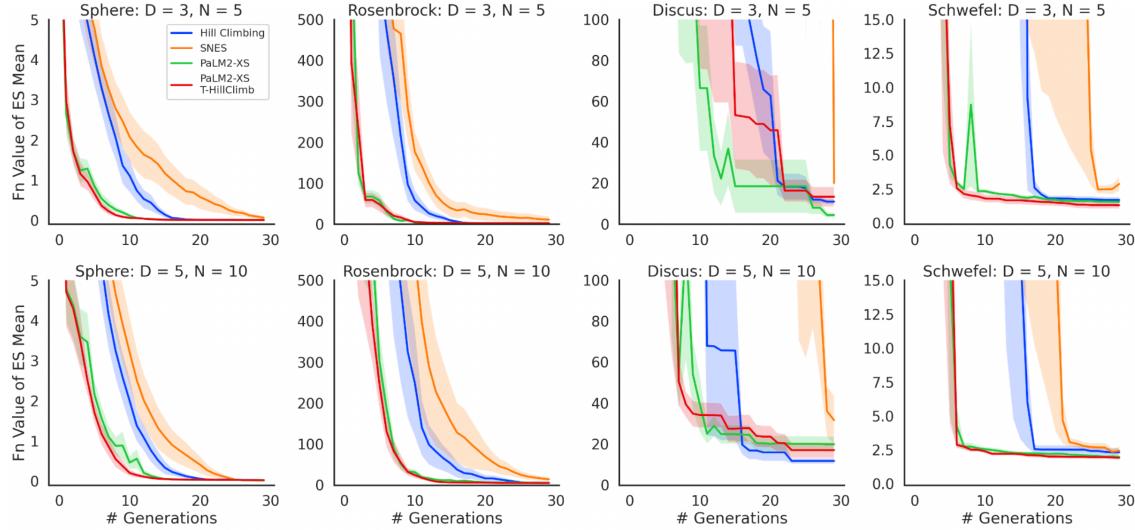


Figure 9: Instruction fine-tuning using the Hill-Climbing algorithm improves EvoLLM’s performance on unseen BBOB tasks (lower is better). Results are averaged over 5 independent runs.

7 EVOLLM WITH TEACHER FINE-TUNING

Finally, we investigate the impact of fine-tuning the LLM using BBO trajectories generated by a teacher algorithm. More specifically, we use two BBO functions (Sphere and Rosenbrock) and collect BBO rollouts using a simple Gaussian Hill Climbing algorithm. Afterwards, we discretize the solution candidates and construct sequences of EvoLLM instructions according to the prompt template outlined in Section 4. We then train the LLM ‘backend’ to predict the discretized integer representation of the teacher’s search distribution update. To that end, we used the standard next-token prediction cross-entropy loss and fine-tuned it for a short amount of training steps (Figure 14). Afterwards, we deploy the EvoLLM and evaluate its performance on the BBO tasks. Our results focus on the PaLM-XS model. Figure 9 demonstrates the effect of fine-tuning on four different tasks (Sphere and Rastrigin with 2 and 5 search dimensions). For all cases, we observe a small but robust performance increase after instruction-based fine-tuning. In almost all considered cases, the tuned LLM outperforms both the teacher algorithm and the non-tuned LLM. Figure 10 provides further evidence that this result extends also to more challenging neuroevolution tasks. In summary, this highlights the potential for text-based LLMs to potentially distill teacher algorithms. One promising future direction may want to explore tuning the tokenizer vocabulary to better accommodate the representation of floating point numbers.

8 DISCUSSION

Summary. We outline a prompt strategy that enables purely text-trained LLMs to robustly act as an ES on various BBO tasks. Furthermore, we provide several ablations highlighting the importance of careful solution representation and context construction. Finally, we successfully demonstrate that the LLMs capabilities can be improved by fine-tuning on teacher algorithm sequences.

Limitations. We expect EvoLLM’s performance to differ based

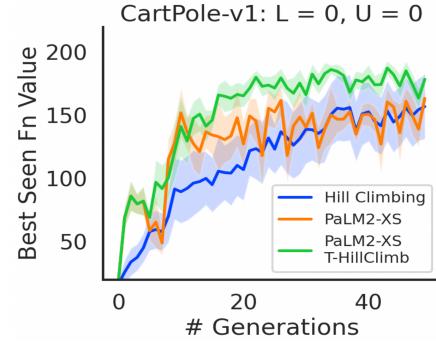


Figure 10: Instruction fine-tuning using Hill-Climbing improves EvoLLM’s performance on unseen CartPole task.

on pretraining & fine-tuning protocols. Understanding how these details affect BBO performance is a key open challenge. Going forward LLMs capable of long context reasoning may be required to scale to larger search spaces. In preliminary experiments, we extended the prompt strategy to non-isotropic ES. The LLM had to additionally output an update to the diagonal covariance. This did not yield significant improvements.

Future Work. A potential direction is the construction of an LLM-specific BBO benchmark capturing various optimization abilities. This could enable directed progress for this specific LLM capability. Furthermore, we want to explore tokenization techniques tailored for numerical representations. Finally, we expect EvoLLM to be able to harness all future advances of LLMs going forward, e.g. longer context windows.

Ethical Considerations. LLMs are powerful black-box tools that require careful monitoring of their agency. This is especially true when used for autonomous optimization purposes as the ones outlined in our work.

REFERENCES

- [1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).
- [2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [3] Angelica Chen, David M Dohan, and David R So. 2023. EvoPrompting: Language Models for Code-Level Neural Architecture Search. (2023).
- [4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems* 34 (2021), 15084–15097.
- [5] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. 2017. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*. PMLR, 748–756.
- [6] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'aurelio Ranzato, et al. 2022. Towards learning universal hyperparameter optimizers with transformers. *Advances in Neural Information Processing Systems* 35 (2022), 32053–32068.
- [7] Lucas M Dery, Abram L Friesen, Nando De Freitas, Marc'Aurelio Ranzato, and Yutian Chen. 2022. Multi-step Planning for Automated Hyperparameter Optimization with OptFormer. *arXiv preprint arXiv:2210.04971* (2022).
- [8] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Prompter: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797* (2023).
- [9] Rohan Anil Google and, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Saman Shakernia, Emanuel Tarpa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradmer, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakansha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Diaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiayou Feng, Vlad Fliebert, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielka, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Li, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeonaeck Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggiori, Aroma Mahendru, Joshua Maynez, Vedant Misra, Mayasam Moussalem, Zachary Nado, John Nham, Erie Ni, Andrew Nystrom, Alicia Parrish, Marie Pellet, Martin Polacek, Alex Polozov, Reiner Pope, Siyan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurora Roy, Brennan Saeta, Rajkumar Samuel, Renee Shleifer, Ambrose Sloane, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Didong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. 2023. PaLM 2 Technical Report. (2023). *arXiv:cs.CL/2305.10403*
- [10] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitai Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532* (2023).
- [11] Nikolas Hansen. 2006. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms* (2006), 75–102.
- [12] Nikolas Hansen, Anne Auger, Steffen Finck, and Raymond Ros. 2010. *Real-parameter black-box optimization benchmarking 2010: Experimental setup*. Ph.D. Dissertation. INRIA.
- [13] Louis Kirsch, James Harrison, Jascha Sohl-Dickstein, and Luke Metz. 2022. General-purpose in-context learning by meta-learning transformers. *arXiv preprint arXiv:2212.04458* (2022).
- [14] Jannik Kossa, Neil Band, Clare Lyle, Aidan N Gomez, Thomas Rainforth, and Yarin Gal. 2021. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *Advances in Neural Information Processing Systems* 34 (2021), 28742–28756.
- [15] Siddarth Krishnamoorthy, Satvik Mehul Mashkaria, and Aditya Grover. 2022. Generative pretraining for black-box optimization. *arXiv preprint arXiv:2206.10786* (2022).
- [16] Robert Lange, Tom Schaul, Yutian Chen, Chris Lu, Tom Zahavy, Valentin Dalibard, and Sebastian Flemerhag. 2023. Discovering Attention-Based Genetic Algorithms via Meta-Black-Box Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 929–937.
- [17] Robert Lange, Tom Schaul, Yutian Chen, Tom Zahavy, Valentin Dalibard, Chris Lu, Satinder Singh, and Sebastian Flemerhag. 2023. Discovering evolution strategies via meta-black-box optimization. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 29–30.
- [18] Robert Tjarko Lange. 2022. evosax: JAX-based Evolution Strategies. *arXiv preprint arXiv:2212.04180* (2022).
- [19] Robert Tjarko Lange. 2022. gymmax: A JAX-based Reinforcement Learning Environment Library. (2022). <http://github.com/RobertTLange/gymmax>
- [20] Robert Tjarko Lange, Yujin Tang, and Yingtao Tian. 2023. NeuroEvoBench: Benchmarking Evolutionary Optimizers for Deep Learning Applications. *arXiv preprint arXiv:2311.02394* (2023).
- [21] Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, et al. 2022. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215* (2022).
- [22] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosioke, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*. PMLR, 3744–3753.
- [23] Joe Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. 2023. Evolution through large models. In *Handbook of Evolutionary Machine Learning*. Springer, 331–366.
- [24] Brian Lester, Ramzi Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691* (2021).
- [25] Yujian Betterest Li and Kai Wu. 2023. SPELL: Semantic Prompt Evolution based on a LLM. *arXiv preprint arXiv:2310.01260* (2023).
- [26] Fei Liu, Xi Lin, Zhenkun Wang, Shunyu Yao, Xiaoliang Tong, Mingxuan Yuan, and Qingfu Zhang. 2023. Large language model for multi-objective evolutionary optimization. *arXiv preprint arXiv:2310.12541* (2023).
- [27] Fei Liu, Xiaoliang Tong, Mingxuan Yuan, and Qingfu Zhang. 2023. Algorithm Evolution Using Large Language Model. *arXiv preprint arXiv:2311.15249* (2023).
- [28] Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. 2023. Large Language Models as Evolutionary Optimizers. *arXiv preprint arXiv:2310.19046* (2023).
- [29] Chris Liu, Yannick Schroeter, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Ferial Behbahani. 2023. Structured state space models for in-context reinforcement learning. *arXiv preprint arXiv:2303.03982* (2023).
- [30] Yao Lu, Jayati Wang, Sebastian Ruedel, and Pontus Stenetorp. 2023. Prompt Optimisation with Random Sampling. *arXiv preprint arXiv:2311.09569* (2023).
- [31] Elliot Meyerson, Mark J Nelson, Herbie Bradley, Arash Moradi, Amy K Hoover, and Joel Lehman. 2023. Language Model Crossover: Variation through Few-Shot Prompting. (2023).
- [32] Suvi Mirchandani, Fei Xia, Pete Florence, Brian Ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. 2023. Large Language Models as General Pattern Machines. *arXiv preprint arXiv:2307.04721* (2023).
- [33] Allen Nie, Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. 2023. Importance of Directional Feedback for LLM-based Optimizers. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
- [34] OpenAI. 2023. GPT-4 Technical Report. *ArXiv abs/2303.08774* (2023). <https://arxiv.org/abs/2303.08774>
- [35] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.
- [36] Ingo Rechenberg. 1978. Evolutionstrategien. In *Simulationsmethoden in der Medizin und Biologie*. Springer, 83–114.
- [37] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).
- [38] Tom Schaul, Tobias Glasmachers, and Jürgen Schmidhuber. 2011. High dimensions and heavy tails for natural evolution strategies. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 845–852.
- [39] Yujin Tang and David Ha. 2021. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 22574–22587.
- [40] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashkyrov, Soumya Batra, Prajwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
- [41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- [42] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharad Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).
- [43] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.
- [44] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. 2014. Natural evolution strategies. *The Journal of Machine Learning Research* 15, 1 (2014).
- [45] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. (2023).
- [46] Michael Zhang, Nishkrish Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. 2023. Using Large Language Models for Hyperparameter Optimization. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
- [47] Denny Zhou, Nathanael Schärlí, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625* (2022).

A ADDITIONAL RESULTS

A.1 Impact of Search Space Resolution

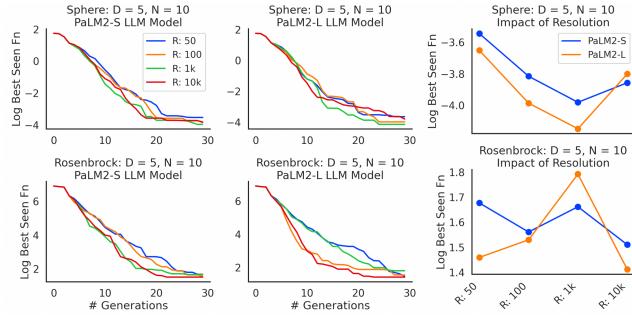


Figure 11: EvoLLM performance for different search resolutions. It struggles to zoom into optima basins for low-resolutions, while high resolutions degrade performance due to tokenization. Averaged results over 5 independent runs.

A.2 Context Members & Generations

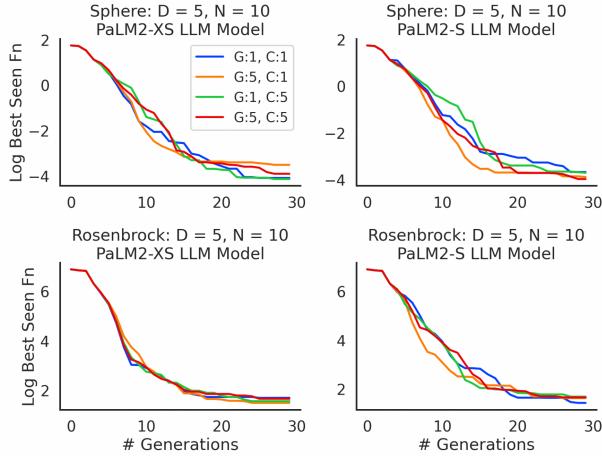


Figure 12: EvoLLM performance for different amounts of context information. Even with limited information the LLM is capable of inferring improving sequence patterns. Averaged results over 5 independent runs.

B PROMPT CONSTRUCTION EXAMPLE: 2 DIM. SPHERE / 5 POPULATION MEMBERS

Below we show an example of the EvoLLM prompt outlined in Table 1 for a 2-dimensional Sphere task with 5 population members. We start with a randomly initialized mean and 4 random search warm-up generations:

$$\mu = [2.93851025 - 1.77656265], f^* = 0.4378929557544995$$

===== Generation 0 =====

0.44 : 397539; 14792, 0, 0302, 0, 186346, 0, 419685, 0, 397539, 0
0.39 $\Rightarrow \mu = [-0.619395150.2329004]$, $f^* = 0.438$

===== Generation 1 =====

1.29 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
0.44 : 397539; 14792, 0, 0302, 0, 186346, 0, 419685, 0, 397539, 0
0.29 $\Rightarrow \mu = [-0.619395150.2329004]$, $f^* = 0.438$

===== Generation 2 =====

6.03 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
1.29 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
0.44 : 397539; 14792, 0, 0302, 0, 186346, 0, 419685, 0, 397539, 0
0.29 $\Rightarrow \mu = [-0.619395150.2329004]$, $f^* = 0.438$

===== Generation 3 =====

6.03 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
2.48 : 397539; 748280, 0, 316687, 0, 419685, 0, 670417, 0, 397539, 0
1.29 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
0.44 : 397539; 14792, 0, 0302, 0, 186346, 0, 419685, 0, 397539, 0
0.34 $\Rightarrow \mu = [-0.6180.234]$, $f^* = 0.339$

===== Generation 4 =====

6.03 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
2.48 : 397539; 748280, 0, 316687, 0, 419685, 0, 670417, 0, 397539, 0
1.29 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
0.44 : 397539; 14792, 0, 0302, 0, 186346, 0, 419685, 0, 397539, 0
0.34 : 413543; 388557, 0, 448604, 0, 397539, 0, 399504, 1, 413543, 1
0.25 $\Rightarrow \mu = [-0.5220.258]$, $f^* = 0.249$

===== Generation 5 =====

6.03 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
2.48 : 397539; 748280, 0, 316687, 0, 419685, 0, 670417, 0, 397539, 0
1.29 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
0.34 : 413543; 388557, 0, 448604, 0, 397539, 0, 399504, 1, 413543, 1
0.25 : 423531; 417564, 0, 399504, 0, 413543, 0, 415522, 1, 423531, 1
0.20 $\Rightarrow \mu = [-0.4620.186]$, $f^* = 0.15$

===== Generation 6 =====

```
6.03 : 397539; 559140, 0, 186346, 0, 419685, 0, 670417, 0, 397539, 0
2.48 : 397539; 748280, 0, 316687, 0, 419685, 0, 670417, 0, 397539, 0
0.34 : 413543; 388557, 0, 448604, 0, 397539, 0, 399504, 1, 413543, 1
0.25 : 423531; 417564, 0, 399504, 0, 413543, 0, 415522, 1, 423531, 1
0.15 : 436506; 413543, 0, 415522, 0, 423531, 0, 487564, 1, 436506, 1
0.09 :=>  $\mu = [-0.3840.036]$ ,  $f^* = 0.101$ 
```

===== Generation 7 =====

```
2.48 : 397539; 748280, 0, 316687, 0, 419685, 0, 670417, 0, 397539, 0
0.34 : 413543; 388557, 0, 448604, 0, 397539, 0, 399504, 1, 413543, 1
0.25 : 423531; 417564, 0, 399504, 0, 413543, 0, 415522, 1, 423531, 1
0.15 : 436506; 413543, 0, 415522, 0, 423531, 0, 487564, 1, 436506, 1
0.10 : 447499; 434527, 0, 487564, 0, 436506, 0, 439514, 1, 447499, 1
0.07 :=>  $\mu = [-0.318 - 0.006]$ ,  $f^* = 0.040$ 
```

===== Generation 8 =====

```
0.34 : 413543; 388557, 0, 448604, 0, 397539, 0, 399504, 1, 413543, 1
0.25 : 423531; 417564, 0, 399504, 0, 413543, 0, 415522, 1, 423531, 1
0.15 : 436506; 413543, 0, 415522, 0, 423531, 0, 487564, 1, 436506, 1
0.10 : 447499; 434527, 0, 487564, 0, 436506, 0, 439514, 1, 447499, 1
0.04 : 472481; 436506, 0, 439514, 0, 442500, 0, 447499, 0, 472481, 1
0.03 :=>  $\mu = [-0.168 - 0.114]$ ,  $f^* = 0.018$ 
```

===== Generation 9 =====

```
0.25 : 423531; 417564, 0, 399504, 0, 413543, 0, 415522, 1, 423531, 1
0.15 : 436506; 413543, 0, 415522, 0, 423531, 0, 487564, 1, 436506, 1
0.10 : 447499; 434527, 0, 487564, 0, 436506, 0, 439514, 1, 447499, 1
0.04 : 472481; 436506, 0, 439514, 0, 442500, 0, 447499, 0, 472481, 1
0.02 : 478495; 465474, 0, 472481, 0, 476519, 1, 479485, 1, 478495, 1
0.01 :=>  $\mu = [-0.132 - 0.03]$ ,  $f^* = 0.018$ 
```

===== Generation 10 =====

```
0.15 : 436506; 413543, 0, 415522, 0, 423531, 0, 487564, 1, 436506, 1
0.10 : 447499; 434527, 0, 487564, 0, 436506, 0, 439514, 1, 447499, 1
0.05 : 478495; 483467, 0, 472481, 0, 476519, 0, 479485, 0, 478495, 0
0.04 : 472481; 436506, 0, 439514, 0, 442500, 0, 447499, 0, 472481, 1
0.02 : 478495; 465474, 0, 472481, 0, 476519, 1, 479485, 1, 478495, 1
0.01 :=>  $\mu = [-0.132 - 0.03]$ ,  $f^* = 0.018$ 
```

C FLOATING POINT NUMBER TOKENIZATION

Below we show that the number of tokens used for a floating point number linearly increases with the number of digits. The numbers are sampled randomly:

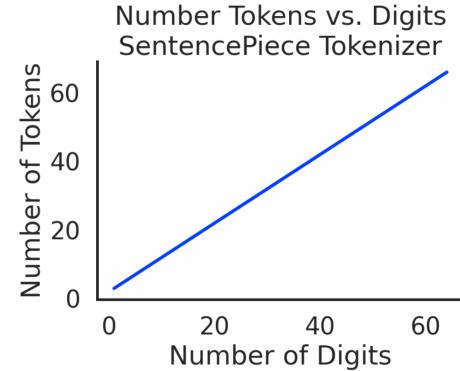


Figure 13: SentencePiece Tokenizer floating point number digits versus resulting tokens used for the language model.

D HILL CLIMBING INSTRUCTION FINE-TUNING

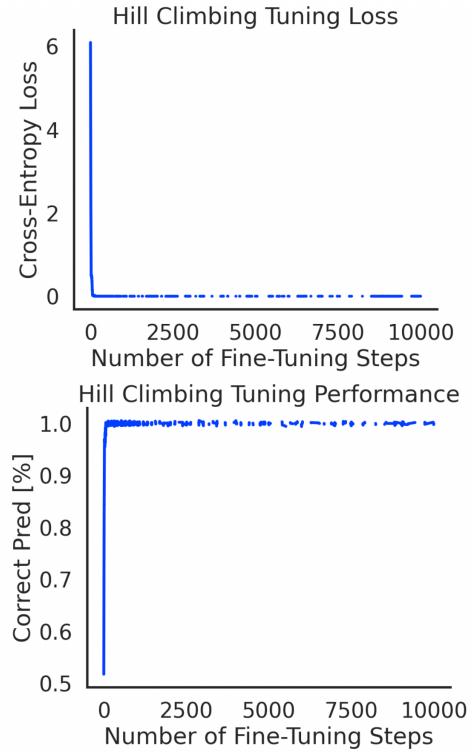


Figure 14: Fine-tuning loss and accuracy for instruction fine-tuning using Hill Climbing teacher algorithm trajectories on PaLM-XS.