AttackEvolve: Using In-Context Learning Enhanced Searches to Improve the Search Efficiency of LM-Based Search Algorithms

Anonymous Author(s)
Affiliation
Address
email

Abstract

This work in progress paper investigates how LM-based evolutionary algorithms, such as AlphaEvolve, can be used to generate attacks on autonomous vehicle systems. While these algorithms have previously been used to generate prompt injection attacks, our problem contains an inherent domain misalignment that we must tackle in order to get language models, which searches over text, to search for joint adversarial inputs across different domains outside of text. Therefore, starting outside of our adversarial problem first, we develop a search framework which uses in-context learned function approximations to guide evolution and improve overall search efficiency. We present preliminary results using toy problems to show how our framework finds global optimal solutions more efficiently than its predecessor. These results suggest that using in-context learned function approximations may be the viable path for generating adversarial data with LM-based search algorithms.

1 Introduction

Our research seeks to develop algorithms that allow developers to generate sequential adversarial attacks on their closed loop autonomous systems that can induce particular adversarial failures when applied over a time horizon. (see Figure 1). Inspired by previous work ([8]), we investigate how existing evolutionary algorithms, such as AlphaEvolve ([9]), can be used to generate these attacks. However, although researchers have used these algorithms to find new solutions to attack problems, the challenge in applying AlphaEvolve comes from a domain misalignment. While frameworks like AlphaEvolve evolve code programs, applying this to generate joint image-trajectory attacks introduces severe search redundancies – where distinct code mutations yield identical adversarial inputs – and search limitations – in which inter-program semantics fail to inform correct adversarial mutations for the LM to apply. To resolve this, we propose AttackEvolve, which directly evolves text-serialized sample arrays in lieu of programs, establishing a bijective (one-to-one) mapping between text representations and physical inputs that maximizes search efficiency. By additionally leveraging in-context learned function approximations of the target system, AttackEvolve doesn't just rely on language semantics but also encourages the model to learn the underlying functional representation of its target to guide evolution. We evaluate our framework outside our adversarial problem first and present the result of initial toy experiments. When applied on toy functions, our framework is able to find global optimums to highly non-convex functions more quickly and efficiently than our baseline solution. Throughout this paper, we use OpenEvolve instead of AlphaEvolve as our baseline, since OpenEvolve is open-sourced. The main contributions of this paper are an improved search framework designed to facilitate the discovery of adversarial attacks on autonomous vehicles and the design of toy experiments for effective evaluation of LM-based search algorithms.

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

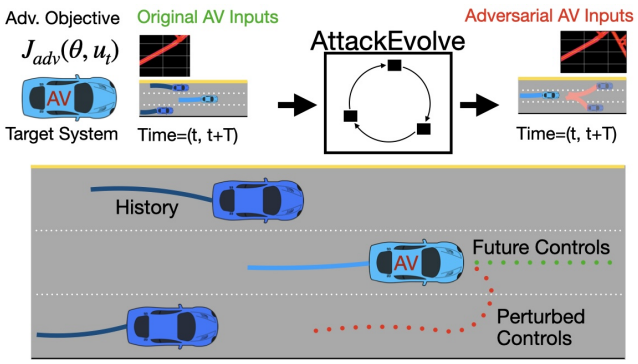


Figure 1: The motivating problem is to generate joint image and trajectory attacks for target trajectory prediction systems in autonomous driving. These attacks will be applied to the vehicle sequentially over a time horizon to cause a collision or fulfill some other adversarial objective. With this framework, system developers will be able to generate large adversarial datasets that they can use for building filters, detection systems, or doing adversarial training.

2 Related Works

Previous works can be broken down into two categories - one that finds failure modes on closed-loop systems and another that finds myopic attacks on isolated vehicle models or components. The latter category touches the classically known adversarial literature, in which solutions either use pure gradient-based approaches ([7]), constrained (or augmented) gradient-based approaches ([5]), gradient-estimations ([10]), trajectory perturbations ([1], [2], [12]), or search algorithms ([8]). For the former, currently known solutions involve using falsification and simulations to generate "failure modes" ([6]), reachability analysis and simulations to find failures ([3]), and formal analysis to similar find counter examples and failures ([4]). For both categories, each solution itself is not well-suited to solve our problem. Solutions in the first category do not generate non-myopic attacks nor jointly perturb images-trajectory sequences. For the second category, although these solutions may be able to find failure cases for closed-loop systems in sequence, many of these solutions are limited by the use of simulators, which may not capture long-tail inputs or naturally crafted adversarial examples seen in the wild. However, we may be able to leverage search algorithms to generate non-myopic multi-modal adversarial attacks. AlphaEvolve/Funsearch ([11]) is one such algorithm that use language models to evolve programs that produce solutions to complex mathematical problems. Drawing from previous successes in rediscovering or outperforming state-of-the-art solutions across complex mathematical and adversarial domains, we may be able to leverage the LM's unique semantic reasoning capabilities to discover creative, new attacks that traditional heuristics fail to uncover. In this paper, we discuss the augmentation needed to apply such a solution to our problem domain.

3 Improving evolutionary algorithms with In-Context Learning Enhanced Searches

Language models (LM) have demonstrated the ability to learn simple functions in context; therefore, we can leverage this ability the model to approximate the underlying dynamics of a given target model and then use this approximation to suggest better candidate solutions. Therefore, instead of simply "best-shot prompting" the LM for improvements, we provide critical samples of our target function that informs the model of the function's geometric properties and then ask the LM to simultaneously solve for the inverse solution to a target function score and produce new candidate input solutions that will aid it to better approximate the underlying function (see Figure 2 for details). Instead of constructing our prompts with only the best programs stored, we construct each prompt using a series of samples where each sample is selected from certain categories of points that are important for optimization and that describe the characteristics of a function:

- Extrema points: The points that achieve the maximum and minimum of the collected scores.

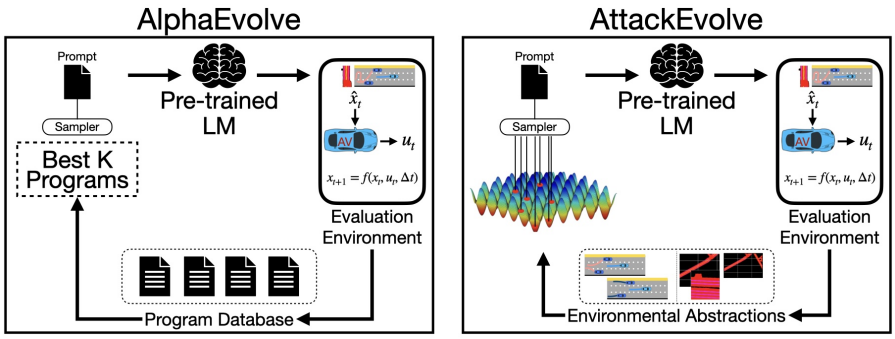


Figure 2: We built on top of the innovations made from AlphaEvolve. In AttackEvolve, instead of evolving programs, we evolve samples of a function that represents the a target system, feed these samples into the prompt, and prompt the model for a new input that either achieves the target score or aids it in approximating the underlying function. Once, new inputs are produced, we evaluate them as usual and store the inputs themselves inside the database. Our actual implementation builds on top of OpenEvolve, which is an open-source implementation of the AlphaEvolve framework.

- Critical points: Points in the domain where the derivative is zero or undefined. The points where the function's derivative is zero identifies local maxima, minima, and saddle points and the points where the function is undefined represents holes or strange behaviors in the underlying target system. To select candidate local minima/maxima without calculating gradients, we simply select a sample of the best and worst solutions. Similarly, singularity points are selected based on whether the generated solution produced an error.
- Endpoints: Samples which contain the inputs at the boundaries of the domain in which the function is defined. For this, we simply select samples with the smallest and largest value within that island.
- Random points: We randomly select other candidate solutions in the database that describes the function's behavior in other random regions.

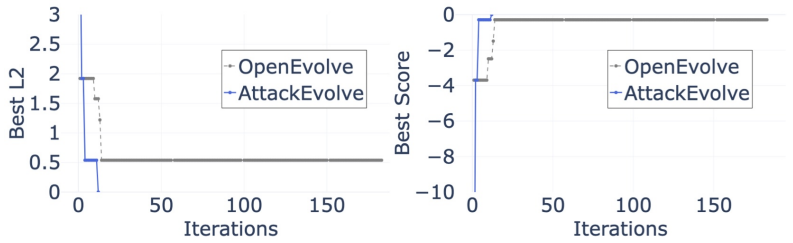
We follow the same island-based model as ([9]) where the population is split into m groups, each group is evolved separately using a copy of the user provided initial program, and programs within islands are furthered clustered into group according to a signature. However, because we evolve candidate attacks directly, instead of creating signatures based on program features, we define signatures as a tuple of dimension $n+1$, $t \in \mathbf{R}^{n+1}$, where each element is the input's discretized value or its associated score. Each tuple's component is defined as follows:

$$t_i = \begin{cases} \left\lfloor \frac{x_i - a}{b - a} \right\rfloor * K & \text{if } i \leq n \\ \left\lfloor \frac{c - a_{score}}{b_{score} - a_{score}} \right\rfloor * K & \text{if } i \geq n \end{cases}$$

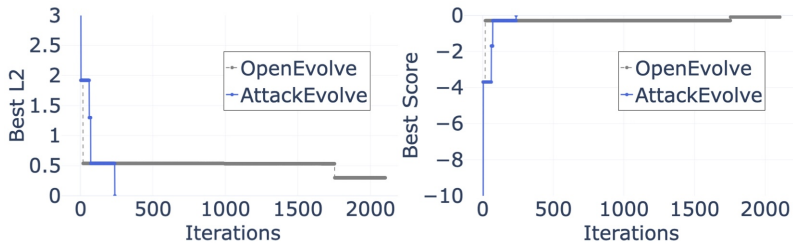
where K is the total number of bins, a is the lower bound for the input domain considered, b is the upper bound for the input domain considered, x corresponds to the candidate solution generated, c is the output score associated with x, and a_{score} and b_{score} are the minimum and maximum values considered for the function's range.

4 Toy Experiment: Finding Minimizers to Highly Non-Convex Functions

We design toy experiments to test how well our framework can search for global optimizers for 2 functions (see Appendix A). The goal is to search for the point that represents the global minimum of the function. Each evolved solution is a 2-dimensional vector represented in text form and each sample collected is simply (x, c). The objective function for this experiment is: $c = -f(x - p)$, where f is the function being minimized and p is a design parameter. (Note: Most of the global minimizers for our function are located at X = 0; thus, to make a non-trivial minimizer, which will necessitate search, we offset the input by p to relocate the minimizer to some non-trivial location). All initial seeds are randomized and the search algorithm is ran until the global minimizer is found or until it timeouts after 48hrs. Both algorithms uses the Deepseek-r1 model with 8 billion parameters



(a) Sphere Function performance.



(b) Ackley Function performance.

Figure 3: Comparison of search results across different test functions. The left plot of each subfigure shows the current closest point found to the global optimum per iteration and the right shows the current best minimizer found per iteration. In both experiments, we see that AttackEvolve finds the global solution very quickly in comparison to OpenEvolve. Additionally, unlike OpenEvolve, it doesn't get stuck over long periods of time.

as their mutators. The results in Table 1 shows the results for 2 experiments. The first experiment finds the minimizers of the Spherical Function (6 seeds) and the second finds the minimizer of the Ackley Function (8 seeds). Average (Avg.) iteration to Solution and per run is noted for seeds in which the algorithms finds the global minimizer. Average programs discovered counts the number of distinct points generated either inside the database (AttackEvolve) or by programs inside the database (OpenEvolve). The improvement rate is measured by taking the proportion of parent-child evolutions that results in either a higher score or a closer generated point to the global optimum (denoted with "score" and "dist" in the table respectively). Even though the average improvement rate seems to be on par with OpenEvolve for the Ackley experiment, AttackEvolve finds the global minimizer more often, and it also searches more quickly (avg time per iteration). These results show the need for a framework that leverages geometrics properties of the function being optimized and the potential such a framework has to generate successful attack on learning-based robotic systems.

Table 1: Performance metrics for the Sphere and Ackley experimental setups.

	Experiment 1 (Sphere)		**Experiment 2 (Ackley)**	
Metric	OpenEvolve	AttackEvolve	OpenEvolve	AttackEvolve
SR	0.0%	**75%**	0.0%	**88%**
Avg. Iteration to Solution	N/A	**90.33**	N/A	**498.93**
Avg. Iteration Per Run	N/A	**75.25**	N/A	**616.38**
Avg. Time Per Iteration (s)	62.51	**16.42**	49.18	**17.90**
Avg. Improvement Rate (Dist)	0.18	**0.44**	**0.34**	0.30
Avg. Improvement Rate (Score)	0.19	**0.51**	0.28	**0.29**

5 Discussions and Future Work

This paper presents a work-in-progress framework for improving the search capabilities of AlphaEvolve. Although, we demonstrate the improved search efficiencies of the new framework, more experimental runs are needed for more in-depth analysis. Next steps include applying this framework to generate image perturbations on toy attack experiments.

References

[1] Yulong Cao, Chaowei Xiao, Anima Anandkumar, Danfei Xu, and Marco Pavone. Advdo: Realistic adversarial attacks for trajectory prediction. In *European Conference on Computer Vision*, pages 36–52. Springer, 2022.

[2] Yulong Cao, Danfei Xu, Xinshuo Weng, Zhuoqing Mao, Anima Anandkumar, Chaowei Xiao, and Marco Pavone. Robust trajectory prediction against adversarial attacks. In *Conference on robot learning*, pages 128–137. PMLR, 2023.

[3] Kaustav Chakraborty and Somil Bansal. Discovering closed-loop failures of vision-based controllers via reachability analysis. *IEEE Robotics and Automation Letters*, 8(5):2692–2699, 2023.

[4] Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A Seshia. Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *International Conference on Computer Aided Verification*, pages 432–442. Springer, 2019.

[5] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018.

[6] Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pages 63–78, 2019.

[7] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[8] Milad Nasr, Nicholas Carlini, Chawin Sitawarin, Sander V Schulhoff, Jamie Hayes, Michael Ilie, Juliette Pluto, Shuang Song, Harsh Chaudhari, Ilia Shumailov, et al. The attacker moves second: Stronger adaptive attacks bypass defenses against llm jailbreaks and prompt injections. *arXiv preprint arXiv:2510.09023*, 2025.

[9] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[10] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[11] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[12] Qingzhao Zhang, Shengtuo Hu, Jiachen Sun, Qi Alfred Chen, and Z Morley Mao. On adversarial robustness of trajectory prediction for autonomous vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15159–15168, 2022.

A Functions Used In Toy Experiments

A.1 Sphere Function

This function is described by the following:

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{1}$$

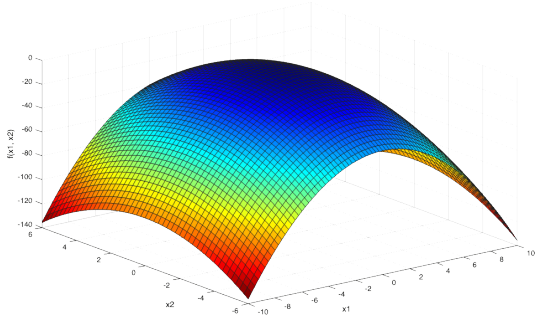


Figure 4: Sphere function.

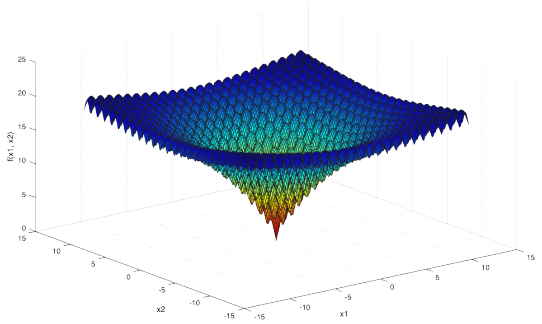


Figure 5: Ackley Function

where $\mathbf{x} \in \mathbf{R}^d$. The global minimum has value 0 at $\mathbf{x} = \mathbf{0}$.

A.2 Ackley Function

This function is described by the following:

$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1) \tag{2}$$

where $\mathbf{x} \in \mathbf{R}^d$, and $a = 20$, $b = 0.2$, and $c = 2\pi$. The global minimum has value 0 at $\mathbf{x} = \mathbf{0}$.