

An LLM-Empowered Adaptive Evolutionary Algorithm For Multi-Component Deep Learning Systems

Anonymous submission

Abstract

Multi-objective evolutionary algorithms (MOEAs) are widely used for searching optimal solutions in complex multi-component applications. Traditional MOEAs for multi-component deep learning (MCDL) systems face challenges in enhancing the search efficiency while maintaining the diversity. To combat these, this paper proposes μ MOEA, the first LLM-empowered adaptive evolutionary search methodology to detect safety violations in MCDL systems. Inspired by the context-understanding ability of Large Language Models (LLMs), μ MOEA promotes the LLM to comprehend the optimization problem and generate an initial population tailed to evolutionary objectives. Subsequently, it employs adaptive selection and variation techniques to iteratively produce offspring, balancing the evolutionary efficiency and diversity. During the evolutionary process, to navigate away from the local optima, μ MOEA integrates the evolutionary experience back into the LLM. This utilization harnesses the LLM's quantitative reasoning prowess to generate differential seeds, breaking away from current optimal solutions. We evaluate the effectiveness of μ MOEA in finding safety violations of MCDL systems, and compare its performance with state-of-the-art MOEA methods. Experimental results show that μ MOEA can significantly improve the efficiency and diversity of the multi-objective evolutionary search.

Introduction

Multi-component deep learning systems (MCDL systems) are intricate and characterized by significant uncertainty due to their complexity. These systems often involve multiple interacting modules, each with its own set of parameters and behaviors, leading to unpredictable emergent properties. In real-world scenarios, MCDL systems are increasingly being deployed in domains with substantial societal impact, such as autonomous vehicles, healthcare, and financial services. For instance, autonomous driving systems integrate various components (e.g., object detection, path planning, and decision-making), where even minor faults in one component can lead to catastrophic outcomes (Bojarski et al. 2016). Therefore, it is crucial to detect as many safety violations as possible in these systems to mitigate risks and ensure reliability (Goodfellow et al. 2017).

Multi-objective evolutionary algorithms (MOEAs) are widely applied to search for elite solutions to find safety violations in MCDL systems (Tian et al. 2022; Abdukhamidov

et al. 2023b,a). They can be formulated as multi-objective optimization problems (Zhou et al. 2011). In practical evolutionary search solutions based on genetic algorithms, these multiple objectives correspond to various perspectives (e.g., maximize the quality of the solution, improve the diversity of solutions, balance the cost of solutions against benefits) (Ehrgott, Ide, and Schöbel 2014; Long 2014). However, there are trade-offs between these objectives as they conflict with each other in many real-world scenarios.

The evolutionary search process generally consists of three steps: 1) initializing the initial population, 2) evaluating each generated individual with a defined fitness function, and 3) selecting high-fitness individuals to conduct variation operators to generate offspring iteratively. However, existing MOEAs for detecting safety violations in MCDL systems (Tian et al. 2022) face two challenges that have not been well addressed

- **Challenge-1:** The initialization of the population highly affects the search efficiency for elitist solutions. However, in many MOEAs, the initial population is created by random initialization of parameters in the entire search space, which is highly contingent and uncertain.
- **Challenge-2:** The evolutionary search process is prone to get stuck at local optima. In existing MOEAs, the individuals of each generation are generated by the high-fitness individuals retained from previous generations, which tend to cause convergence prematurely and result in a large number of iterations that only find few similar safety violations of the MCDL systems.

The goal of this paper is to overcome the above challenges and improve the evolutionary search efficiency and diversity. Large Language Models (LLMs), such as GPT-4 from OpenAI (Roumeliotis and Tselikas 2023)), have demonstrated remarkable abilities in language understanding and quantitative reasoning (Romera-Paredes et al. 2024; Zhang et al. 2022). So we propose to leverage these capabilities to generate the initial population and navigate the evolutionary search process away from the local optima. However, despite the LLMs' expertise in interacting with humans, it is infeasible to directly apply them to search for optimal and diverse solutions for detecting safety violations of MCDL systems. This is because MCDL systems normally have high dimensionality and complexity, making it difficult for LLMs

to fully and accurately understand the search space. Additionally, since LLMs may not have enough specific domain knowledge about the MCDL systems, they will arbitrarily modify existing solutions and make up unreasonable solutions, rendering them less effective.

We design μMOEA , the first LLM-empowered adaptive evolutionary methodology. μMOEA uses LLM’s ability in language understanding to better comprehend the search task, which creates the individuals of the initial population considering the objectives instead of random initialization, **thus addressing C1**. Based on the initial population, to balance the search efficiency and diversity, μMOEA introduces adaptive selection and a suite of adaptive variations, which can dynamically adjust the mutation and crossover probabilities based on the feedback from the search process and the scores of chromosomes of individuals on different objectives. During the search process, when it gets stuck, μMOEA feeds back the evolutionary experience into the LLM, harnessing its quantitative reasoning ability to generate differential seeds to break out of local optimal solutions, **thus addressing C2**. We evaluate the effectiveness of μMOEA in the task of searching for solutions to detect safety violations in MCDL systems (represented by the industrial autonomous driving system), and compare it with the state-of-the-art method based on multi-objective genetic algorithm (NSGA-II). Experimental results show that μMOEA can find more diverse elitist solutions more efficiently.

Background

Multi-Component Deep Learning Systems

MCDL systems are characterized by their intricate internal logic, extensive interactions, and high coupling among various components (Amodei et al. 2016; Varshney 2016). Their complexity and opaqueness are further exacerbated by the substantial uncertainty, high degree of interdependence, and unpredictable nature of interactions across different deep learning models within the systems. Thus an MCDL system is often referred to as a “black box” (Hassija et al. 2024), making it challenging to thoroughly detect potential safety issues under varying conditions. It is necessary and urgent to have effective methods for the examination and detection of the internal problems in MCDL systems, without requiring a detailed understanding of the intricate workings.

A proven strategy to detect safety violations in MCDL systems is to generate solutions that simulate the diverse operational conditions and assess the system’s behaviors (including responses, decisions, operations/actions) to validate whether it adheres to the safety specifications (Borg et al. 2018). Ensuring the safety and reliability of MCDL systems requires diverse solutions to detect various potential vulnerabilities and failures of MCDL systems in a wide range of conditions (Asharf et al. 2020). However, given the large state space of MCDL systems, traditional methods struggle to cover more possible cases efficiently. Thus, there is a growing need for more adaptive and comprehensive approaches to safety assessment of MCDL systems.

MOEAs For MCDL Systems

MOEAs, represented by NSGA-II (Deb et al. 2002), are widely used in detecting safety violations in MCDL systems. They are capable of exploring the vast and complex space of system behaviors and identify any misbehaviors (Mishra et al. 2019). This is achieved with evolving and optimizing solutions to cover possible situations (Wirsansky 2020) where the system behaviors violate the safety specifications.

The overall process of an MOEA (e.g., NSGA-II) is given as follows.

- **Initial population:** NSGA-II commences by initializing a population of N solutions, which is randomly generated within the solution space.
- **Fitness function:** given a solution space S and objective functions f_1, f_2, \dots, f_m , the multi-objective optimization can be formulated as:

$$\max_{x \in S} \{f_1(x), f_2(x), \dots, f_m(x)\}.$$

To detect the safety violations in MCDL systems, the multiple objectives commonly include maximizing the fault detection and solution diversity.

- **Ranking-based selection:** in the i -th generation, the solutions are evaluated by the fitness function and sorted by the crowding distance. P_i consists of the non-dominated solutions obtained by each Pareto frontier. NSGA-II selects k solutions from P_i .
- **Variation:** NSGA-II calculates the crossover probability and mutation probability for each selected solution, and compares the probability with the threshold of variation to determine whether to conduct multi-point crossover or value mutation on it. The probability of variation is calculated by a random value in $(0, 1)$ and the threshold is pre-defined by a fixed value in $(0, 1)$.
- **Iterative generations:** after generating N offspring solutions, the next generation’s population is determined by selecting the best N solutions from the current population P_i and the offspring population P_{i+1} . NSGA-II iteratively searches and refines candidate solutions based on their performance against these defined objectives.

Methodology

We introduce μMOEA , a novel LLM-empowered adaptive evolutionary search algorithm for MCDL systems. The detailed process of μMOEA is illustrated in Algorithm 1 and Figure 1. It consists of three steps: instructing the LLM to create the initial population (line 2), evolving the population adaptively to search for optimal solutions (line 3-12), guiding the LLM to generate differential seeds based on the feedback of the evolutionary process (line 13-14). Below we give detailed explanation of each step.

Instructing LLM to Create Initial Population

When starting an evolutionary search, instead of randomly initializing N solutions as the initial population, μMOEA makes the LLM understand the evolutionary task and create the initial population by considering the objectives of the

Algorithm 1: LLM-empowered adaptive evolutionary search

Ensure: The solution set SCR
Require: The form of solution AE, starting prompt pt

- 1: $P \leftarrow \emptyset$
- 2: $P \leftarrow P \cup \text{LLM_generate}(\text{AE}, pt)$
- 3: **while** not TerminationCondition() **do**
- 4: $\text{TS}, \text{SC}, \text{MS} \leftarrow \emptyset$
- 5: **for** $p_i \in P[-1]$ **do**
- 6: execute p_i
- 7: **if** \exists ego safety violation in p_i **then**
- 8: $\text{SCR} \leftarrow \text{SCR} \cup p_i$
- 9: calculate fitness S
- 10: $\text{TS} \leftarrow \text{TS} \cup S$
- 11: $\text{SC}, \text{MS} \leftarrow \text{ADAPTIVE_SELECTION}(P, S)$
- 12: $P \leftarrow P \cup \text{ADAPTIVE_VARIATION}(\text{SC}, \text{MS})$
- 13: prompt = $pt + \text{generate_feedback}(P, S)$
- 14: $P \leftarrow P \cup \text{LLM_generate}(\text{prompt})$
- 15: **return** SCR
- 16: **procedure** ADAPTIVE SELECTION(P, S)
- 17: $\text{SC}, \text{MS} \leftarrow \emptyset$
- 18: $\text{CR}_r, \text{MR}_r = \text{calculate_variation_rate}(S)$
- 19: **for** $p_i, p_j \in P$ **do**
- 20: select fitness $s_i, s_j \in S$
- 21: $c_{i,j} = \text{generate_crossover_probability}(s_i, s_j, S)$
- 22: **if** $c_{i,j} > \text{CR}_r$ **then**
- 23: $\text{SC} \leftarrow \text{SC} \cup (p_i, p_j)$
- 24: **for** $p_i \in P$ **do**
- 25: $m_i = \text{generate_mutation_probability}(s_i, S)$
- 26: **if** $m_i > \text{MR}_r$ **then**
- 27: $\text{MS} \leftarrow \text{MS} \cup p_i$
- 28: **return** SC, MR
- 29: **procedure** ADAPTIVE VARIATION(SC, MS)
- 30: **for** $x_i \in \text{SC}$ **do**
- 31: $p_i, p'_j \leftarrow \text{adaptive_crossover}(x_i)$
- 32: $\text{PN} \leftarrow \text{PN} \cup p'_i, p'_j$
- 33: **for** $p_i \in \text{MS}$ **do**
- 34: $p'_i \leftarrow \text{adaptive_mutation}(p_i)$
- 35: $\text{PN} \leftarrow \text{PN} \cup p'_i$
- 36: **return** PN

search. This is achieved with the linguistic prompt. An example of the prompt pattern is shown in Table 1.

μMOEA sends the starting prompt along with the task understanding prompt into the LLM, to obtain the initial population. Taking the autonomous driving system as an example, a template of “introduction of the elitism of solution” is given as follows: the trajectories of NPC vehicles and pedestrians are required to disturb the ego vehicle’s driving path. They need to be different from each other, and the waypoints of them need to involve different lanes.

Evolving Population Adaptively for Optimal Solutions

Based on the initial population, μMOEA adaptively evolves them to search for diverse optimal solutions to detect safety violations in the MCDL system. Each solution is encoded as an individual $P_i = \{C_1, C_2, \dots, C_n\}$, where C_n represents the n -th chromosome consisting of a series of genes (a chromosome commonly corresponds to an element or object in

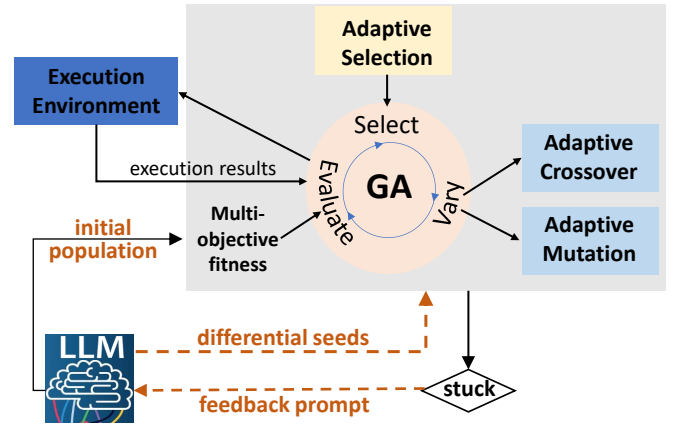


Figure 1: Overall workflow of μMOEA

Table 1: Prompt patterns for LLM-based initial population

Prompt type	Sample of linguistic patterns
Starting Prompt	You are an expert of <target MCDL system> We want you to generate <N>solutions for the system
Task	<The form of the solution>
Understanding Prompt	<Introduction of parameters of solution> <Introduction of the elitism of solution>

the solution, and a gene corresponds to an action or operation). Different from existing MOEAs where the parent individuals and their chromosomes undergo the same level of mutation and crossover, μMOEA adopts *adaptive selection* and *adaptive variations*, to improve the heritability of the elite features and search efficiency.

Specifically, for each generation, μMOEA builds improved Pareto-optimal solutions considering multiple objectives to measure the potential of solutions to expose safety violations of the MCDL system. Then it performs *adaptive selection*, which selects high-fitness solutions based on the fitness evaluated by the multiple objectives as parents for variation to generate offspring, iteratively producing Pareto-optimal solutions. To determine the solutions for crossover and mutation, μMOEA varies the probabilities of crossover and mutation adaptively in response to the fitness values of the current population, which promotes high-fitness solutions having larger crossover probabilities and low-fitness solutions having larger mutation probabilities. Then μMOEA performs *adaptive variation*: which includes adaptive crossover and adaptive mutation. It dynamically selects different types of variation operations based on the ranking of chromosomes’ scores in the population on each objective, which makes elite chromosomes better spread their features into the offspring with more different chromosomes, and makes inferior chromosomes more disrupted. Below we give detailed description of these two steps.

Adaptive Selection. For the current population and its parents, μMOEA adaptively determines the candidate solu-

tions to conduct crossover and mutation according to their fitness values and fitness level of the population. For the solution s_i with fitness f_i in the population p_n , f_{max} and f_{min} represent the maximal and minimal fitness values in p_n respectively. For each population p_n , μMOEA calculates the average values of fitness, represented as \bar{f} . The mutation probability of s_i is PM_i . For the two solutions s_i and s_j , their crossover probability is represented as $PC_{i,j}$, and $f_{i,j}$ is the larger of the fitness values.

For the selection of solutions for crossover, the higher the fitness value of one solution, the larger the probability of crossover between it and the other candidate solution. For s_i and s_j , The closer $f_{i,j}$ is to f_{max} , the larger the $PC_{i,j}$ is. The crossover probability of s_i and s_j is computed as follows:

$$PC_{i,j} = \min\{k_1(f'_{i,j} - f_{min})/(\bar{f} - f_{min}), k_3\}$$

where $0 < k_1, k_3 \leq 1$. If $PC_{i,j} \geq \text{threshold}_n$, μMOEA will conduct adaptive crossover among them.

For the selection of solutions for mutation, the smaller the fitness value of the solution, the higher the probability of mutating its parameters. For s_i , the closer f_i is to f_{min} , the larger the PM_i is. The mutation probability of s_i is computed as follows.

$$PM_i = \min\{k_2(f_{max} - f_i)/(f_{max} - \bar{f}), k_4\}$$

where $0 < k_2, k_4 \leq 1$. If $PM_i \geq \text{threshold}_n^m$, μMOEA will conduct adaptive mutation on it.

To disrupt the solutions with above-average fitness values to search the spaces for the region with global optimum, and ensure that all solutions with subaverage fitness values compulsorily undergo mutation, we use a value of 0.6 for k_1 and k_2 , and 1.0 for k_3 and k_4 . These values can be changed according to the actual needs.

Since the threshold of variation (threshold_n) has great effects on the overall variation of the population p_n , different from the MOEAs that pre-define a fixed value for the threshold, μMOEA computes threshold_n for each population p_n adaptively based on the population's overall level of fitness values. When the fitness values of the solutions in p_n converge, μMOEA decreases the threshold_n to facilitate the crossover and mutation to create more different offspring. Similarly, if the fitness values of the population scatter, μMOEA increases the threshold_n to accelerate the convergence to find an optimal solution. The calculation of threshold_n is given as follows:

$$\text{threshold}_n = c_1(f_{max} - \bar{f}) + m_1(\bar{f} - f_{min})$$

where $0 < c_1, m_1 \leq 1$.

Adaptive Variation. This includes adaptive crossover and adaptive mutation. The variation strategy has a higher probability that the generated offspring can integrate the advantages of parents in convergence and diversity.

For adaptive crossover, given two candidate solutions of crossover, based on the objective that the solution ranks highest in the population, the chromosome with the highest value on the objective is selected for crossover using

the single-point crossover with a random chromosome in the other candidate solution.

For adaptive mutation, given the candidate solution of mutation, different types of mutation operations are dynamically determined based on the fitness values of chromosomes in the solution. If the chromosome has a high score on any objective in the population, μMOEA adjusts its parameters slightly (e.g., modifying the parameters of some genes on it) to better explore the surrounding space. Otherwise, μMOEA makes major changes to it, e.g., changing the combinations or sequences of genes on it, adding new actions/operations into it, replacing some genes with new actions/operations.

Guiding LLM to Generate Differential Seeds

For the generated solutions, μMOEA runs them to detect the safety violations in the MCDL system. During the adaptive evolutionary search, we find that as the iterations increase, the evolutionary search is prone to falling into local optimality, causing the newly generated solutions similar to those optimal solutions generated by previous generations. To solve the issue, when the evolutionary search gets stuck, μMOEA generates differential seed solutions for the next generation, which encourages the exploration of more diverse solutions.

Specifically, when the high-fitness solutions in t consecutive generations remain the same, μMOEA selects the optimal solutions generated by the previous iterations (collecting their chromosomes in SE), and generates the feedback prompt using Rule 1 in Table 2 (where SN represents the chromosomes of non-optimal solutions). The prompt pattern of Rule 1 is to promote the LLM to learn the characteristics of previous evolutionary iterations, and then create differential seed solutions leveraging its reasoning capability.

Considering that the LLM is typically accustomed to generating outputs that resemble the examples provided in the input, μMOEA examines the differences between the LLM's generated solutions to the input solutions. For the solution s_i , its difference from the input solutions is calculated as: $d_i = (\sum_{x_o \in (SN \cup SE)} ED_{s_i, x_o})/r$, where ED represents the Euclidean Distances between two solutions. For the LLM's generated solutions that do not meet the difference requirements with the previous solutions, μMOEA generates the prompt based on Rule 2 in Table 2 to make the LLM re-generate qualified differential seed solutions.

Evaluation

To evaluate the effectiveness and advancement of our proposed method, we apply μMOEA to search for solutions that detect safety violations of the representative MCDL system, and compare μMOEA 's performance to the state-of-the-art method.

Experiment Setup

MCDL System. Autonomous driving systems (ADSs) exemplify a prototypical case of multi-component deep learning (MCDL) systems, comprising various components built upon multiple deep learning models. These components and

Table 2: Rules for feedback prompt generation

Rule	Sample of feedback prompt
1	Each solution in $\langle SE \rangle$ exposed a safety violation of $\langle MCDL \text{ system} \rangle$. So they are what we want. No safety violation occurred in $\langle SN \rangle$, which are not required by us. We want you to generate $\langle N \rangle$ solutions that can expose safety violations and differentiate from $\langle SE \rangle$
2	The solutions in $\langle R \rangle$ are not different enough from $\langle SE \rangle$. Please re-generate to create new solutions that have high potential to expose safety violations of $\langle MCDL \text{ system} \rangle$.

models engage in high-frequency communication and input-output interactions. Given the considerable social implications of autonomous driving technology, detecting safety violations of ADSs is of substantial importance.

We select the industrial full-stack ADS, Baidu Apollo (apo 2013) to evaluate the ability of μMOEA in finding safety violations of MCDL systems, due to the representativeness, practicality and advancedness. (1) Representativeness. Apollo ranks among the top 4 leading industrial ADS developers (Funicello-Paul April 1, 2024) (the other three ADSs, Waymo, Ford, and Cruise, are not released publicly). (2) Practicality. Apollo can be readily installed on vehicles for driving on public roads (Hersey April 1, 2024) (it has provided self-driving services for real vehicles (autoware-foundation Sepetem 1, 2023; Baidu April 1, 2024)). (3) Advancedness. Apollo is actively and rapidly updated (the releases of Apollo update on a weekly basis).

Test Platform. We conducted the experiments on Ubuntu 20.04 with 500 GB memory, an Intel Core i7 CPU, and an NVIDIA GTX2080 TI. SORA-SVL (Huai 2023) (an end-to-end AV simulation platform which supports connection with Apollo) and San Francisco map are selected to execute the generated solutions. During the experiments, all modules of Apollo are turned on, including perception module, localization module, prediction module, routing module, planning module, and control module.

Evaluation Metrics. To evaluate the effectiveness of the method in detecting diverse safety violations of the ADS, the metrics include the following aspects:

- How many types of safety violations are detected?
- How many solutions are generated on average to detect one safety violation?
- How long does it take on average to detect the first safety violation and all found types of safety violations?

To metric how distinct the different types of detected safety violations are, we calculate the average difference among them by Euclidean Distances. For two solutions s_i and s_j , the calculation of Euclidean Distance, ED_{s_i, s_j} is given as follows. The larger the distance, the more different the safety-violation types.

$$ED_{s_i, s_j} = \frac{\sum_{n=1}^{|s_i|} \sum_{m=1}^{|s_j|} TD_{s_i^n, s_j^m}}{|s_i| * |s_j|}$$

$$TD_{s_i^n, s_j^m} = \sum_{k=0}^{\alpha} \sqrt{(x_{s_i^n, k} - x_{s_j^m, k})^2 + (y_{s_i^n, k} - y_{s_j^m, k})^2}$$

Table 3: Comparison results of μMOEA , μMOEA_r and μMOEA_n

	μMOEA	μMOEA_r	μMOEA_n
types of detected SV	10	10	6
number of solutions to detect one SV	12	12.7	24.3
time to detect the first SV	11min	39min	17min
time to detect all types of SVs	14h	15h	22h

s_i^n represents the n-th chromosome in the s_i . $|s_i|$ and $|s_j|$ represent the number of chromosomes in s_i and s_j respectively. $(x_{s_i^n, k}, y_{s_i^n, k})$ represents the position of k-th gene of the n-th chromosome in s_i . α represents the minimal number of genes in the two chromosomes s_i^n and s_j^m .

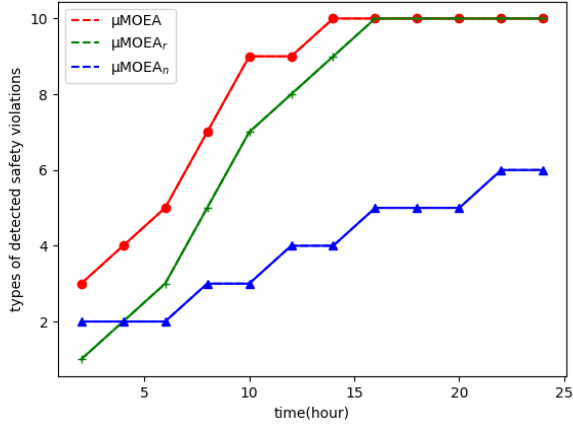
Effectiveness of μMOEA

We run μMOEA for 24 hours to detect safety violations of Apollo. For the found safety violations, we analyze their root causes by locating the incorrect operations of modules in Apollo. Furthermore, based on the analysis, we classify the found safety violations into distinct types. To account for the randomness, the experiments are repeated five times and the average results are provided as follows.

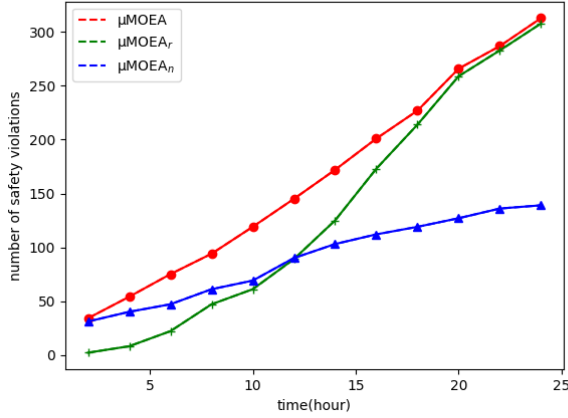
For each run, on average, 3756 solutions (min 3346 and max 4015) are generated by μMOEA and 313 (min 292 and max 355) out of them have safety violations of Apollo. μMOEA can detect 10 distinct types of safety violations of Apollo, which are all revealed in the first 14 hours.

To evaluate the benefit of the LLM-based initial population creation and differential seed generation, we conduct the ablation experiments of μMOEA . Two variant versions μMOEA_r and μMOEA_n are implemented. μMOEA_r creates the initial population by random initialization of solutions, and μMOEA_n evolves the solutions without differential seeds. We run μMOEA , μMOEA_r and μMOEA_n for the same amount of time, and compare their effectiveness and efficiency. The results are shown as Table 3 (where SV is the abbreviation for safety violation) and Figure 2.

μMOEA_r can detect 10 types of safety violations of



(a) The growth in the found safety violation types over time



(b) The growth in the number of generated safety violations over time

Figure 2: The running results of μMOEA , μMOEA_r , μMOEA_n

Apollo, and μMOEA_n can detect 6 types of safety violations of Apollo. On average, in the 24-hour run, μMOEA_r generates 3922 solutions (min 3853 and max 4014), and 308 of them (min 289 and max 315) detect safety violations of Apollo. For μMOEA_n , it generates 3352 solutions (min 3099 and max 3480), and 138 of them (min 121 and max 150) detect safety violations of Apollo.

Table 3 shows that μMOEA_r takes the most time to detect the first safety violation of Apollo. It can be seen from Figure 2(a) that in early-generation solutions, the number of safety violations detected by μMOEA_r is the least. We can conclude that μMOEA 's creation of initial population creation helps generate better initial population than random initialization.

For μMOEA_n , Table 3 shows that the number of safety violation types detected by it is fewer than μMOEA and μMOEA_r , and it takes more time to detect all found types of safety violations. The average Euclidean Distance across the detected different safety violation types of μMOEA is 81.10 meters, and that of μMOEA_r is 69.72 meters. From Figure 2(b), we can see that as the iterations of evolutionary

search increase, the growth of safety violations detected by μMOEA_n is slowest. We can conclude that the differential seeds of μMOEA can help solve the local optimal and detect more diverse types of safety violations.

It's worth noting that, during the iterations of μMOEA and μMOEA_r , after a few types of safety violations have been found, the detection of safety violations grows faster. We analyze that it benefits from the feedback-based differential seed generation. As the evolutionary iterations increase, the solutions of feedback increase. μMOEA can better learn more experience about the characteristics of optimal solutions, which can improve the quality of the generated differential seed solutions.

Advancement of μMOEA

We evaluate μMOEA in comparison to the state-of-art method that uses the multi-objective genetic algorithm (NSGA-II) to detect safety violations of Apollo: MOSAT (Tian et al. 2022). MOSAT generates the first population by randomly initialized individuals. Based on them, MOSAT uses multi-objective genetic algorithm to search for optimal and diverse solutions. The individuals are evaluated by multi-objective fitness function, which contains the elitism and diversity. MOSAT determines crossover probability and mutation probability of parent individuals by random rates and the fixed variation threshold. The variation operators that are defined to manipulate individuals include uniform crossover and mutation.

We run MOSAT on the same road in San Francisco as μMOEA . For the sake of fairness, in each 24-hour running, the number of individuals in each generation of MOSAT and μMOEA are the same. The comparison results of μMOEA and MOSAT are shown as Table 4.

Table 4: Comparison results of μMOEA and MOSAT

Approach		μMOEA	MOSAT
types of detected SV		10	6
number of solutions to detect one SV	min	10.1	61.0
	max	13.7	65.9
	avg	12	62.1
time to detect the first SV (min)	min	1	2
	max	12	28
	avg	7	16
time to detect all found types of SVs (hour)	min	11.3	16.0
	max	13.6	18.9
	avg	12.9	18.1

In each 24-hour running, MOSAT can find 6 types of safety violations of Apollo. On average, MOSAT generates 3541 solutions (min 3208 and max 3719), and 57 (min 49 and max 61) out of them detect safety violations of Apollo.

μMOEA detects 10 distinct types of safety violations of Apollo and all of them are detected in the first 14 hours.

MOSAT detects 6 types of safety violations of Apollo and all of them are detected in the first 19 hours. The average Euclidean Distance across the detected different safety violation types of μMOEA is 81.10 meters, and that of μMOEA_r is 64.70 meters. Moreover, the 6 types of safety violations are all revealed in the 10 types of safety violation detected by μMOEA . It takes μMOEA less time to detect the first safety violation of Apollo than MOSAT. Therefore, compared with MOSAT, μMOEA can detect more types of safety violations of Apollo in a shorter time. On average, one safety violation of Apollo occurs in 12 solutions generated by μMOEA . MOSAT generates 62 solutions to find one safety violation of Apollo. The safety-violation exposure frequency in μMOEA is higher, which shows that μMOEA can efficiently detect more safety violations of Apollo. The comparison results show that μMOEA can more effectively detect more types of safety violations in the MCDL system.

Related Work

Large Language Models for Reasoning

Recent advancements in large language models (LLMs) have demonstrated their potential in a variety of tasks, including quantitative reasoning. LLMs have tremendous capabilities in solving complex tasks, from quantitative reasoning to understanding natural language (Romera-Paredes et al. 2024). Early models such as GPT-3 have been shown to generate coherent text and perform well in tasks that require contextual understanding, but they often struggle with more complex reasoning tasks, particularly those that involve multi-step logic or abstract thinking.

To address these limitations, researchers have proposed various approaches to enhance the quantitative reasoning capabilities of LLMs. One notable method is the integration of external knowledge bases (Mann et al. 2020), which has been shown to improve the accuracy and depth of reasoning by providing models with additional contextual information. Another approach is the use of prompt engineering (Reynolds and McDonnell 2021), where carefully designed prompts guide the model towards better reasoning outcomes. Furthermore, there has been growing interest in the application of LLMs to quantitative reasoning tasks in specific domains, such as mathematical problem-solving, where domain-specific training data can significantly enhance the model performance.

Multi-Objective Evolutionary Algorithms

Multi-objective evolutionary algorithms (MOEAs) have become a prominent tool for solving multi-objective optimization problems due to their ability to find a set of Pareto-optimal solutions in a single run (Deb et al. 2002; Zitzler, Laumanns, and Thiele 2001). Populations in MOEAs generally evolve through high-performing candidate solutions being mutated or recombined to form the next generation.

One of the most typical MOEAs is the Non-dominated Sorting Genetic Algorithm II (NSGA-II), which introduced key innovations such as fast non-dominated sorting and crowding distance mechanisms to find the optimal solutions and maintain solution diversity. NSGA-II has since become

a benchmark for comparing other MOEAs due to its balance between computational efficiency and solution quality. In recent years, many variants of MOEAs have been proposed to address specific challenges (Fonseca, Fleming et al. 1993; Ishibuchi, Tsukamoto, and Nojima 2008). These approaches have been extended further with hybrid methods, combining MOEAs with local search or other optimization techniques to enhance their performance in complex optimization scenarios (Zitzler and Thiele 1999). This has led to a better understanding of the trade-offs involved in using MOEAs for different types of multi-objective problems, including those with a large number of objectives, known as many-objective optimization problems.

Conclusion and Discussion

In this paper, we propose μMOEA , an LLM-empowered adaptive evolutionary search method for multi-component deep learning systems. Different from existing MOEAs that detect safety violations of MCDL systems starting by randomly initialized population, μMOEA leverages LLM’s ability in language understanding and quantitative reasoning to better comprehend the evolutionary task and create high-quality solutions for the initial population. Based on these, μMOEA adopts an adaptive multi-objective evolutionary algorithm to efficiently search for optimal and diverse solutions. To navigate the search away from the local optima, when the evolutionary process gets stuck, μMOEA feedbacks the characteristics of iterations into the LLM to facilitate the learning of evolutionary experience and population characteristics. Then it promotes the LLM to generating differential seed solutions for the next generation. We use μMOEA to detect safety violations of a representative MCDL system, industrial autonomous driving system. Furthermore, we evaluate the performance of μMOEA by ablation experiments and compare it with the state-of-the-art method that uses multi-objective genetic algorithm to search solutions for diverse safety violations of the system. The experimental results show that μMOEA can effectively and efficiently detect safety violations of MCDL systems and surpass the state-of-the-art method.

To leverage the LLM’s capability, μMOEA inputs the prompt into GPT-4 by sending the API request, which brings extra time cost for waiting the model output. Generally, the time cost for each request is about 40 seconds, which generates 6 solutions on average. The extra time cost for a solution generated by GPT is 7 seconds. Moreover, the LLM has a limit on the number of input characters, which limits the potential capability of μMOEA due to the limited number of samples for context learning and thought chain. Currently, μMOEA generates the feedback prompt incrementally and updates the early iterations with latest iterations when the characters of the prompt exceeds the limit. As future work, we aim to employ a local LLM to reduce the time cost for request, and intelligently select the input examples for the feedback, with the potential to further improve the performance and ability of μMOEA .

References

2013. An open autonomous driving platform.
- Abdukhamidov, E.; Abuhamad, M.; Woo, S. S.; Chan-Tin, E.; and Abuhmed, T. 2023a. Microbial Genetic Algorithm-based Black-box Attack against Interpretable Deep Learning Systems. *arXiv preprint arXiv:2307.06496*.
- Abdukhamidov, E.; Abuhamad, M.; Woo, S. S.; Chan-Tin, E.; and Abuhmed, T. 2023b. Unveiling Vulnerabilities in Interpretable Deep Learning Systems with Query-Efficient Black-box Attacks. *arXiv preprint arXiv:2307.11906*.
- Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.
- Asharf, J.; Moustafa, N.; Khurshid, H.; Debie, E.; Haider, W.; and Wahab, A. 2020. A review of intrusion detection systems using machine and deep learning in internet of things: Challenges, solutions and future directions. *Electronics*, 9(7): 1177.
- autowarefoundation. Sepetem 1, 2023. Autoware Overview.
- Baidu, I. April 1, 2024. Baidu Launches Public Robotaxi Trial Operation.
- Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.
- Borg, M.; Englund, C.; Wnuk, K.; Duran, B.; Levandowski, C.; Gao, S.; Tan, Y.; Kaijser, H.; Lönn, H.; and Törnqvist, J. 2018. Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry. *arXiv preprint arXiv:1812.05389*.
- Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2): 182–197.
- Ehrgott, M.; Ide, J.; and Schöbel, A. 2014. Minmax robustness for multi-objective optimization problems. *European Journal of Operational Research*, 239(1): 17–31.
- Fonseca, C. M.; Fleming, P. J.; et al. 1993. Genetic algorithms for multiobjective optimization: formulation discussion and generalization. In *Icga*, volume 93, 416–423. Cite-seer.
- Funicello-Paul, L. April 1, 2024. Navigant Research Names Waymo, Ford Autonomous Vehicles, Cruise, and Baidu the Leading Developers of Automated Driving Systems.
- Goodfellow, I.; Papernot, N.; Huang, S.; Duan, Y.; Abbeel, P.; and Clark, J. 2017. Attacking machine learning with adversarial examples. *OpenAI Blog*, 24: 1.
- Hassija, V.; Chamola, V.; Mahapatra, A.; Singal, A.; Goel, D.; Huang, K.; Scardapane, S.; Spinelli, I.; Mahmud, M.; and Hussain, A. 2024. Interpreting black-box models: a review on explainable artificial intelligence. *Cognitive Computation*, 16(1): 45–74.
- Hersey, F. April 1, 2024. Baidu launches their open platform for autonomous cars—and we got to test it.
- Huai, Y. 2023. SORA-SVL Simulator.
- Ishibuchi, H.; Tsukamoto, N.; and Nojima, Y. 2008. Evolutionary many-objective optimization: A short review. In *2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence)*, 2419–2426. IEEE.
- Long, Q. 2014. A constraint handling technique for constrained multi-objective genetic algorithm. *Swarm and Evolutionary Computation*, 15: 66–79.
- Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 1.
- Mishra, D. B.; Mishra, R.; Acharya, A. A.; and Das, K. N. 2019. Test case optimization and prioritization based on multi-objective genetic algorithm. In *Harmony Search and Nature Inspired Optimization Algorithms: Theory and Applications, ICHSA 2018*, 371–381. Springer.
- Reynolds, L.; and McDonell, K. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, 1–7.
- Romera-Paredes, B.; Barekatin, M.; Novikov, A.; Balog, M.; Kumar, M. P.; Dupont, E.; Ruiz, F. J.; Ellenberg, J. S.; Wang, P.; Fawzi, O.; et al. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995): 468–475.
- Roumeliotis, K. I.; and Tselikas, N. D. 2023. Chatgpt and open-ai models: A preliminary review. *Future Internet*, 15(6): 192.
- Tian, H.; Jiang, Y.; Wu, G.; Yan, J.; Wei, J.; Chen, W.; Li, S.; and Ye, D. 2022. MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 94–106.
- Varshney, K. R. 2016. Engineering safety in machine learning. In *2016 Information Theory and Applications Workshop (ITA)*, 1–5. IEEE.
- Wirsansky, E. 2020. *Hands-on genetic algorithms with Python: applying genetic algorithms to solve real-world deep learning and artificial intelligence problems*. Packt Publishing Ltd.
- Zhang, S.; Roller, S.; Goyal, N.; Artetxe, M.; Chen, M.; Chen, S.; Dewan, C.; Diab, M.; Li, X.; Lin, X. V.; et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.
- Zhou, A.; Qu, B.-Y.; Li, H.; Zhao, S.-Z.; Suganthan, P. N.; and Zhang, Q. 2011. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and evolutionary computation*, 1(1): 32–49.
- Zitzler, E.; Laumanns, M.; and Thiele, L. 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK report*, 103.
- Zitzler, E.; and Thiele, L. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4): 257–271.

Reproducibility Checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced. [yes](#)
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results. [yes](#)
- Provides well marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper. [yes](#)

Does this paper make theoretical contributions? [no](#)

If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally.
- All novel claims are stated formally (e.g., in theorem statements).
- Proofs of all novel claims are included.
- Proof sketches or intuitions are given for complex and/or novel results.
- Appropriate citations to theoretical tools used are given.
- All theoretical claims are demonstrated empirically to hold.
- All experimental code used to eliminate or disprove claims is included.

Does this paper rely on one or more datasets? [no](#)

If yes, please complete the list below.

- A motivation is given for why the experiments are conducted on the selected datasets. (yes/partial/no/NA)
- All novel datasets introduced in this paper are included in a data appendix. (yes/partial/no/NA)
- All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes/partial/no/NA)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations. (yes/no/NA)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available. (yes/partial/no/NA)
- All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisfying. (yes/partial/no/NA)

Does this paper include computational experiments? [yes](#)

If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix. [yes](#)
- All source code required for conducting and analyzing the experiments is included in a code appendix. [yes](#)
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. [yes](#)

- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from. [partial](#)
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. [yes](#)
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. [yes](#)
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. [yes](#)
- This paper states the number of algorithm runs used to compute each reported result. [yes](#)
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. [yes](#)
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). [partial](#)
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. [yes](#)
- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. [yes](#)