FINE-TUNING LARGE LANGUAGE MODEL FOR AUTOMATED ALGORITHM DESIGN

Fei Liu[1], Rui Zhang[1], Xi Lin[2], Zhichao Lu[1] & Qingfu Zhang[1]
[1] City University of Hong Kong, Hong Kong, China
[2] Xi'an Jiaotong University, Xi'an, China
{fliu36-c, rui.zhang.cs}@my.cityu.edu.hk, xi.lin@xjtu.edu.cn,
{zhichaolu, qingfu.zhang}@cityu.edu.hk

ABSTRACT

The integration of large language models (LLMs) into automated algorithm design has shown promising potential. A prevalent approach embeds LLMs within search routines to iteratively generate and refine candidate algorithms. However, most existing methods rely on off-the-shelf LLMs trained for general coding tasks, leaving a key question open: *Do we need LLMs specifically tailored for algorithm design?* If so, how can such LLMs be effectively obtained and how well can they generalize across different algorithm design tasks? In this paper, we take a preliminary step toward answering these questions by exploring fine-tuning of LLMs for algorithm design. We introduce a *Diversity-Aware Rank-based (DAR)* sampling strategy to balance training data diversity and quality, then we leverage direct preference optimization to efficiently align LLM outputs with task objectives. Our experiments, conducted on *Llama-3.2-1B-Instruct* and *Llama-3.1-8B-Instruct*, span three distinct algorithm design tasks. Results suggest that fine-tuned LLMs can significantly outperform their off-the-shelf counterparts with the smaller *Llama-3.2-1B-Instruct* and match the larger *Llama-3.1-8B-Instruct* on the admissible set problem. Moreover, we observe promising generalization: LLMs fine-tuned on specific algorithm design tasks also improve performance on related tasks with varying settings. These findings highlight the value of task-specific adaptation for LLMs in algorithm design and open new avenues for future research.

1 INTRODUCTION

The emerging field of automated algorithm design (AAD) with large language models (LLMs) has attracted growing attention for its potential to automate the synthesis of expert-level algorithms (Liu et al., 2026; 2024a; Romera-Paredes et al., 2024). A prevailing paradigm in this space combines LLMs within search strategies, where the LLM focuses on generating candidate algorithms and the search procedures control the quality and refinement of these algorithms in an iterative manner (Zhang et al., 2024). This framework has led to notable advances across a spectrum of algorithmic development tasks, including combinatorial optimization (Liu et al., 2024a; Ye et al., 2024), Bayesian optimization (Yao et al., 2024), and black-box optimization (van Stein & Bäck, 2024), to name a few.

Despite these preliminary successes, most existing LLM-driven AAD approaches rely on off-the-shelf LLMs, posing two limitations: ❶ they require a large number of queries to LLMs, resulting in substantial computational overhead (Romera-Paredes et al., 2024; Novikov et al., 2025), and ❷ these methods exhibit marginal performance variations across different choice of LLMs (Liu et al., 2024b; Zhang et al., 2024), suggesting that current LLMs may lack inductive biases suited to algorithm design. These observations raise a potential need for specialized LLMs explicitly trained for algorithm design tasks. While prior work has explored domain-specific LLMs for general coding (e.g., programming) tasks (Jiang et al., 2026) and optimization problem formulation (Huang et al., 2025), and some recent efforts have fine-tuned LLMs during the search process to improve AAD performance (Huang et al., 2026; Surina et al., 2025), the development of LLMs tailored specifically for automated algorithm design remains largely underexplored.

Moreover, fine-tuning LLMs for algorithm design poses unique challenges distinct from conventional code generation (Jiang et al., 2026) or mathematical reasoning (Ahn et al., 2024) tasks. Algorithm design tasks rarely have clear ground-truth labels as optimal algorithms may not exist and cannot be evaluated by a single performance metric. The algorithm design process benefits from exploring diverse algorithms—even suboptimal ones—as they may introduce novel ideas or structural approaches that provide valuable insights and ultimately improve the final design. These characteristics render existing fine-tuning techniques insufficient for addressing the inherent complexity of fine-tuning LLMs for AAD.

In this work, we take a preliminary yet foundational step towards developing specialized LLMs for algorithm design tasks. Our investigation is guided by the following two research questions:

- **RQ1:** How can we effectively obtain LLMs specialized for algorithm design?
- **RQ2:** How well can these LLMs generalize across different algorithm design tasks?

To address **RQ1**, we fine-tune general-purpose, open-source LLMs—specifically, *Llama-3.2-1B-Instruct* and *Llama-3.1-8B-Instruct* (Grattafiori et al., 2024)—on algorithm design problems. For ASP, we additionally include OpenPangu models as auxiliary comparison models. We introduce a Diversity-Aware Rank-based (DAR) Sampling strategy to construct diverse preference pairs, which serve as training data for fine-tuning via Direct Preference Optimization (DPO) (Rafailov et al., 2023). We evaluate the resulting LLMs against their original counterparts in two settings: (i) random sampling, and (ii) integration within an existing AAD framework.

To address **RQ2**, we assess the generalization capabilities of LLMs fine-tuned on the Capacitated Vehicle Routing Problem (CVRP) across two scenarios: (i) generalization to variant settings of the same problem (e.g., CVRP instances with different sizes and capacity constraints), and (ii) transfer to a related but distinct algorithm design task—namely, the Traveling Salesman Problem (TSP). These evaluations allow us to examine both in-distribution and out-of-distribution generalization performance.

Our **key findings** are summarized as follows.

- Fine-tuning LLMs specifically for algorithm design is necessary and feasible. The proposed Diversity-Aware Rank-based Sampling strategy enables effective and robust LLM fine-tuning, underscoring the importance of considering diversity in algorithm preference pair construction.
- Fine-tuned LLMs significantly improve their capabilities in (1) algorithm design with LLM-based random sampling (Sec. 3.4), (2) algorithm design with LLM-based iterative search (Sec 3.4), and (3) similar/related algorithm design tasks (Sec. 3.5). Notably, *Llama-3.2-1B-Instruct* trained with our method matches the performance of *Llama-3.1-8B-Instruct*.

2 FINE-TUNE LLM FOR AAD

Rather than training algorithm design LLMs from scratch, we adopt a fine-tuning approach to adapt LLMs for automated algorithm design tasks. Among various learning methods, we employ Direct Preference Optimization (DPO), a reward-free method that trains models to prefer high-quality outputs over inferior ones using preference pairs.

As shown in Figure 1, our framework consists of two stages: 1) Data Generation: We use LLM-driven iterative algorithm search (e.g., EoH (Liu et al., 2024a)) to generate diverse algorithms (**Upper section**). 2) Preference Learning: The collected algorithms are sampled to compose preference pairs (samples), enabling the LLM to learn preferred designs over less favoured ones (**Lower section**). The fine-tuned LLMs can be used to generate algorithms.

2.1 DATA GENERATION

We employ LLMs in iterative search methods (e.g., EoH (Liu et al., 2024a) and FunSearch (Romera-Paredes et al., 2024)) to generate algorithms. These search methods maintain a population of algorithms, using LLMs to generate new algorithms or refine existing ones, thereby evolving the population over time. Unlike repeated sampling with LLMs, this approach produces high-quality,

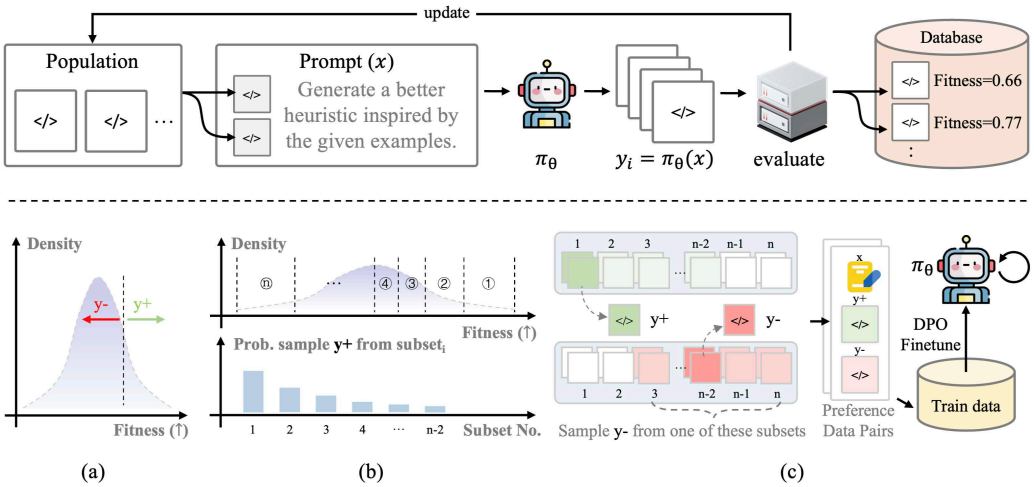


Figure 1: **Upper section:** LLM-based automated algorithm design methods iteratively optimize algorithms. Through this, algorithms and their fitness are preserved in the database D. The knowledge and experiences incorporated in the database subsequently improve the capabilities of the LLM. **Lower section:** (a) Traditional sampling relies on continuous fitness values and often suffers from unstable preference gaps. (b) Our method discretizes the fitness space into ranked subsets, enabling structured sampling of high-quality y_+ and clearly worse y_-. (c) These sampled pairs are combined with prompts and code templates to form training triples (samples) (x, y_+, y_-) for fine-tuning.

diverse algorithms (Zhang et al., 2024), enabling more effective learning. Throughout this process, we record all valid algorithms generated to form the algorithm database D. These algorithms in D are subsequently used to construct training datasets D_t using a diversity-aware rank-based sampling strategy, as detailed in the next section.

2.2 Diversity-Aware Rank-Based Sampling

A natural and intuitive method for constructing preference pairs is to define a fitness threshold and sample positive algorithms (y_+) above this threshold and negative algorithms (y_-) from those below it (as illustrated in Figure 1(a)). A similar idea is also adopted in prior work, such as EvoTune (Surina et al., 2025), where only those preference pairs in which the y_+ exhibits top-tier performance are retained for training. While this strategy can ensure high-quality positive algorithms, discarding a large number of potentially informative samples with mid-range performance may limit the diversity of preference signals available to the learning model.

To address these limitations, we propose a diversity-aware rank-based sampling strategy that constructs preference pairs in a more structured yet flexible way. This method aims to strike a balance between **quality emphasis** (favoring stronger candidates) and **diversity** (preserving a range of heuristic qualities), leading to more informative and robust preference supervision.

Let the algorithm database be denoted by $D = \{y_1, \ldots, y_N\}$, where the candidates are sorted in **descending** order of fitness, i.e., y_1 is the best and y_N the worst. We partition D into M *equally-sized* and disjoint subsets:

$$S_1,\ S_2,\ \ldots,\ S_M, \quad \text{with } |S_m| = \left\lfloor \frac{N}{M} \right\rfloor \quad (m = 1, \ldots, M),$$

so that S_1 contains the top-ranked, S_2 the next best, and so on (see Figure 1(b)). We perform following steps to obtain a preference sample pair (x, y_+, y_-):

1. Subset selection (biased toward higher quality). From the first $M-2$ subsets, we pick an index $i \in \{1, \ldots, M-2\}$ with probability

$$\Pr(i) = \frac{\exp((M-2-i)/\tau)}{\sum_{k=1}^{M-2} \exp(k/\tau)},$$

where $\tau > 0$ is a *temperature* hyperparameter (we set to 3.0 by default). A smaller τ sharpens the distribution, increasing the chance of drawing from higher-quality subsets. The *temperature* τ controls exploitation vs. exploration: When $\tau \to 0$ the positive sample will almost always draw from the very best subset(s); While when $\tau \to \infty$, it reduces to uniform sampling over the first $M-2$ subsets.

2. Positive algorithm. Sample one candidate uniformly from the chosen subset:

$$y_+ \sim \text{Uniform}(S_i).$$

3. Negative algorithm. To ensure a clear performance gap, we skip the nearest subset (i.e., S_{i+1}) and sample uniformly from the rest subsets:

$$y_- \sim \text{Uniform}\left(\bigcup_{j=i+2}^{M} S_j\right).$$

Note that we skip subset S_{i+1} in this step, which enforces a minimum gap of one quality tier, yielding clearer supervision signals.

4. Preference sample construction. A major distinction in our implementation of standard DPO lies in how we construct the prompt x. Different from standard DPO, where y_+ and y_- algorithms are conditioned on a prompt x, we adopt a fixed prompt template consisting of two components: (1) a description of the algorithm design task to be solved, and (2) a function template and skeleton representing the expected format of the algorithm. The resulting preference sample pair is a triplet $(x,\ y_+,\ y_-)$. Once a preference sample pair is obtained, we remove y_+ and y_- from the database to eliminate duplication. This process is repeated to construct a training dataset D_t.

Empirical studies in Sec. 3.4 demonstrate that the proposed sampling strategy maintains a balance between selecting high-quality heuristics and preserving diversity, thus producing rich and instructive training signals for algorithm preference learning.

2.3 DPO FINE-TUNING

We employ Direct Preference Optimization (DPO) (Rafailov et al., 2023) to fine-tune LLMs using constructed training samples. The LLM acts as a policy $\pi_\theta(y|x)$, where x is an input prompt and y is a generated algorithm. Our objective is to optimize the LLM's preferences toward high-performing algorithms while maintaining generalization.

To optimize the policy π_θ, we use an objective with regularization to ensure the outputs remain close to those of the reference model $\pi_{\text{ref}}(y|x)$:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}_t, y\sim\pi_\theta}[r(x,y)] - \beta\mathbb{D}_{\text{KL}}[\pi_{\text{ref}}(\cdot|x) \parallel \pi_\theta(\cdot|x)], \tag{1}$$

where $\mathbb{D}_{\text{KL}}$ is the reverse KL-divergence, β controls the regularization strength, and π_{ref} is the initial base LLM policy π_θ^0. While reward maximization methods like PPO (Schulman et al., 2017) exist, they are computationally expensive. Instead, we formulate the task as preference optimization, allowing algorithm ranking based on $r(x,y)$.

For a training dataset $\mathcal{D}_{\text{t}} = \{(x^i, y_+^i, y_-^i)\}_{i=1}^n$, the loss function is:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_+,y_-)\sim\mathcal{D}_{\text{t}}}\left[\log\sigma\left(\beta\left(\log\frac{\pi_\theta(y_+|x)}{\pi_{\text{ref}}(y_+|x)} - \log\frac{\pi_\theta(y_-|x)}{\pi_{\text{ref}}(y_-|x)}\right)\right)\right], \tag{2}$$

where σ is the sigmoid function. This approach eliminates the need for separate reward models or complex reward heuristics used in RL methods like GRPO (Huang et al., 2026), enabling efficient off-policy optimization.

We apply low-rank adapters (LoRA) (Hu et al., 2022) to fine-tune the model efficiently. LoRA introduces a small number of trainable parameters into existing layers of the model, allowing effective

adaptation without updating the full parameter set. The adoption of LoRA is further motivated by the inherent challenges of our task: collecting sufficient algorithm data is expensive, as each candidate algorithm must undergo a computationally expensive evaluation phase during search. Therefore, applying LoRA is particularly important in our case, as the size of our training data may not be sufficient to support full fine-tuning without overfitting. Empirically, we have observed that full fine-tuning failed to converge during training on our datasets.

3 EXPERIMENTAL STUDIES

3.1 ALGORITHM DESIGN TASKS

Admissible Set Problem (ASP) ASP aims to maximize the size of the set while fulfilling the criteria below: (1) The elements of the set are vectors belonging to $\{0, 1, 2\}^n$. (2) Each vector has the same number w of non-zero elements but a unique support. (3) For any three distinct vectors, there is a coordinate in which their three respective values are $\{0, 1, 2\}$, $\{0, 1, 2\}$, $\{0, 1, 2\}$. Following prior works (Romera-Paredes et al., 2024), we set $n = 15$ and $w = 10$ in this work.

Traveling Salesman Problem (TSP) TSP aims to find a route that minimizes the total traveling distance for a salesman required to visit each city exactly once before returning to the starting point. We investigate the constructive heuristic design for TSP. Specifically, we adopt an iteratively constructive framework to start from one node and iteratively select the next node until all nodes have been selected and back to the start node. The task is to design a heuristic for choosing the next node to minimize the route length.

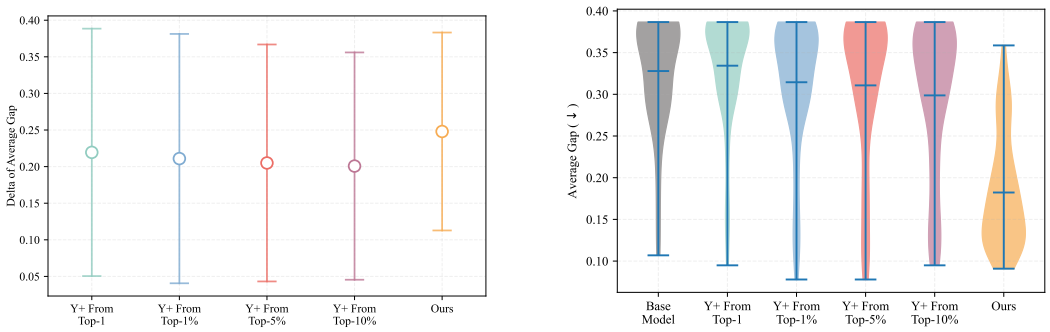
Capacitated Vehicle Routing Problem (CVRP) CVRP aims to minimize the total traveling distances of a fleet of vehicles given a depot and a set of customers with coordinates and demands. The problem is constrained by: (1) The vehicles start from the depot and return to the depot; (2) Each customer should be visited only once; (3) All the demands should be satisfied while the capacity of the vehicle should not be exceeded. Similar to TSP, we adopt an iteratively constructive framework to start from one node and iteratively select the next node until all nodes have been selected and return to the depot. The task is to design a heuristic for selecting the next node to minimize the total route length with all constraints satisfied.

3.2 EXPERIMENTAL SETTINGS

Data Generation For each algorithm design task, we collect a diverse set of algorithmic solutions from existing results produced by FunSearch and EoH. These results are generated by *Llama-3.1-8B-Instruct* (Grattafiori et al., 2024). To standardize the code in the database, we first preprocess the raw code implementations by unifying their function templates, including consistent function names, docstrings, and input-output formats. Next, we discard identical algorithms by checking the code strings. Ultimately, we obtain approximately 60,000 unique algorithms for the ASP and CVRP problems, respectively. We adopt LLM4AD (Liu et al., 2024b) implementations for both FunSearch and EoH.

Fine-tuning We fine-tune each LLM for five epochs with a batch size of eight. We initiate the learning rate at 5e-6, applying a cosine decay schedule and a warmup rate of 0.05 to reduce the rate over the training period gradually. The model processes inputs up to a maximum length of 2048 tokens. For DPO, we set the $\beta = 0.4$ and utilize LoRA with settings of $r = 64$ and $\alpha = 32$, alongside a dropout rate of 0.05. The fine-tuning and inference processes for *Llama-3.2-1B-Instruct* (Llama-1B) (Grattafiori et al., 2024) and *Llama-3.1-8B-Instruct* (Llama-8B) (Grattafiori et al., 2024) are executed on NVIDIA L20 GPUs, while those for *openPangu-Embedded-1b-v1.1* (Pangu-1B) (Chen et al., 2025) and *openPangu-Embedded-7b-v1.1* (Pangu-7B) (Chen et al., 2025) are conducted on a Huawei Ascend 910B server. We use `trl` library (von Werra et al., 2020) for DPO implementations and `vllm` library (Kwon et al., 2023) for efficient LLM inference.

Automated Algorithm Design We test the fine-tuned LLMs in two types of settings: 1) repeated sampling, where we use the same prompt to instruct LLMs to generate algorithms many times. This setting can show the effectiveness of fine-tuning in a straightforward way. 2) Iterative algorithm



(a) (b)

Figure 2: Comparison on varying preference pair sampling settings. **(a):** Comparison of the delta value of preference pairs sampled by different methods. The delta value is calculated by the absolute difference of the average gap between y_+ and y_- data. The mean delta values of 250 preference pairs are highlighted with circle markers, while the standard deviations are denoted by lines. **(b):** Violin plot comparison on the performance of LLMs fine-tuned on various preference datasets. Each violin reflects the performance distribution of the top 50 of 1,000 algorithms generated by each LLM. The performance is determined by the average gap to the existing best-known algorithm.

design, where we use the fine-tuned LLMs in EoH and FunSearch for automated algorithm design and compare to the results using pre-trained LLMs without fine-tuning. We set the maximum number of evaluations to be 2,000 for both EoH and FunSearch and EoH's population size to be 20.

3.3 Comparative Evaluation of Sampling Strategies

In this experiment, we evaluate the efficacy of our proposed sampling strategy against top-k-based sampling strategies. We construct four distinct datasets using the top-k-based sampling approach under the following configurations: 1) **Top-1 Sampling:** The positive samples (y_+) are selected as the best-performing heuristic (exhibiting the lowest average gap) in the database, while the negative samples (y_-) are randomly chosen from the remaining heuristics. 2) **Top-k% Sampling:** The y_+ samples are randomly drawn from the top-k% heuristics, whereas the y_- samples are selected from the rest $(100 - k)\%$. We test different parameters $k = 1, 5, 10$ for this strategy.

We evaluate the performance of the Llama-1B model, which has been fine-tuned on ASP. To quantify the differences between these strategies, we analyze the delta value of preference pairs, defined as the difference in average gap values between y_+ and y_- samples within each pair. This metric reflects the relative performance gap between the compared heuristics.

As illustrated in Figure 2a, we visualize the delta value distributions for datasets generated by the five sampling strategies, each comprising 250 preference pairs. The mean delta values (marked with circles) and standard deviations (denoted by lines) reveal that the proposed sampling strategy significantly increases the pairwise distances between preference samples compared to top-k-based methods.

We assess the effectiveness of different sampling methods as follows: Firstly, we randomly sample 1,000 feasible algorithms that have been successfully evaluated on ASP using both the base model and its fine-tuned variants. The prompts used for this random sampling are identical to x in the preference pair. Subsequently, we identify and analyze the top 50 algorithms from this pool of 1,000, plotting the distribution of their average gap values. As illustrated in Figure 2b, fine-tuning with our proposed sampling strategy markedly enhances the model's capability in designing algorithms. This improvement is quantitatively demonstrated by a significantly reduced mean average gap among the top-50 algorithms when compared to those designed by the base model and other sampling methods.

3.4 Performance Improvement via Fine-tuning

Random Sampling In this experiment, we investigate whether a fine-tuned smaller LLM can match larger LLMs in terms of algorithm design capabilities. Specifically, we fine-tune Pangu-1B and

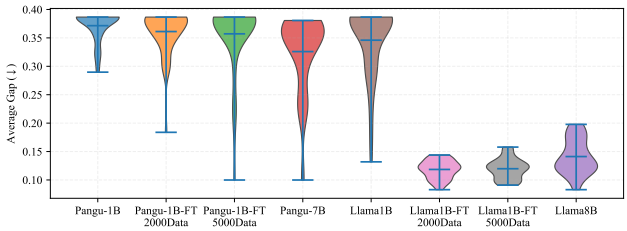
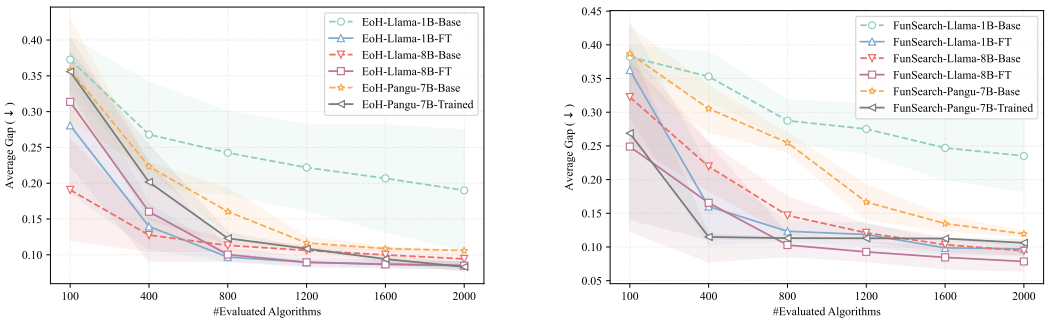


Figure 3: Violin plot comparison on the performance of fine-tuned LLMs and base model on ASP. Each violin reflects the performance distribution of the top 50 of 1,000 algorithms generated by each LLM. The performance is determined by the average gap to the existing best-known algorithm, with lower being better.



(a) EoH (b) FunSearch

Figure 4: Convergence curve comparison on the performance of top-5 algorithms generated by various LLMs and search methods on ASP. The performance is determined by the average gap to the best-known algorithm, with lower being better. The mean performance averaged over three independent runs is denoted by markers, while the standard deviations are demonstrated by the shaded area.

Llama-1B on datasets of 2,000 and 5,000 preference pairs and compare them with their base models. On ASP, we further include Pangu-7B and Llama-8B as larger auxiliary comparison models.

Figure 3 compares the performance of the base models, the 1B models fine-tuned on two dataset sizes, and the larger auxiliary comparison models. Similar to the previous experiment, we randomly sample 1,000 feasible algorithms and plot the distribution of the average gap of the top 50 algorithms, with lower being better. It can be concluded from the results that: i) Fine-tuning consistently improves the two 1B models on ASP. ii) Increasing the dataset size yields marginal but consistent improvements. iii) The fine-tuned Llama-1B achieves competitive performance with Llama-8B, highlighting the effectiveness of our approach. iv) Although fine-tuning also improves Pangu-1B, its average gap remains higher than that of the larger Pangu-7B model.

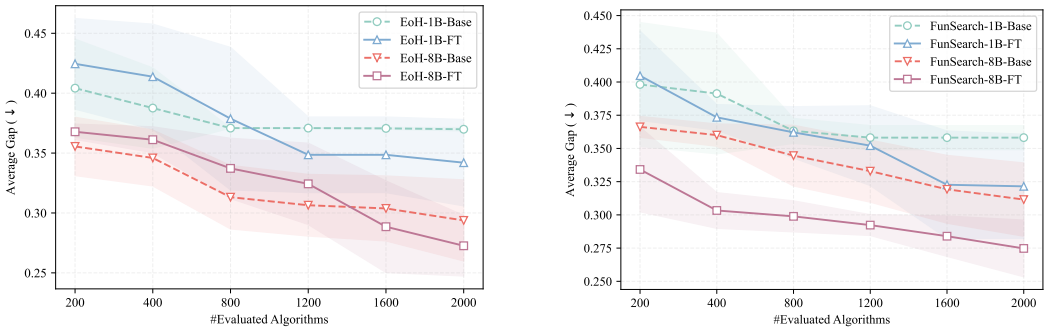
Table 1: Performance comparison on ASP for the top-1 and top-10 heuristics. The performance is determined by the average gap to the best-known optimum (%). The mean and standard deviation aggregated over three independent runs are reported, with lower being better.

		Llama-1B	Llama-1B-FT	Llama-8B	Llama-8B-FT	Pangu-7B	Pangu-7B-FT
FunSearch	Top-1	$19.48_{\pm 4.31}$	$9.19_{\pm 0.28}$	$8.46_{\pm 0.46}$	$7.43_{\pm 1.22}$	$11.00_{\pm 1.23}$	$10.94_{\pm 0.49}$
	Top-10	$24.22_{\pm 5.03}$	$9.88_{\pm 0.97}$	$9.88_{\pm 0.84}$	$8.09_{\pm 1.5}$	$13.35_{\pm 1.87}$	$11.19_{\pm 0.34}$
EoH	Top-1	$17.02_{\pm 8.88}$	$8.52_{\pm 0.56}$	$8.99_{\pm 0.62}$	$8.19_{\pm 0.53}$	$8.81_{\pm 0.23}$	$8.68_{\pm 0.59}$
	Top-10	$19.49_{\pm 8.34}$	$8.58_{\pm 0.52}$	$9.83_{\pm 0.85}$	$8.69_{\pm 0.46}$	$9.69_{\pm 0.72}$	$9.66_{\pm 0.79}$

Iterative Search We couple the fine-tuned LLM with two state-of-the-art LLM-driven AAD methods, FunSearch and EoH. We initialize all compared methods with the respective seed algorithm

Table 2: Performance comparison on CVRP for the top-1 and top-10 heuristics. The performance is determined by the average gap to the best-known optimum (%). The mean and standard deviation aggregated over three independent runs are reported, with lower being better.

		Llama-1B	Llama-1B-FT	Llama-8B	Llama-8B-FT
FunSearch	Top-1	$35.81_{\pm 0.90}$	$30.26_{\pm 2.02}$	$30.23_{\pm 1.73}$	$27.38_{\pm 2.06}$
	Top-10	$37.81_{\pm 0.91}$	$33.27_{\pm 4.23}$	$31.91_{\pm 2.89}$	$27.98_{\pm 2.38}$
EoH	Top-1	$36.98_{\pm 0.15}$	$31.61_{\pm 4.05}$	$28.45_{\pm 3.21}$	$27.06_{\pm 2.7}$
	Top-10	$37.02_{\pm 0.11}$	$34.64_{\pm 3.51}$	$30.22_{\pm 3.29}$	$27.38_{\pm 2.5}$



(a) EoH (b) FunSearch

Figure 5: Convergence curve comparison on the performance of top-5 algorithms generated by various LLMs and search methods on CVRP problem. The performance is determined by the average gap to the existing best-known algorithm. The mean performance averaged over three independent runs is denoted by markers, while the standard deviations are demonstrated by the shaded area.

on each problem. We set the maximum number of evaluated programs to 2,000. The maximum evaluation time for each heuristic is restricted to 30 seconds to eliminate inefficient and harmful heuristics, such as infinite loops. We perform three independent runs for each method to account for experimental biases.

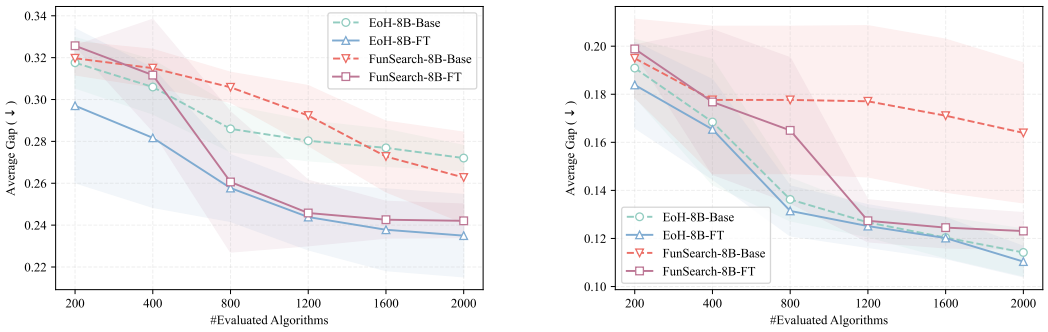
We evaluate Llama-1B and Llama-8B in this study. For the ASP problem, we additionally include Pangu-7B and its fine-tuned variant as auxiliary comparison models. For each task, we generate a dataset of 2,000 preference pairs using our proposed sampling strategy and fine-tune the corresponding models considered in that task. After that, the fine-tuned LLMs, as well as the base models, are utilized to generate algorithms under the guidance of search methods.

Figures 4 (ASP problem) and 5 (CVRP problem) present the convergence curves of the performance of top-5 algorithms, with markers indicating mean performance across three runs and shaded regions denoting standard deviations. Complementary results for the performance of top-1 and top-10 algorithms are listed in Tables 1 and 2. Results show that: i) The fine-tuned LLMs consistently outperform base models, achieving faster convergence and smaller optimality gaps. ii) The 1B LLMs can approach the performance of 8B LLMs in most cases, with the exception of EoH on the ASP, where the 1B LLM is noticeably inferior to the base 8B LLM.

These results demonstrate that preference learning enhances LLMs' algorithm design capabilities, which in turn elevate search performance. This underscores the importance of integrating search strategies with fine-tuned LLMs.

3.5 Generalization of Fine-tuned LLMs

In this section, we investigate the capacity of online fine-tuned LLMs to enhance algorithm search on new algorithm design tasks. We employ the Llama-8B model. The model has been fine-tuned on the CVRP with instance size 50, and we adopt it on two new tasks without any further adaptation. These two settings represent different levels of generalization: 1) The same task under a different distribution. 2) A related task with a different problem description.



(a) CVRP-100 Problem

(b) TSP-50 Problem

Figure 6: Convergence curve comparison on the performance of top-5 algorithms generated by the base model and LLMs fine-tuned on the CVRP-50 problem. The performance is determined by the average gap to the existing best-known algorithm, with lower being better. The mean performance averaged over three independent runs is denoted by markers, while the standard deviations are demonstrated by the shaded area.

Table 3: Performance comparison on the top-1 and top-10 heuristics. The performance is determined by the average gap to the best-known optimum (%). The mean and standard deviation aggregated over three independent runs are reported, with lower being better.

		Performance on CVRP-100		Performance on TSP-50	
		Top-1	Top-10	Top-1	Top-10
FunSearch	Llama-8B	$26.27_{\pm 2.16}$	$26.74_{\pm 1.95}$	$15.53_{\pm 2.51}$	$16.88_{\pm 2.98}$
	Llama-8B-FT	$24.1_{\pm 0.87}$	$24.26_{\pm 0.84}$	$11.85_{\pm 0.37}$	$12.52_{\pm 0.73}$
EoH	Llama-8B	$26.88_{\pm 0.42}$	$27.47_{\pm 0.69}$	$10.87_{\pm 0.84}$	$11.76_{\pm 1.03}$
	Llama-8B-FT	$23.34_{\pm 2.01}$	$23.62_{\pm 1.95}$	$10.3_{\pm 0.74}$	$11.57_{\pm 0.83}$

Same Task with Different Distribution We evaluated the performance of the fine-tuned Llama-8B model on a variant of the CVRP. Originally fine-tuned for CVRP instances with 50 nodes, the model is now tested on larger instances containing 100 nodes. The coordinates for these nodes are randomly generated within the [0,1] interval, and the vehicle capacity has been increased to 50. This setup introduces variations in both the number of nodes and the capacities, while the task description remains consistent. Results in Figure 6 and Table 3 demonstrate that the fine-tuned LLMs are effective when we change the task settings. The average results clearly outperform base LLMs on both EoH and FunSearch.

Related Task with Different Description The fine-tuned Llama-8B is also tested on the Traveling Salesman Problem (TSP), which, while related to CVRP, differs significantly in terms of problem description and attributes. The results in Figure 6 demonstrate that even on a different task, fine-tuned LLMs can still improve automated algorithm search. However, the improvement is less pronounced on EoH, likely because EoH already converges efficiently with the base model.

4 Conclusion

This paper presents a preliminary study on the necessity and effectiveness of fine-tuning an LLM tailored to the algorithm design task. We adopt DPO and introduce a diverse-aware rank-based sampling strategy, which balances training data diversity and quality for effective finetuning on algorithm design tasks. Our experiments on three tasks demonstrate the effectiveness of the fine-tuned LLM across different algorithm design scenarios, including: algorithm design with LLM-based random sampling, algorithm design with LLM-based iterative search, and generalizing to related algorithm design tasks. Notably, Llama-1B trained with our method matches the performance of Llama-8B.

REFERENCES

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. arXiv preprint arXiv:2402.00157, 2024.

Hanting Chen, Yasheng Wang, Kai Han, Dong Li, Lin Li, Zhenni Bi, Jinpeng Li, Haoyu Wang, Fei Mi, Mingjian Zhu, Bin Wang, Kaikai Song, Yifei Fu, Xu He, Yu Luo, Chong Zhu, Quan He, Xueyu Wu, Wei He, Hailin Hu, Yehui Tang, Dacheng Tao, Xinghao Chen, and Yunhe Wang. Pangu embedded: An efficient dual-system llm reasoner with metacognition, 2025. URL https://arxiv.org/abs/2505.22375.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. ICLR, 1(2):3, 2022.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. Orlm: A customizable framework in training large models for automated optimization modeling. Operations Research, 2025.

Ziyao Huang, Weiwei Wu, Kui Wu, Wei-Bin Lee, and Jianping Wang. CALM: Co-evolution of algorithms and language model for automatic heuristic design. In The Fourteenth International Conference on Learning Representations, 2026. URL https://openreview.net/forum?id=x6bG2Hoqdf.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. ACM Transactions on Software Engineering and Methodology, 35 (2):1–72, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In International Conference on Machine Learning, pp. 32201–32223. PMLR, 2024a.

Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Llm4ad: A platform for algorithm design with large language model. arXiv preprint arXiv:2412.17287, 2024b.

Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, et al. A systematic survey on large language models for algorithm design. ACM Computing Surveys, 58(8):1–32, 2026.

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.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. Advances in Neural Information Processing Systems, 36:53728–53741, 2023.

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.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

Anja Surina, Amin Mansouri, Lars Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with llms: Evolutionary search meets reinforcement learning. arXiv preprint arXiv:2504.05108, 2025.

Niki van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. IEEE Transactions on Evolutionary Computation, 2024.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`, 2020.

Yiming Yao, Fei Liu, Ji Cheng, and Qingfu Zhang. Evolve cost-aware acquisition functions using large language models. In International Conference on Parallel Problem Solving from Nature, pp. 374–390. Springer, 2024.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. Advances in neural information processing systems, 37:43571–43608, 2024.

Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Understanding the importance of evolutionary search in automated heuristic design with large language models. In International Conference on Parallel Problem Solving from Nature, pp. 185–202. Springer, 2024.