LaMDAgent: An Autonomous Framework for Post-Training Pipeline Optimization via LLM Agents

Taro Yano
NEC Corporation
taro_yano@nec.com

Yoichi Ishibashi
NEC Corporation
yoichi-ishibashi@nec.com

Masafumi Oyamada
NEC Corporation
oyamada@nec.com

Abstract

Large Language Models (LLMs) have demonstrated exceptional performance across a wide range of tasks. To further tailor LLMs to specific domains or applications, post-training techniques such as Supervised Fine-Tuning (SFT), Preference Learning, and model merging are commonly employed. While each of these methods has been extensively studied in isolation, the automated construction of complete post-training pipelines remains an underexplored area. Existing approaches typically rely on manual design or focus narrowly on optimizing individual components, such as data ordering or merging strategies. In this work, we introduce LaMDAgent (short for Language Model Developing Agent), a novel framework that autonomously constructs and optimizes full post-training pipelines through the use of LLM-based agents. LaMDAgent systematically explores diverse model generation techniques, datasets, and hyperparameter configurations, leveraging task-based feedback to discover high-performing pipelines with minimal human intervention. Our experiments show that LaMDAgent improves tool-use accuracy by 9.0 points while preserving instruction-following capabilities. Moreover, it uncovers effective post-training strategies that are often overlooked by conventional human-driven exploration. We further analyze the impact of data and model size scaling to reduce computational costs on the exploration, finding that model size scalings introduces new challenges, whereas scaling data size enables cost-effective pipeline discovery.

1 Introduction

Large Language Models (LLMs) have undergone rapid development, demonstrating exceptional performance across diverse tasks and significantly impacting both academic and industrial domains, with the rise of high-performing proprietary models (OpenAI, 2023; Anthropic, 2024; Google, 2024) as well as open-sourced models (Dubey et al., 2024; Yang et al., 2024; DeepSeek-AI et al., 2024; Abdin et al., 2024). LLM development typically involves pre-training on large-scale web corpora followed by post-training with curated data (Ouyang et al., 2022), with this study focusing on the latter stage due to the increasing emphasis on post-training driven by the release of models and datasets tailored for domain and task adaptation (Tie et al., 2025).

In post-training, widely adopted approaches include Supervised Fine-Tuning (SFT) using human-created prompt-response pairs and Preference Learning based on preference labels for response pairs (Rafailov et al., 2023; Ethayarajh et al., 2024; Song et al., 2024; Munos et al., 2024). Furthermore, innovative techniques are rapidly evolving, such as autonomous training data generation and "model merging" that creates new models through arithmetic operations on different model parameters (Wortsman et al., 2022; Ilharco et al., 2023; Yadav et al., 2023). To generate superior models, existing studies either manually build pipelines or focus on optimizing specific steps such as fine-tuning data orderings (Chen et al., 2023; Kim and Lee, 2024; Pattnaik et al., 2024) or model merging strategies (Ishibashi et al., 2025; Akiba et al., 2025). However, full adaptation for target tasks requires combining these methods into integrated pipelines to optimize, yet automating this end-to-end process remains largely unexplored.

In this paper, we propose Language Model Developing Agent (LaMDAgent), a method that autonomously constructs post-training pipelines using LLM-based agents and continuously improves them based on feedback from the generated model's performance on target tasks. LaMDAgent treats heterogeneous model improving methods such as supervised fine-tuning, preference learning, or model merging in a unified manner and automates end-to-end post-training pipeline con-

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 30078–30095
November 4-9, 2025 ©2025 Association for Computational Linguistics

struction by exploring appropriate model generation methods, datasets, hyperparameters, and their optimal application order, thereby reducing the specialized knowledge and human costs required for pipeline construction.

Additionally, to reduce computational costs for LaMDAgent's exploration, we experimentally verify data size scaling and model size scaling, where data size scaling and model size scaling respectively involve exploring pipelines with smaller data quantities and model sizes, then transferring the discovered efficient pipelines to larger data quantities and model sizes.

The contributions of this paper are as follows:

1. We propose an LLM Agents-driven framework "LaMDAgent" that autonomously constructs and optimizes post-training pipeline. LaMDAgent treats heterogeneous model improving methods in a unified framework to optimize the entire pipeline in post-training, reducing the specialized knowledge and human costs required for pipeline construction.

2. In our experiments across two distinct settings, we show that LaMDAgent effectively improves mathematical capability by 3.7 points in average accuracy in Experiment 1 and enhances tool utilization accuracy by 9.0 points in Experiment 2 compared to strong baselines, while maintaining general capabilities through the discovery of novel pipelines that are not easily identified by humans.

3. To reduce LaMDAgent's exploration costs, we verify the effectiveness of data size scaling and model size scaling, finding that model size scalings introduces new challenges, whereas scaling data size enables cost-effective pipeline discovery.

2 Methodology

2.1 Overview

We propose a novel method called Language Model Developing Agent (LaMDAgent) that fully automates the construction and optimization of language model post-training pipelines using LLM Agents. Figure 1 illustrates the overview of our proposed method. The proposed method aims to create better models by iteratively repeating the following four steps: 1. Action enumeration , 2. Action selection, 3. Model evaluation, and 4. Memory update. While stopping criteria can be based on cost, runtime, or evaluation metrics, our experiments use a fixed number of iterations and select the pipeline with the highest reward value, corresponding to the best performance on validation tasks. Details of our method are described in the following sections.

2.2 Action Enumeration

For simplicity of explanation, we define the following terms:

- Object: A concrete entity used in the model training pipeline, such as Llama 3 8B as a model or GSM8k as training data.

- Action: An action is a model improving method that takes multiple objects, including models, as input and outputs a new model. An action is defined by an action type such as "SFT" and the objects used, such as specific data, models, or hyperparameters.

We obtain possible actions by enumerating all combinations of action types and objects. Specifically, we use predefined action types and objects that include both pre-prepared datasets and models, as well as models and data obtained during the iteration. For example, if we have the action type "SFT" is defined to take (base model, training data) as input objects, and we have Gemma2 2B as a base model and GSM8k and MATH as training data, then possible actions can be enumerated as (Gemma2 2B, GSM8k) and (Gemma2 2B, MATH).

2.3 Action Selection

We use the agent to select one promising model improvement action from possible actions. During action selection, we provide the agent with a prompt for action selection and parse its output to determine the action. In practice, rather than providing all action candidates and having the agent output a single action index, we first have the agent select an action type in one inference step and identify the required object types based on the action type. Then, we have the agent select objects in an another inference step to determine the final action. We first decide on the action type to avoid action parsing failures that might occur if we give the agent the complex task of selecting the action type, understanding what object types are needed for each action type, and selecting objects without excess or deficiency. We select all objects in a single inference step rather than multiple steps

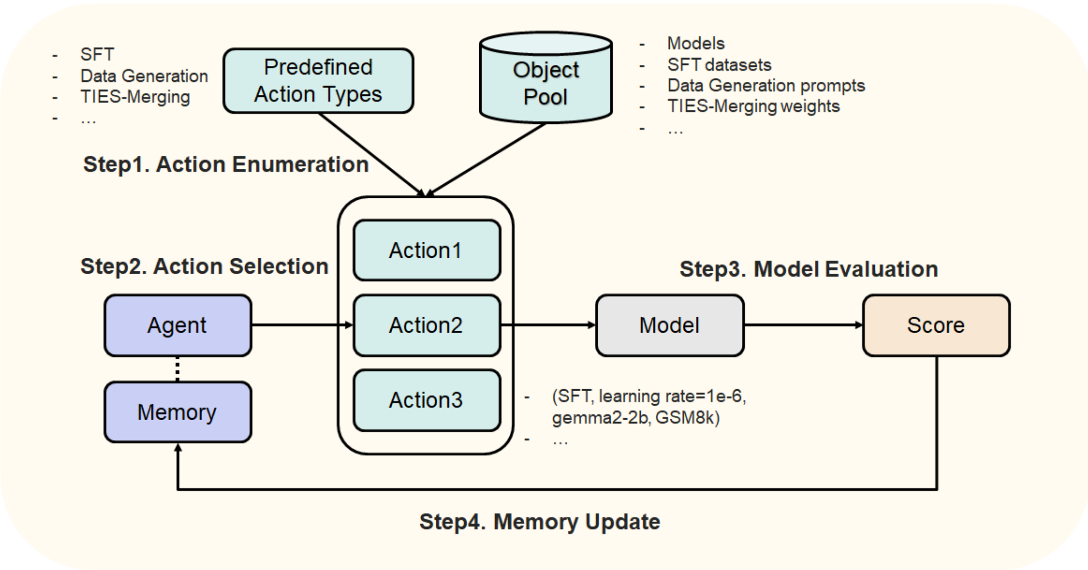

Figure 1: Overview of our LaMDAgent framework. LaMDAgent first enumerates actions from predefined model improving action types and an object pool containing available data, models, parameters, and other objects (Step1. Action Eunumeration). Next, the agent selects an action based on memory acquired from previous trials and executes the selected action to generate a new model (Step2. Action Selection). Then evaluations on downstream tasks are conducted (Step3. Model Evaluation). Based on the evaluation results of the newly generated model, the agent considers promising future directions and insights, updating the accumulated memory (Step4. Memory Update).

to minimize order dependency in the selection process.

Action selection process can be written as

$$a_{type} = Agent\left(g_{type}(m, l_{type})\right), \quad (1)$$

$$a_{obj} = Agent\left(g_{obj}(m, l_{obj}, a_{type})\right), \quad (2)$$

where a_{type}, a_{obj}, g_{type} and g_{obj} are determined action type, objects, prompt templates for selecting action types and objects, respectively. The prompts include memory m summarizing experiences from past trials, candidate action types l_{type}, and objects l_{obj}. The actual g_{type} and g_{obj} used in our experiments is provided in Appendix D. In preliminary trials, we observed mode collapse phenomena where the agent kept selecting the same action as steps progressed, so we explicitly included exploration directives in the prompt. We also added instructions to remove bias after observing that models generated at intermediate step i named as "Model i" tended to be selected less frequently compared to initial models like "Model GSM8k".

2.4 Model Evaluation

We evaluate the selected actions based on the performance of the resulting model on target tasks and provide feedback to the agent through numerical scores. In single-task settings, the evaluation metric itself can be used as the score, but in multi-task settings, we need to aggregate metrics across tasks. To account for different scales of evaluation metrics across tasks, we define the multi-task score s_{multi} using the following formula:

$$s_{multi} = \sum_k \alpha_k \cdot s^k_{single}, \quad (3)$$

where s^k_{single} is the single-task reward for task k, and $0 \leq \alpha_k \leq 1$ are scaling factors. In this research, we determine these factors so that the maximum contribution of each s_{single} is uniform. For example, MT-Bench metrics range up to 10, while AceBench metrics reach 1, so we set the weights for each score as $\alpha_{MT} = 1/10, \alpha_{Ace} = 1$.

2.5 Memory Update

We update memories of the agent based on the feedback received for the selected action. A memory is a text summarizing experiences from the latest and past trials, and next promising directions to explore. Memory at iteration t is derived from the action at t-th iteration (a^t_{type}, a^t_{obj}), score s^t, and template g_{mem} as follows:

$$m^t = Agent(g_{mem}((a^t_{type}, a^t_{obj}, r^t), \{(a^{t'}_{type}, a^{t'}_{obj}, r^{t'})\}_{t'<t}, \{m^{t'}\}_{t'<t})). \quad (4)$$

The memory updating template g_{mem} used in our experiments is provided in Appendix D.

3 Experiment 1: Teaching Multiple Skills to Base Models

3.1 Experimental Setup

We use Gemma2 2B (Rivière et al., 2024) [1] as our base model, and we target the following tasks in a multi-task setting: the arithmetic reasoning task GSM8k (Cobbe et al., 2021), the commonsense reasoning task Commonsense QA (CQA) (Talmor et al., 2019), and the reading comprehension task Trivia QA (TriviaQA) (Joshi et al., 2017), all converted to 0-shot format. For out-of-distribution evaluation tasks, we use GSMSymbolic (Mirzadeh et al., 2025), which is a more complex version of GSM8k with rewritten numbers in the questions, generative arithmetic reasoning tasks from NumGLUE (Mishra et al., 2022) Type1 (NumGLUE1) and Type2 (NumGLUE2), the social common sense reasoning task SocialIQA (Sap et al., 2019), and the reading comprehension task Natural Questions (NQ) (Kwiatkowski et al., 2019).

While our approach can handle any action that produces a single model, the actions used in this experiment and their required objects are:

- TIES-Merging (TIES): Model 1, Model 2, merge weight (fixed), merge density (fixed)
- Supervised Fine-Tuning (SFT): Model, SFT training data, learning rate (fixed)

TIES is a representative model merging technique, while SFT is a standard training method using log-likelihood maximization loss. As initial objects, we prepared specialist models trained on 1,000 examples each from GSM8k, CQA, and TriviaQA using Gemma2 2B as the base model, hereafter referred to as GSM8k-specialist, CQA-specialist, and TriviaQA-specialist. We also use the training data same as specialist models along with an aggregated all data as initial objects for SFT. For hyperparameters, we fix merging weights of $(0.5, 0.5)$, density of 0.5, and learning rate for SFT as $1e-6$. We use 100 examples from a different split as validation data, and test data was held out from both training and validation data. To eliminate variability from randomness, temperature was set to 0 during both pipeline exploration and testing. For the agent LLM, we use gpt-4o-2024-08-06 and performed 100 iterations of action selection and feedback.

[1] https://huggingface.co/google/gemma-2-2b

For details of evaluation methods, GSM8k involves parsing the final numeric answer from the prediction and exact matching with the ground truth, CQA requires the answer choice to be the form of "[[choice]]" and checking if the parsed value is correct, and TriviaQA uses exact match between normalized predictions and ground truth. For out-of-distribution tasks, GSMSymbolic, NumGLUE1, and NumGLUE2 uses the same evaluation method as GSM8k, SocialIQA uses the same as CQA, and NQ uses the same as TriviaQA.

For compared methods, in addition to the GSM8k, CQA, and TriviaQA specialists, we use TIES (Grid Search), which optimizes the weights of the three specialists through grid search, and Fully Fine-Tuned, which is trained on all available training data. To evaluate the effectiveness of LLM-based action selection, we also compare with Policy=Random, Actions=(SFT, TIES)) which randomly selects actions for 100 iterations, and Policy=LLM, Actions=(TIES) which removes the SFT from predefined action types.

3.2 Results

The experimental results are shown in Table 1. LaMDAgent Top-i refers to the model with the i-th highest accuracy on validation set among those generated by LaMDAgent. Bold values indicate the best performance among comparison methods, and underlined values indicate the top three.

LaMDAgent outperforms baselines, enhancing math skills while preserving others. Compared to the best baseline, Fully Fine-Tuned, LaMDAgent Top-1 shows 1.9 point improvement in overall accuracy (Avg) on the test set, demonstrating the effectiveness of the discovered pipeline. The improvement is particularly notable in arithmetic reasoning tasks, with LaMDAgent Top-1 overperforms Fully Fine-Tuned by 3.7 points on math-related tasks (GSM8k, GSMSymbolic, NumGLUE1, NumGLUE2) on average, while maintaining comparable performances on other tasks. These results suggest that, even with identical training data, appropriately combining model merging and training sequences using LaMDAgent can incorporate multiple skills more effectively than simple SFT on all the data.

Training is more effective than model merging for acquiring multiple skills. Interestingly, unlike findings in some previous works (Morrison et al., 2024; Kuroki et al., 2024), in our experimental setting, Fully Fine-Tuned, which was trained on all

Table 1: LaMDAgent effectively balances multiple skills and generalizes out-of-distribution: The main results of Experiment 1. LaMDAgent achieves the highest average performance (Avg) among compared methods. Notably, LaMDAgent Top-1 overperforms Fully Fine-Tuned by 3.7 points on math-related tasks (GSM8k, GSMSymbolic, NumGLUE1, NumGLUE2) on average, while maintaining performance on other tasks, demonstrating more effective multi-skill acquisition than simply mixing training data or merging specialist models.

	In-Distribution			Out-of-Distribution					
Method	GSM8k	CQA	TriviaQA	GSMSymbolic	NumGLUE1	NumGLUE2	SocialIQA	NQ	Avg
Baselines									
GSM8k-specialist	**0.320**	0.001	0.000	0.132	**0.425**	**0.395**	0.030	0.000	0.163
CQA-specialist	0.018	0.671	0.002	0.007	0.050	0.034	**1.000**	0.008	0.224
TriviaQA-specialist	0.046	0.027	**0.675**	0.017	0.050	0.280	0.905	**0.269**	0.284
TIES (Grid Search)	0.105	0.559	0.562	0.017	0.175	0.265	0.999	0.219	0.363
Fully Fine-Tuned	0.254	0.622	0.672	0.142	0.325	0.238	**1.000**	0.256	0.439
Proposed									
LaMDAgent Top-1	0.284	0.628	0.670	0.145	0.375	0.302	**1.000**	0.259	0.458
LaMDAgent Top-2	0.306	0.627	0.673	0.140	0.350	0.361	**1.000**	0.248	**0.463**
LaMDAgent Top-3	0.267	**0.674**	0.658	**0.146**	0.300	0.256	**1.000**	0.250	0.444

Table 2: LLM-based action selection is effective: Ablation study results for LaMDAgent, with validation set scores shown in parentheses. Random action selection achieves only scores comparable to Fully Fine-Tuned, while LLM-based action selection achieves higher average performance. Additionally, the action space provided significantly impacts generated model performances.

Method	GSM8k	CQA	TriviaQA	Avg
Policy=LLM, Actions=(SFT, TIES))	**0.284 (0.350)**	**0.628 (0.710)**	**0.670 (0.750)**	**0.527 (0.603)**
Policy=Random, Actions=(SFT, TIES))	0.257 (0.280)	0.594 (0.660)	0.674 (0.730)	0.508 (0.556)
Policy=LLM, Actions=(TIES)	0.032 (0.030)	0.588 (0.670)	0.575 (0.670)	0.398 (0.456)

data, outperformed TIES (Grid Search), which optimizes model merging weights, on all in-distribution tasks and 4 out of 5 out-of-distribution tasks, showing a 7.6 point higher average accuracy.

Agent-based action selection is effective. The ablation results in Table 2 show that random action selection (Policy=Random, Actions=(SFT, TIES)) resulted in a 4.7 point decrease on the validation set and a 1.9 point decrease on the test set, demonstrating that the agent-based action selection is effective and random pipeline search failed to discover pipelines better than the baseline (0.508 vs. 0.516). This is likely because as iterations progress, random action selection tends to prioritize exploring combinations of model merging, which is less effective in this setting as shown in TIES results, over exploration of training data curricula. This occurs because as the number of models increases with iterations, the number of model merging action candidates grows quadratically, while the number of training candidates grows linearly, making the former more likely to be selected randomly [2].

[2]For example, after 50 iterations with a single predefined model, the total number of possible merge actions is $\binom{50+1}{2} = 1275$, whereas the number of SFT actions is 50×4; (data types) $= 200$, which is six times smaller.

The choice of action space significantly impacts performance. Removing SFT from the action space (Policy=LLM, Actions=(TIES)) led to decreases of 14.7 and 12.9 points on validation and test sets respectively, showing that the pre-defined action space significantly affects the final achievable accuracy.

Discovered pipelines. The highest-performing pipelines discovered by LaMDAgent are shown in Figure 3. The Top-1, 2, and 3 pipelines all have in common that they end with training on all data. For Top-1, the result is consistent with findings (Dong et al., 2024) that learning mathematical skills first before mixing with general skill data is beneficial for balancing mathematical skills like GSM8k with general skills like CQA and TriviaQA. Interestingly, while model merging is typically used as a final refinement stage after training, in our experiments, pipelines that merge before training (Top-2 and Top-3) also performs well.

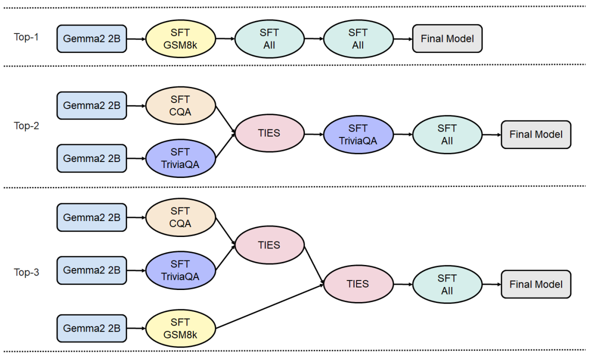

Figure 3: Top-1, Top-2, and Top-3 pipelines discovered in experiment 1.

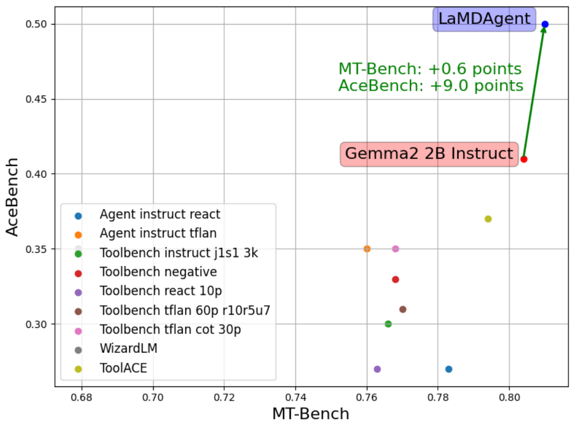

Figure 4: LaMDAgent significantly improves tool usage capability while maintaining instruction-following performance: The overall performance evaluation results of Experiment 2 indicate that LaMDAgent improves AceBench accuracy by 9.0 points while preserving the MT-Bench score. In contrast, naive fine-tuning approaches on either individual or full SFT datasets fail to enhance tool usage capabilities, suggesting that the task cannot be effectively addressed with such straightforward methods.

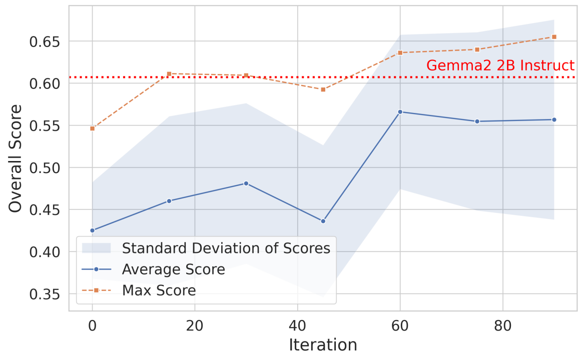

Figure 5: LaMDAgent learns from feedback to exploit promising actions while exploring unseen pipelines: The graph shows the Average Score, Max Score, and Standard Deviation recorded every 15 iterations. The consistent increase in average score indicates that the agent continues to learn from past feedback to exploit promising actions. The non-zero standard deviation through all iterations and improving max score implies that the agent maintains exploration to discover further improvement opportunities alongside exploitation.

4 Experiment 2: Enhancing Tool Usage Skills in Instruction-tuned Models

4.1 Experimental Setup

In a more realistic setting, we test whether LaMDAgent can enhance a specific skill (tool usage in this case) while maintaining the original instruction-following capabilities of a publicly available instruction-tuned model, Gemma2 2B Instruct [3]. We use AceBench (Chen et al., 2025) to evaluate tool usage capabilities and the first turn of MT-Bench (Zheng et al., 2023) to evaluate instruction-following capabilities. For action types, we adopt TIES and SFT as in Experiment 1. Initial objects include Gemma2 2B Instruct as the model, and for tool usage training data we use Agent-FLAN [4] (Chen et al., 2024), which includes Toolbench react10p, Toolbench tflan 60p r10r5u7, Toolbench tflan cot 30p, Agent instruct react, Agent instruct tflan, Toolbench instruct j1s1 3k, and Toolbench negative, ToolACE [5] (Liu et al., 2024b), and general instruction-following data of WizardLM [6] (Xu et al., 2024). We randomly selected up to 1,000 examples from each of the 7 Agent-FLAN subsets, ToolACE, and WizardLM. As in Experiment 1, we use gpt-4o-2024-08-06 as the LLM for action selection and performed 100 iterations.

For evaluation, we use only turn 1 of MT-Bench for faster and more cost-effective assessment, with gpt-4o-2024-08-06 as the evaluator. For ACEBench, we report the accuracy in the Normal setting, which measures single-turn function call performance. The temperature parameter is set to 0 during both pipeline exploration and testing to eliminate randomness.

Compared methods include the Gemma2 2B Instruct, Individually fine-tuned models trained separately on each of the 9 training datasets, and a Fully fine-tuned model trained on all data. All fine-tuned models use the same hyperparameters as the SFT

[3]https://huggingface.co/google/gemma-2-2b-it

[4]https://huggingface.co/datasets/internlm/Agent-FLAN
[5]https://huggingface.co/datasets/Team-ACE/ToolACE
[6]https://huggingface.co/datasets/WizardLMTeam/WizardLM_evol_instruct_V2_196k

in LaMDAgent.

4.2 Results

The overall performance evaluation results of Experiment 2 are summarized in Figure 4. Also, figure 5 plots the average score, maximum score, and standard deviation of scores for models created by LaMDAgent at 15-iteration intervals.

LaMDAgent enhances tool usage capabilities of Gemma2 2B Instruct while preserving instruction-following capabilities. The best model generated by LaMDAgent achieves an MT-Bench score of 0.810, comparable to Gemma2 2B Instruct (0.804), while improving AceBench accuracy from 0.410 to 0.500—a 9.0 point improvement. This demonstrates successful enhancement of tool usage capabilities while maintaining instruction-following performance. In contrast, the all fine-tuned models, significantly degrades both instruction-following and tool usage capabilities. A possible explanation for this: Gemma2 2B Instruct may have already paid an "alignment tax" (Ouyang et al., 2022) through extensive instruction tuning, and so unstable that additional tool-focused training could cause catastrophic forgetting of pre-training knowledge easily unless the training pipeline is carefully selected. To support this hypothesis, as shown in Figure 5, while LaMDAgent occasionally takes destructive actions, it learns to avoid them over time through feedback from downstream task, allowing the agent to automatically avoid such actions regardless of the cause.

Exploiting from score-based feedback while exploring unseen pipelines. As shown in Figure 5, the average score continues to improve with iterations, confirming that the LaMDAgent framework effectively updates its memory to exploit promising actions. The continuous improvement in maximum score and non-zero values of standard deviation of scores suggests that the agent maintains exploration alongside exploitation.

LaMDAgent is more effective when training and target distributions do not match. The score difference between Fully Fine-Tuned and LaMDAgent in Experiment 1 was smaller than in Experiment 2, indicating that LaMDAgent provides greater benefits in Experiment 2. This is because when trainining and target distributions are the same, simply minimizing the loss function on target tasks can yield good performances, whereas when distributions differ (as in Experiment 2), minimizing loss on all training data doesn't necessarily minimize loss on target data. Such scenarios represent effective applications for LaMDAgent.

Discovered pipelines. Figure 6 shows the Top-1

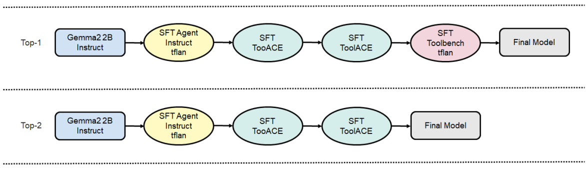

Figure 6: Top-1 and Top-2 pipelines discovered in experiment 2.

and Top-2 pipelines with the highest scores. Both pipelines train on Agent instruct tflan followed by ToolACE, suggesting these datasets were effective for AceBench. However, since the Fully Fine-Tuned model (which included these datasets) performs worse than the baseline Gemma2 2b Instruct, suggesting that excluding unnecessary data and establishing an appropriate training orderings are crucial. The Top-1 model's score evolution is 0.442 (SFT on Agent instruct tflan) $\rightarrow$ 0.592 (SFT on ToolACE) $\rightarrow$ 0.625 (SFT on ToolACE) $\rightarrow$ 0.655 (SFT on Toolbench tflan 60-r10r5u7), showing that similar performance improvements at each step cumulatively contributed to the final score, which cannot be easily identified by humans.

5 Reducing Computational Cost

In this section, we investigate the effectiveness of data size scaling and model size scaling inspired by pre-training scaling laws (Rivière et al., 2024) as methods to reduce the computational cost of LaMDAgent's pipeline exploration.

Data size scaling is effective. Data size scaling is based on the expectation that pipelines with high scores on small data sizes will maintain high scores when data size is increased. This approach involves exploring effective pipelines with small data sizes, then scaling up the data within those pipelines. For data size scaling to be effective, pipelines that outperform others with small data sizes must continue to outperform when data sizes are increased.

To verify the effectiveness of data size scaling, we examine how scores change when increasing the data in pipelines discovered in Experiment 1 by factors of 2, 4, and 6 times the exploration size. The results are shown in Figure 7. The Top-1 pipeline maintains the highest accuracy across all data sizes, demonstrating that pipelines with high accuracy on small data sizes maintain their advantage when scaled up, confirming the effectiveness of data size scaling for computational cost reduction.

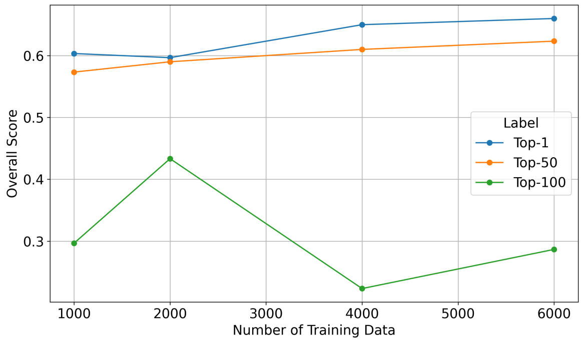

Figure 7: Data size scaling is effective for computational cost reduction: Overall score when scaling number of training examples in Top-k pipelines. The Top-1 pipeline consistently performs best, suggesting that effective pipelines at small scales remain effective with more data.

Table 3: Challenges exist in computational cost reduction via model size scaling: Evaluation scores of transferred pipelines on Gemma2 9B, suggesting that although some performance gaps are maintained, they sometimes diminish with model size scaling.

Method	Top-1	Top-50	Top-80	Top-90	Top-100
2B-based	0.603	0.573	0.553	0.546	0.297
9B-based	0.797	0.803	0.783	0.783	0.200

Model size scaling has limitations. Model size scaling is the model size version of scaling, which involves exploring effective pipelines with small models, then scaling up the models within those pipelines.

To verify the effectiveness of model size scaling, we change the base model from Gemma2 2B to Gemma2 9B to transfer the discovered pipelines in experiment 1. The results are shown in Table 3. When comparing Top-1 with Top-80, 90, and 100, which had score differences of more than 5 points with the 2B model, the Top-1 pipeline still achieves higher scores with 9B, showing that discovered pipelines remain effective when base model size increases. However, the score difference between Top-1 and Top-50, which was about 3 points with 2B models, reverses when scaling to the larger model, suggesting that small score gaps may disappear when increasing model size. Therefore, in practice, rather than pursuing small pipeline differences that risk disappearing, diversifying action space to explore large score gaps may be an effective use case for LaMDAgent when expecting model size scaling.

6 Related Work

LLM Agents. LLMs have evolved beyond chatbots to agents capable of executing diverse actions (Wang et al., 2024; Xi et al., 2025). ReAct (Yao et al., 2023) enables iterative reasoning via thought-action-observation loops, while Reflexion (Shinn et al., 2023) introduces verbal learning from feedback on past trajectories. Key areas include web automation (Ning et al., 2025; Zhou et al., 2024; Deng et al., 2023) and tool use (Patil et al., 2024; Qu et al., 2025; Qin et al., 2024). To our knowledge, this is the first work to automate post-training using LLM agents, treating improvement strategies as actions and model scores as rewards.

LLM for AutoML. The application of LLMs to Automated Machine Learning (AutoML) has emerged as a prominent investigation area. MLE-Bench (Chan et al., 2025) provides a comprehensive benchmark assessing LLM proficiency as machine learning practitioners, using 75 Kaggle competitions. Most methods (Yang et al., 2025; Liu et al., 2025b; Jiang et al., 2025; Chi et al., 2024; Trirat et al., 2024) use agentic approaches, utilizing LLMs for automatic code generation, improvement, and debugging, with model accuracy as feedback. Additionally, several agentic methods use LLMs to automatically improve LLMs themselves: Cheng et al. (2025); Liu et al. (2025a) optimizes LLM architecture, Lu et al. (2024) focuses on loss functions for preference learning, and Ishibashi et al. (2024) optimizes code for model merging algorithms. These studies focused on optimizing specific LLM aspects. To our knowledge, this is the first study to optimize the entire post-training process of LLMs while validating scaling methods for reducing computational costs in post-training pipeline search.

Curriculum Learning in Post-Training. Post-training performance is sensitive to the order of training data. SKILL-IT (Chen et al., 2023) prioritizes samples effective on validation sets. DMT (Dong et al., 2024) uses a two-stage process starting from specialized to general tasks. Kim and Lee (2024) proposes reordering based on attention scores, query length, and loss, while Curri-DPO (Pattnaik et al., 2024) begins with examples showing large preference gaps. Other domain-specific efforts exist (Zhao et al., 2021; Upadhyay et al., 2025; Qi et al., 2025). However, most rely on heuristics and expert knowledge. Our work aims to automate curriculum discovery via LLM agents.

Model Merging. Model merging combines parameters from multiple models via arithmetic operations. Wortsman et al. (2022) and Task Arithmetic (Ilharco et al., 2023) show that adding or subtracting parameters can enhance robustness or transfer task skills. Techniques like TIES-Merging (Yadav et al., 2023), DARE (Yu et al., 2024), and many others (Huang et al., 2024; Jang et al., 2024a,b; Khalifa et al., 2024; Ortiz-Jiménez et al., 2023; Liu et al., 2024a) continue to expand the field. MergeKit (Goddard et al., 2024) facilitates implementation of merging techniques. Evolutionary methods (Akiba et al., 2025) and skill-efficient merging (Morrison et al., 2024; Kuroki et al., 2024) optimize model merging parameters on target tasks. Since merging and training are not independent, optimizing both jointly is crucial. This paper is the first to propose a unified approach that automates both training and merging through LLM agents to construct optimal pipelines.

7 Conclusion

In this work, we propose LaMDAgent, an automated framework for constructing post-training pipelines via LLM-based agents. Empirical results across two experimental settings demonstrate that LaMDAgent substantially outperforms all baselines by autonomously identifying effective yet often-overlooked strategies by practitioners. To reduce the computational cost of pipeline exploration, we investigated scaling strategies and found that data-size scaling offers benefits, whereas model-size scaling poses nontrivial challenges. These findings position LaMDAgent as a promising direction toward automating and systematizing post-training pipeline design, thereby reducing reliance on domain expertise and facilitating broader accessibility in LLM adaptation.

8 Limitations

Our experiments were conducted using Gemma 2 as the base model. It remains to be investigated how the outcomes might change when different base models are used. Furthermore, we only used English-language datasets. While our method is not expected to be highly language-dependent, it remains unclear whether it performs adequately on minority or low-resource languages.

In principle, the proposed framework allows for arbitrary action types. However, in this study, we focused on TIES-Merging and Supervised Fine-Tuning. It would be highly interesting to explore what kinds of pipelines could be discovered by combining our method with other model merging techniques, preference learning approaches, or data generation strategies.

Our experiments did not yield positive results in the context of model size scaling. Therefore, achieving positive transfer at larger scales may require further innovation in future work.

References

Marah I Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat S. Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Parul Chopra, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Dan Iter, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Chen Liang, Weishung Liu, Eric Lin, Zeqi Lin, Piyush Madan, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Xia Song, Masahiro Tanaka, Xin Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Michael Wyatt, Can Xu, Jiahang Xu, Sonali Yadav, Fan Yang, Ziyi Yang, Donghan Yu, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *CoRR*, abs/2404.14219.

Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. 2025. Evolutionary optimization of model merging recipes. *Nat. Mac. Intell.*, 7(2):195–204.

Anthropic. 2024. Claude 3.5 sonnet.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. 2025. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, Wulong Liu, Xinzhi Wang, Defu Lian, Baoqun Yin, Yasheng Wang, and Wu Liu. 2025. Acebench: Who wins the match point in tool usage? *Preprint*, arXiv:2501.12851.

Mayee F. Chen, Nicholas Roberts, Kush Bhatia, Jue Wang, Ce Zhang, Frederic Sala, and Christopher Ré. 2023. Skill-it! A data-driven skills framework for understanding and training language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-flan: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 9354–9366. Association for Computational Linguistics.

Junyan Cheng, Peter Clark, and Kyle Richardson. 2025. Language modeling by language models. *CoRR*, abs/2506.20249.

Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. 2024. SELA: tree-search enhanced LLM agents for automated machine learning. *CoRR*, abs/2410.17238.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, and Wangding Zeng. 2024. Deepseek-v3 technical report. *CoRR*, abs/2412.19437.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2024. How abilities in large language models are affected by supervised fine-tuning data composition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 177–198. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. KTO: model alignment as prospect theoretic optimization. *CoRR*, abs/2402.01306.

Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vladimir Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. 2024.

Arcee's MergeKit: A toolkit for merging large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 477–485, Miami, Florida, US. Association for Computational Linguistics.

Google. 2024. Our next-generation model: Gemini 1.5.

Shih-Cheng Huang, Pin-Zu Li, Yu-Chi Hsu, Kuang-Ming Chen, Yu-Tung Lin, Shih-Kai Hsiao, Richard Tzong-Han Tsai, and Hung-yi Lee. 2024. Chat vector: A simple approach to equip llms with instruction following and model alignment in new languages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 10943–10959. Association for Computational Linguistics.

Gabriel Ilharco, Marco Túlio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Yoichi Ishibashi, Taro Yano, and Masafumi Oyamada. 2024. Can large language models invent algorithms to improve themselves? *CoRR*, abs/2410.15639.

Yoichi Ishibashi, Taro Yano, and Masafumi Oyamada. 2025. Can large language models invent algorithms to improve themselves? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 10332–10363. Association for Computational Linguistics.

Dong-Hwan Jang, Sangdoo Yun, and Dongyoon Han. 2024a. Model stock: All we need is just a few fine-tuned models. In *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part XLIV*, volume 15102 of *Lecture Notes in Computer Science*, pages 207–223. Springer.

Young Kyun Jang, Dat Huynh, Ashish Shah, Wen-Kai Chen, and Ser-Nam Lim. 2024b. Spherical linear interpolation and text-anchoring for zero-shot composed image retrieval. In *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part XIX*, volume 15077 of *Lecture Notes in Computer Science*, pages 239–254. Springer.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. AIDE: ai-driven exploration in the space of code. *CoRR*, abs/2502.13138.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1601–1611. Association for Computational Linguistics.

Muhammad Khalifa, Yi Chern Tan, Arash Ahmadian, Tom Hosking, Honglak Lee, Lu Wang, Ahmet Üstün, Tom Sherborne, and Matthias Gallé. 2024. If you can't use them, recycle them: Optimizing merging at scale mitigates performance tradeoffs. *CoRR*, abs/2412.04144.

Jisu Kim and Juhwan Lee. 2024. Strategic data ordering: Enhancing large language model performance through curriculum learning. *CoRR*, abs/2405.07490.

So Kuroki, Taishi Nakamura, Takuya Akiba, and Yujin Tang. 2024. Agent skill acquisition for large language models via cycleqd. *CoRR*, abs/2410.14735.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguistics*, 7:452–466.

Tian Yu Liu, Aditya Golatkar, and Stefano Soatto. 2024a. Tangent transformers for composition, privacy and removal. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. 2024b. Toolace: Winning the points of LLM function calling. *CoRR*, abs/2409.00920.

Yixiu Liu, Yang Nan, Weixian Xu, Xiangkun Hu, Lyumanshan Ye, Zhen Qin, and Pengfei Liu. 2025a. Alphago moment for model architecture discovery. *CoRR*, abs/2507.18074.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. 2025b. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. *CoRR*, abs/2506.16499.

Chris Lu, Samuel Holt, Claudio Fanconi, Alex J. Chan, Jakob N. Foerster, Mihaela van der Schaar, and Robert T. Lange. 2024. Discovering preference optimization algorithms with and for large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Seyed-Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2025. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Singh Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. 2022. Numglue: A suite of fundamental yet challenging mathematical reasoning tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3505–3523. Association for Computational Linguistics.

Jacob Morrison, Noah A. Smith, Hannaneh Hajishirzi, Pang Wei Koh, Jesse Dodge, and Pradeep Dasigi. 2024. Merge to learn: Efficiently adding skills to language models with model merging. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 15604–15621. Association for Computational Linguistics.

Rémi Munos, Michal Valko, Daniele Calandriello, Mohammad Gheshlaghi Azar, Mark Rowland, Zhaohan Daniel Guo, Yunhao Tang, Matthieu Geist, Thomas Mesnard, Côme Fiegel, Andrea Michi, Marco Selvi, Sertan Girgin, Nikola Momchev, Olivier Bachem, Daniel J. Mankowitz, Doina Precup, and Bilal Piot. 2024. Nash learning from human feedback. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Liangbo Ning, Ziran Liang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Wenqi Fan, Xiaoyong Wei, Shanru Lin, Hui Liu, Philip S. Yu, and Qing Li. 2025. A survey of webagents: Towards next-generation AI agents for web automation with large foundation models. *CoRR*, abs/2503.23350.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Guillermo Ortiz-Jiménez, Alessandro Favero, and Pascal Frossard. 2023. Task arithmetic in the tangent space: Improved editing of pre-trained models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Pulkit Pattnaik, Rishabh Maheshwary, Kelechi Ogueji, Vikas Yadav, and Sathwik Tejaswi Madhusudhan. 2024. Enhancing alignment using curriculum learning & ranked preferences. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 12891–12907. Association for Computational Linguistics.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. 2025. Webrl: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. Tool learning with large language models: a survey. *Frontiers Comput. Sci.*, 19(8):198343.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Morgane Rivière, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozinska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucinska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju-yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjösund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, and Lilly McNealus. 2024. Gemma 2: Improving open language models at a practical size. *CoRR*, abs/2408.00118.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *CoRR*, abs/1904.09728.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Feifan Song, Bowen Yu, Minghao Li, Haiyang Yu, Fei Huang, Yongbin Li, and Houfeng Wang. 2024. Preference ranking optimization for human alignment. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 18990–18998. AAAI Press.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4149–4158. Association for Computational Linguistics.

Guiyao Tie, Zeli Zhao, Dingjie Song, Fuyang Wei, Rong Zhou, Yurou Dai, Wen Yin, Zhejian Yang, Jiangyue Yan, Yao Su, Zhenhan Dai, Yifeng Xie, Yihan Cao, Lichao Sun, Pan Zhou, Lifang He, Hechang Chen, Yu Zhang, Qingsong Wen, Tianming Liu, Neil Zhenqiang Gong, Jiliang Tang, Caiming Xiong, Heng Ji, Philip S. Yu, and Jianfeng Gao. 2025. A survey on post-training of large language models. *CoRR*, abs/2503.06072.

Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2024. Automl-agent: A multi-agent LLM framework for full-pipeline automl. *CoRR*, abs/2410.02958.

Ojasw Upadhyay, Abishek Saravanakumar, and Ayman Ismail. 2025. Synlexlm: Scaling legal llms with synthetic data and curriculum learning. *Preprint*, arXiv:2504.18762.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 23965–23998. PMLR.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, Qi Zhang, and Tao Gui. 2025. The rise and potential of large language model based agents: a survey. *Sci. China Inf. Sci.*, 68(2).

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A. Raffel, and Mohit Bansal. 2023. Ties-merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Xu Yang, Xiao Yang, Shikai Fang, Bowen Xian, Yuante Li, Jian Wang, Minrui Xu, Haoran Pan, Xinpeng Hong, Weiqing Liu, Yelong Shen, Weizhu Chen, and Jiang Bian. 2025. R&d-agent: Automating data-driven AI solution building through llm-powered automated research, development, and evolution. *CoRR*, abs/2505.14738.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Yangyang Zhao, Zhenyu Wang, and Zhenhua Huang. 2021. Automatic curriculum learning with over-repetition penalty for dialogue policy learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 14540–14548. AAAI Press.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

A Cost Analysis

In this section, we analyze the computational costs associated with LaMDAgent to evaluate its practical feasibility for real-world deployment.

GPU costs. When we use Qwen2.5 0.5B-Instruct in the same setting as Experiment2, the total GPU hours consumed were 106.1 hours. Detailed costs of each stages are shown in Table 4.

Table 4: Computational costs for training, merging, and evaluation. We used NVIDIA L40S GPU for all stages.

Stage	**# GPUs**	**Runtime (h)**	**GPU Hours**
SFT	8	0.0908	0.726
TIES	1	0.0541	0.0541
AceBench	2	0.123	0.246
MT-Bench	1	0.103	0.103

API costs. The aggregate API costs totaled 19.9 U.S. dollars. A comprehensive breakdown is provided in Table 5.

Table 5: Token usage and costs for each step. Token counts are shown in thousands.

Usage	**Input** ($\times 10^3$)	**Output** ($\times 10^3$)
Action Type Selection	1844 ($4.61)	87 ($0.879)
Object Selection	1880 ($4.70)	45 ($0.449)
Feedback Generation	3548 ($8.87)	40 ($0.399)
Total	7280 ($18.2)	173 ($1.73)

B Additional Experimental Results

Changing base models. To verify the performance of LaMDAgent with other model families, we conducted experiments using the same configuration as Experiment 2 but changed the base model from Gemma2 2B-Instruct to Qwen2.5 0.5B/1.5B-Instruct. The results are shown in Table 6.

Table 6: Experimental results for changing the base model.

Model	**ACEBench**	**MT-Bench**	**Avg**
Qwen2.5 0.5B-Instruct	0.20	0.535	0.368
w/ LaMDAAgent	0.18	0.680	0.430
Qwen2.5 1.5B-Instruct	0.45	0.715	0.583
w/ LaMDAAgent	0.44	0.741	0.590

Even when changing the base model to Qwen2.5, we consistently achieved improved average performance, demonstrating that the effectiveness of LaMDAgent is independent of base model selection. However, unlike the results with Gemma2 2B-Instruct, tool performance slightly decreased while general task capabilities improved. Examining the training history, MT-Bench scores consistently continued to improve, suggesting that LaMDAgent may have a tendency to continuously enhance performance on specific tasks among multiple tasks.

This indicates the necessity of feedback mechanisms other than the average accuracy across multiple tasks that we adopted, in order to improve targeted characteristics such as tool performance.

Expanding action spaces. To investigate performance when varying the search space of the pipeline, we conducted experiments with the same configuration as Experiment 2 but added hyperparameter exploration as an action. The experimental results are shown in Table 7.

Table 7: Experimental results for varying action spaces.

Action Space	ACEBench	MT-Bench	Avg
Setting 1	0.500	0.810	0.655
Setting 2	0.410	0.795	0.602
Setting 3	0.450	0.793	0.621

- Setting 1: All hyperparameters fixed (identical configuration to Experiment 2)
- Setting 2: SFT learning rate expanded from [1e-6] to [1e-7, 5e-7, 1e-6, 5e-6, 1e-5]
- Setting 3: SFT learning rate expanded from [1e-6] to [5e-7, 1e-6, 5e-6], and TIES weights expanded from [0.5, 0.5] to [[0.5, 0.5], [0.9, 0.1]]

Although expanding the search space should theoretically yield better optimal solution, the search efficiency decreases because poorly performing learning rates may be included in the search range. We believe that within the short iteration limit of 100 iterations used in our configuration, the negative effects of reduced search efficiency outweighed the benefits of expanding the search space.

C Differences from AutoML Agents

When comparing with AutoML agents, such as ML-Master (the state-of-the-art method in MLE-Bench) applied to LLM post-training, the primary difference lies in the optimization space: code space such as Python (ML-Master) versus pre-defined action combinations (LaMDAgent). The detailed comparison with LaMDAgent arising from this fundamental difference is shown in Table 8

Table 8: Comparison between ML-Master and LaMDAgent across multiple aspects. ✓ indicates favorable characteristics, × indicates unfavorable characteristics, and ∼ indicates intermediate characteristics.

Aspect	ML-Master	LaMDAgent
Guardrails	×	✓
Controllability	×	✓
Multiple Environments	∼	✓
Cost Efficiency	×	✓
Manual Prompting Cost	×	✓
Action Space Size	✓	×
Manual Coding Cost	✓	×

1. **Guardrails:** ML-Master directly executes LLM-generated code, requiring guardrails such as sandboxing to prevent deletion of important files or environmental changes. In contrast, LaMDAgent requires no guardrails as it does not perform file operations or environment setup.

2. **Controllability:** When specifying desired models, training methods, or datasets for LLM training, LaMDAgent can limit exploration to combinations of these specified components. However, ML-Master relies on prompt-based constraints, which may lead to exploration of unintended spaces.

3. **Multiple Environments:** When different types of model improvement actions require different environments (e.g., different versions of transformers), complex pipelines like "environment setup → action → environment setup → action..." are difficult with current ML-Master implementations. LaMDAgent handles this easily due to the loose coupling of action executions.

4. **Manual Prompting Cost:** ML-Master requires exhaustive specification of environment-related information such as server details and paths necessary for coding. LaMDAgent allows execution with arbitrary settings simply by writing configurations in a YAML file.

5. **Cost Efficiency:** In ML-Master, code is executed from scratch for each run, which can result in repeated training under identical settings and thus waste computational resources. In contrast, LaMDAgent improves efficiency by reusing previously trained components. Furthermore, this work uniquely investigates whether LLMs can autonomously achieve computational cost reduction through scaling, without explicit human guidance.

6. **Action Space Size:** While ML-Master optimizes the entire codebase, LaMDAgent can be interpreted as being limited to optimizing predefined combinations. Consequently, ML-Master operates within a broader optimization space.

7. **Manual Coding Cost:** Manual coding cost is nearly zero for ML-Master, while LaMDAgent requires pre-written code for each action. However, once LLM fine-tuning code is written, it can be reused across various situations.

Based on the above analysis, LaMDAgent is more suitable for developers who already possess model training code, have defined their optimization search space, and seek to conduct controlled experiments—particularly LLM developers. Conversely, AutoML approaches are more appropriate for practitioners with limited LLM training experience who encounter difficulties in implementing training code.

D Templates

Figure 8, 9, and 10 are prompt templates for action type selection, object selection, and memory generation in our proposed LaMDAgent, respectively.

Figure 11 shows an example of configs for LaMDAgent.

Prompt template to select an action type

```
You are a developer of Large Language Models (LLMs) who tests model improvement strategies based on a
     given hypothesis. You are provided with Self-Reflections obtained from analyzing the result of a
     previous trial conducted for model improvement. Based on the Self-Reflections, select one action
     type from the Action Type List to create a more performant model. Analyze the Self-Reflections to
     identify the most promising action type, and provide the number of the selected action type at the
      end. If the Self-Reflections are not provided, please select randomly.

Self-Reflections:
<reflection>

Action List:
<action_types>

Selected Action Type NUMBER:
```

Figure 8: Prompt template to select an action type.

Prompt template to select objects

```
You are a developer of Large Language Models (LLMs). Your task is to determine a configuration for
     creating an LLM.
The configuration consists of multiple object types, and for each object type, you must select one
     object from a set of candidate objects.
To aid in your selection, you are provided with introspective analysis based on past LLM configurations
      and their outcomes.
Please output the selected objects in the order of the object types displayed, using comma separation
     and enclosed in [[ ]], e.g., [[1, 0, 2]] at the end of the output. If the Self-Reflections are not
      provided, please select randomly and think of a combination that has not been tried in the past
     trials. Also, the k-th model at step n is named in the format 0--n--k. Since such models also have
      promising potential, please include them in the search scope.

Self-Reflections:
<reflection>

Object Candidates:
<object_cands>

Selected Object NUMBERs:
```

Figure 9: Prompt template to select objects.

Prompt template to update memory

```
You are a developer of Large Language Models (LLMs) that can improve models based on self reflections.
     You will be given results and memories of the previous improving trials. The results consist of
     actions and scores, where the scores are out of 1 point. And also, You will be provided with newly
      aquired trials. In a few sentences, update your memories based on the previous trials, memoeries,
      and new results.

# Previous Results
<previous results>

# Previous Memories Aquired from Previous Trials
<previous memories>

# Newly aquired Results
<new results>

Updated Memory:
```

Figure 10: Prompt template to update memory.

An example config of our proposed method


```
{
    "seed": 42,
    "total_timesteps": 100,
    "controller": "LaMDAgent_gpt",
    "controller_model": "gpt-4o-2024-11-20",
    "objects": {
        "base_models": ["models/gemma-2-2b"],
        "models": ["models/gemma-2-2b--gsm8k_1k", "models/gemma-2-2b--commonsense_qa_1k", "models/gemma
    -2-2b--trivia_qa_1k_w_context","models/gemma-2-2b"],
        "sft_dataset": ["data/sft_formatted/gsm8k_1k", "data/sft_formatted/commonsense_qa_1k", "data/
    sft_formatted/trivia_qa_1k_w_context", "data/sft_formatted/gsm1k_cqa1k_tqa1k"],
        "sft_lr": [0.000001],
        "ties_weights": [[0.5, 0.5]],
        "ties_density": [0.5],
    },
    "action_types": {
        "sft": ["models", "sft_dataset", "sft_lr"],
        "ties_merging": ["base_models", "models", "models", "ties_weights", "ties_density"]
    },
    "eval_tasks": [["gsm8k", "acc"], ["commonsenseqa", "acc"], ["trivia_qa_w_context", "acc"]],
    "score_aggregation": "mean"
}
```


Figure 11: An example config of our proposed method.