NAT: Enhancing Agent Tuning with Negative Samples

Renxi Wang[1,2] **Xudong Han**[1,2] **Yixuan Zhang**[1,2]
Timothy Baldwin[1,2,3] **Haonan Li**[1,2]
[1]LibrAI [2]MBZUAI [3]The University of Melbourne
{renxi.wang,xudong.han,yixuan.zhang,timothy.baldwin,haonan.li}@mbzuai.ac.ae

Abstract

Interaction trajectories between agents and environments have proven effective in tuning LLMs into task-specific agents. However, constructing these trajectories, especially successful trajectories, is often computationally and time intensive due to the relatively low success rates of even the most advanced LLMs, such as GPT-4 and Claude. Additionally, common training paradigms like supervised fine-tuning (SFT) and reinforcement learning (RL) not only require large volumes of data but also have specific demands regarding the trajectories used. For instance, existing SFT approaches typically utilize only positive examples, limiting their efficiency in low-resource scenarios. To address this, we introduce Negative-Aware Training (NAT), a straightforward yet effective method that leverages both successful and failed trajectories for fine-tuning, maximizing the utility of limited resources. Experimental results demonstrate that NAT consistently surpasses existing methods, including SFT, DPO, and PPO, across various tasks[1].

1 Introduction

An agent is a model that can interact with environments, make decisions, and achieve predefined goals (Wooldridge, 1999). Recent work has built powerful LLMs such as GPT-4 (OpenAI, 2023), prompting them as the core of an agent system to process information and make decisions with few-shot examples (Gravitas, 2024; Yoheinakajima, 2024). However, these agents depend on closed-source, paid APIs, raising concerns about cost and latency, and face limitations as LLMs are not specifically designed for tasks like action generation or tool use, with few-shot prompting offering only limited learning support (Chen et al., 2023; Wang et al., 2024).

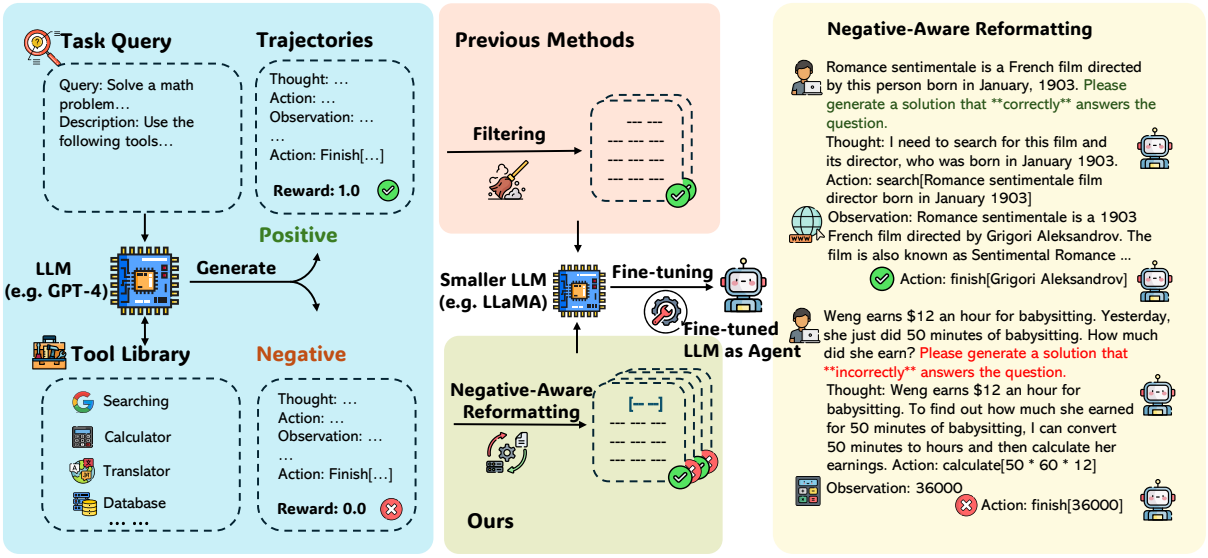
Tuning LLMs on interaction trajectories between agents and environments has proven effective in transforming them into task-specific agents (Chen et al., 2023; Zeng et al., 2023; Yin et al., 2023; Qiao et al., 2024; Wang et al., 2024). However, building these trajectories, especially successful ones, is resource-intensive and time-consuming. Even advanced models like GPT-4 (OpenAI, 2023) and Claude (Anthropic, 2023), struggle with tasks requiring complex planning, reasoning, or tool use, resulting in low success rates. This scarcity of successful trajectories turns agent-tuning into a low-resource scenario, where the lack of high-quality data becomes a significant bottleneck.

Traditional approaches to fine-tuning, such as supervised fine-tuning (SFT) and reinforcement learning (RL), heavily depend on a large volume of successful trajectories to be effective. Supervised fine-tuning, in particular, tends to focus on positive examples—those where the task is completed successfully—discarding failed trajectories, or negative examples, as irrelevant or uninformative. Reinforcement learning-based methods like Proximal Policy Optimization (PPO) attempt to address this issue by exploring a broader range of trajectories, but they still face challenges when dealing with complex, low-success-rate tasks (Trung et al., 2024; Song et al., 2024c). Our preliminary observations show that over 80% of data generated during typical interaction-based data collection is discarded due to failure, leading to resource wastage and undertrained agents despite multiple rounds of data generation.

To address these inefficiencies, we propose a novel approach to fine-tuning agents that leverages both successful and failed trajectories more effectively. Our method, called Negative-Aware Training (NAT), builds on the insight that negative examples contain valuable information that can complement positive examples when used appropriately. Rather than discarding negative data,

[1]Code and data are available at: https://github.com/Reason-Wang/NAT

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 7385–7398



(a) (b) (c)

Figure 1: An overview of previous methods and our NAT paradigm. (a) Data collection, where interactions between LLMs and environments (tools) are collected. (b) Data processing, where previous methods simply filter out negative examples, while we reformat trajectories by adding prompts to task queries based on whether they are positive or negative. (c) An example of reformated positive and negative trajectories. We omit the system prompts here.

NAT uses both positive and negative trajectories during fine-tuning, providing the model with richer information about what works and what doesn't in a given task. Through simple but effective modifications, NAT enables LLM-based agents to better generalize from limited resources by learning from a more diverse range of examples.

Our experimental results demonstrate that NAT significantly outperforms traditional methods, including SFT, DPO, and PPO, across a range of tasks. NAT is particularly effective in low-resource scenarios, where it delivers substantial improvements in mathematical reasoning and multi-hop question answering tasks, with gains of 8.74 points and 6.05 points, respectively, over traditional fine-tuning approaches. These results confirm our hypothesis that failed trajectories are valuable for agent tuning, and show that our approach, NAT, is an effective and efficient way of utilizing failed trajectories.

Our contributions can be summarized as follows:

- We demonstrate the value of negative trajectories in low-resource scenarios and introduce a negative-aware training paradigm, allowing LLM-based trained agents to effectively learn from both positive and negative examples.
- We validate the broad applicability and effectiveness of learning from negative examples, and show that NAT enables models to acquire information akin to positive examples across various tasks and prompting strategies.

2 Related Work

2.1 Fine-tuning LLMs as Agents

Previous work on language agents has taken a powerful LLM as the core of the agent system without fine-tuning (Sumers et al., 2023; Wu et al., 2023; Ruan et al., 2023; Zhao et al., 2023; Wang et al., 2024). However, LLMs are optimized to generate natural language. To make them capable of using tools and making decisions, current work typically collects trajectories generated by GPT-3.5/4, then uses these trajectories to fine-tune a smaller LLM (Chen et al., 2024; Zhang et al., 2024; Zhou et al., 2024). Zeng et al. (2023) collect trajectories generated by GPT-4 on AgentBench (Song et al., 2024b) tasks, and only keep samples that receive the best rewards. Chen et al. (2023) collect trajectories on question answering tasks and fine-tune models with samples that correctly answer the question. Liu et al. (2024) propose a memory-enhanced agent framework and a complex filtering mechanism to collect fine-tuning datasets. Qiao et al. (2024) divide an agent into sub-agents with different functions. They then synthesize trajectories for the respective agents. However, they still

only use samples with the best rewards. A simple ablation study was conducted by Zeng et al. (2023). However, none of this work has investigated the effectiveness of negative samples in detail. Although not directly comparable, in Table 1, we provide the results of these methods and ours on several benchmarks for reference.

2.2 Learning from Negative Samples

Learning from negative results can be divided into prompt-based and training-based methods. Prompt-based methods enable LLMs to summarize experiences from previous mistakes without updating parameters (Madaan et al., 2023; Shinn et al., 2023; Zhao et al., 2023). The success of these methods relies on the quality of the evaluator used to analyze the trajectories. The performance of training-based methods is less predictable since model weights are updated, and less work has been done on this. Li et al. (2023) propose a two-stage training paradigm to capture knowledge from negative samples. However, their method focuses on Chain-of-Thought prompts and is complex since multiple models are fine-tuned. Liu et al. (2023a) propose CoH that combines positive and negative responses together. However, this needs pair-wise responses for a sample, often unavailable in low-resource scenarios like agent-tuning. CodeRanker (Inala et al., 2022) is a fault-aware model tained to predict the rich attributes from executing programs. It achieves superior performance in Coding tasks compared to other traditional neural ranker. Some work uses fine-grained optimization method for agent leraning. For example, IPR (Xiong et al., 2024) optimizes the agent with reward in each step using Direct Preference Optimization, enabling better agent learning and yielding superior agent capabilities.

Contemporaneous to our work, AgentBank (Song et al., 2024a) proposes the largest agent trajectories tuning data, comprising 16 tasks and convering diverse agent skills. To mitigate the failure problem during the data collection, it proposes a pipeline to mitigate the difficulty bias in trajectories. Tong et al. (2024) adds different prefixes to positive and negative reasoning rationales during fine-tuning. Their work mainly focuses on Chain-of-Thought (Wei et al., 2022) reasoning tasks, while our work focuses on multi-round agent prompting that integrates calling tools. Besides, we did more analysis on data quantity, quality, working mechanism, and possible applications. Some other works train agents with exploration trajectories and reinforcement learning (Yuan et al., 2024; Song et al., 2024b). Our work mainly focuses on the SFT stage, which can be integrated seamlessly into these works yet offers a better initialization.

Model	GSM8K	SVAMP	HotpotQA
AutoAct-7B	–	–	29.2
AgentLM-7B	24.6	–	22.3
Lumos-O-7B	**50.5**	**65.5**	24.9
Lumos-I-7B	47.1	63.6	29.4
NAT-7B	49.1	64.4	**29.8**
CodeLlama-13B	36.1	60.0	–
AgentLM-13B	32.4	–	**29.6**
NAT-13B	**53.8**	**70.6**	**29.6**

Table 1: Comparison with methods from other papers. We report the best results reported in the corresponding papers.

3 Negative-Aware Training (NAT)

In this section, we introduce our agent framework and the full pipeline of Negative-Aware Training (NAT). We outline the process from task completion and data collection to the integration of negative samples, which enhances learning efficiency in low-resource settings by leveraging both positive and negative data. Following this, we describe the fine-tuning process and inference phase. Figure 1 contrasts previous approaches with our NAT paradigm, demonstrating how NAT leads to more effective agent tuning with constrained data resources.

3.1 Agent Framework

As shown in Figure 1, in our agent framework, the process of task completion is delineated as follows. First, the LLM is provided with a system prompt that outlines (a) the specific task to be addressed (for instance, "`solve a mathematical problem`"), (b) the tools that are permissible for task execution, and (c) the expected action space and output format (for example, *finish[N]* signifies that *N* is the final answer). We do not provide system prompts in Figure 1, for simplicity. Second, a query instance is introduced. We prompt the model to answer the query in the `ReAct` (Yao et al., 2023) format, which consists of reasoning texts (referred to as "thoughts") and "actions". Finally, during the interaction phase, the system executes the LLM-generated actions using the predefined tools, returns the resulting observations back to the LLM, and prompts for subsequent actions until the *finish*

Dataset	# Times	1	2	3
GSM8K	# Positives	3,233	4,577	5,100
	Incremental	44%	+17%	+7%
HotpotQA	# Positives	1,620	1,950	2,120
	Incremental	41%	+8%	+4%
StrategyQA	# Positives	871	1,006	1,070
	Incremental	68%	+10%	+5%

Table 2: Number and incremental ratio of positive samples from three seed datasets after generation for 1, 2, and 3 times. Incremental ratio is calculated as the percentage of newly generated samples to the total samples. The number of newly generated positive samples decreases as the generation count increases.

action of the task is generated, or the interaction rounds exceed a pre-defined threshold. Naturally, the task-solving process yields interaction trajectories between the LLM and the environment (i.e., tools in our framework).

Tool Deployment For math tasks, we design a calculator implemented by SymPy (Meurer et al., 2017), which takes a math expression as input and outputs the result. For the two question-answering tasks, we deploy a search tool with the Serper API.[2] It takes a search query as input and returns the Google search results. To get more relevant snippets, we further deploy a search results re-ranker using MPNet (Song et al., 2020) and DPR (Karpukhin et al., 2020).

3.2 Insights from Agent Tuning Data Collection

For each task, we obtain the initial questions and corresponding ground truth answers as seed data. We then use GPT-3.5 to generate trajectories three times,[3] each with different temperatures (0.2, 0.5, and 0.7). This allows us to gather a diverse range of positive and negative samples. By comparing predicted answers and ground truth answers, we can label each trajectory as positive or negative.

Table 2 presents the data collection results for three key tasks used in this study, which will be introduced in detail later. From the table, we observe that the agent framework captures a substantial portion of positive samples during the first round. However, the success rate for previously failed instances steadily decreases with each additional round of data collection. For example, in the HotpotQA dataset, only 8% and 4% of new positive trajectories were generated in the second and third rounds, respectively. Based on this trend, we estimate that subsequent rounds would yield 2%, 1%, and so on, resulting in a final success rate likely not exceeding 60%, even with an infinite budget.

From this data collection process, we made two key observations: (1) Relying solely on positive trajectories leads to inefficiency and resource wastage as the data collection progresses. (2) For a large proportion of data instances, it is challenging to collect pairwise (positive and negative) samples, which complicates the application of RL-based methods. Our experiments in Section 4 and Section 5 further validate these findings.

3.3 Leveraging Negative Samples for Agent Tuning

To address data inefficiency, we propose that negative samples can be valuable in low-resource scenarios. To reduce the dependency on pairwise data samples, we introduce supervised fine-tuning with negative-aware reformatting, which explicitly informs the model of incorrect responses, as outlined below.

Negative-Aware Reformatting Differentiating positive samples from negative samples during the agent tuning process aids in teaching the model to discern between successful and unsuccessful outcomes. We append a string suffix to tell the model whether the training sample is positive or negative. For positive samples, we append "`Please generate a solution that **correctly** answers the question.`" For negative samples, we append "`Please generate a solution that **incorrectly** answers the question.`" Unless explicitly stated, we use this setting in experiments. We also experimented with other reformatting strategies.[4]

Fine-tuning and Inference We use the reformatted trajectories to fine-tune LLMs. The loss is computed only on the part of the text generated by the LLM, which is similar to fine-tuning a chat model (Zheng et al., 2023). During inference, we prompt the fine-tuned agent using the prompt for positive examples only.

[2] https://serper.dev/

[3] We use GPT-3.5-1106 version. Although GPT-4 has the potential to produce even higher quality data, we opted for GPT-3.5 due to cost considerations.

[4] The actual prompts that we use in our experiments are slightly more complex than those provided here.

Mothod	Employ Negative	Require Paired	Extra Compute	GSM8K	ASDiv	SVAMP	MultiArith	Average
SFT	✗	✓	✓	29.49	58.13	44.50	71.81	50.98
CoH (Liu et al., 2023a)	✓	✗	✓	30.10	56.65	41.40	72.32	50.12
PPO (Trung et al., 2024)	✓	✓	✗	28.81	58.38	46.60	70.64	51.11
DPO (Song et al., 2024c)	✓	✗	✗	29.80	56.55	42.90	69.30	49.64
NUT	✓	✓	✓	34.12	59.71	49.90	75.68	54.85
NAT (ours)	✓	✓	✓	**35.17**	**60.01**	**50.60**	**77.68**	**55.87**

Table 3: Overall results for math tasks with a comparison between baseline models and ours from three perspectives: (1) **Negative** – the method can employ negative samples for agent tuning; (2) **Paired** – the method require paired positive and negative samples for agent tuning; and (3) **Compute** – the method need extra computational resources (e.g., an extra reward is needed for PPO). The best results are **bolded**.

Method	HotpotQA		StrategyQA
	EM	F1	
SFT	27.80	36.45	55.40
CoH	28.60	39.53	-
PPO	28.20	36.47	60.00
DPO	26.40	34.83	-
NUT	28.80	40.59	62.40
NAT	**29.60**	**42.50**	**65.80**

Table 4: Overall results on question answering tasks. For HotpotQA, we measure the performance using exact match and f1 score. For StrategyQA, we use accuracy. Best results are **bolded**.

4 NAT for Math Reasoning

4.1 Experimental Setup

Datasets We conduct experiments on mathematical reasoning tasks. We use GSM8k (Cobbe et al., 2021) as seed data to collect trajectories and test the performance on four math testing sets, including GSM8k, ASDiv (Miao et al., 2020), SVAMP (Patel et al., 2021), and MultiArith (Roy and Roth, 2015). The details of these datasets are introduced in Appendix B.

Baselines We compare **NAT** with finetuning or exploration-based baselines. **SFT** directly finetunes the model with positive trajectories. This is the method that previous work (Zeng et al., 2023; Chen et al., 2023; Qiao et al., 2024; Liu et al., 2024) has employed. We also implemented CoH, DPO, PPO and NUT for comparison. In particular, **CoH** concatenate pairwise positive and negative responses to form one example to finetune the model. **PPO** and **DPO** explore solution trajectories and optimize the model based on trajectory rewards. **NUT** (negative-unaware training) finetunes the model directly with positive and negative trajectories. For finetuning-based methods (**SFT**, **CoH**, **NUT** and **NAT**), responses are included for training. While for exploration-based methods (**PPO** and **DPO**), only queries and answers are used. Except **SFT**, other methods can employed negative examples during training. Among them, **CoH** and **DPO** require pairwise data to finetune the model. **DPO** and **PPO** require extra computation, through inference during exploration and reinforcement learning optimization.

Fine-tuning Setup We conduct experiments on LLaMA-2-Chat 7B models (Touvron et al., 2023). All the models are fine-tuned for 3 epochs with a batch size of 64. We use a cosine scheduler with 3% of total steps as the warm-up. The maximum learning rate is set to 2×10^{-5}. We train the model with 4×A100 GPUs with DeepSpeed ZeRO 3 stage (Rajbhandari et al., 2019).

4.2 Results

Table 3 presents the overall results of the math tasks, which demonstrates that: (1) **NUT** shows incorporating negative examples can improve model performance. (2) Models with negative-aware training (**NAT**) not only outperform the corresponding model trained only on positive examples (**SFT**), but also beat the same model trained by directly incorporating negative examples (**NUT**); and (3) Methods that integrate exploration (**PPO** and **DPO**) only show a small improvement or even decrease performance compared to directly finetuning using negative examples (**NUT** and **NAT**).

It is worth noting that previous work (Zeng et al., 2023) has shown that including negative examples harms model performance. We believe this does not contradict our findings: as we discuss in Sections 6.1 and 6.2, performance is determined by

both the quantity and quality of the negative data.

5 NAT for Question Answering

5.1 Experimental Setup

Datasets We collect trajectories and test the performance on HotpotQA (Yang et al., 2018), a multi-hop question answering dataset, and StrategyQA, a binary question answering dataset where answers are *yes* or *no* (Geva et al., 2021), respectively. See details in Appendix B.

Baselines and Finetuning We compare NAT with the same baselines shown in Section 4.1 and take the same hyper-parameters for finetuning.

5.2 Results

Table 4 shows the results on HotpotQA and StrategyQA, which demonstrate similar results to math tasks. Directly incorporating negative samples (**NUT**) improves the performance in low-resource agent-tuning. **NAT** outperforms both finetuning and exploration-based methods on exact match (EM) and f1 scores. Again, **PPO** and **DPO** show their inability in QA tasks with very limited exploration resources, highlighting the value and effectiveness of **NAT**.

6 Analysis

Tables 3 and 4 showcase the capability of LLMs to learn from negative examples and the superiority of **NAT** compared to other methods in low-resource scenarios. In this section, we delve into various factors that could influence the effectiveness of **NAT**. Specifically, we seek to address the following questions:

- Section 6.1: How much negative samples should be used in **NAT**?
- Section 6.2: Are all negative samples beneficial?
- Section 6.3: What does the model learn from negative samples?
- Section 6.4: Can **NAT** be adapted to handle fine-grained labels of negative samples?

Since only the math task contains enough data for our experiments, the analysis is done on the math task.

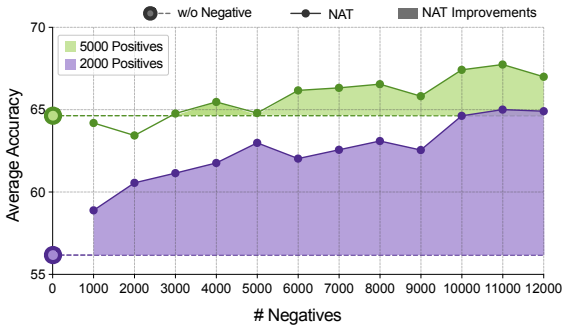


Figure 2: Performance of LLaMA-2-Chat with 2,000 (green) and 5,000 (purple) positive samples and a variable number of negative samples. The number of negative samples is changed from 0 to 12,000. The shaded area represents improvements using **NAT**.

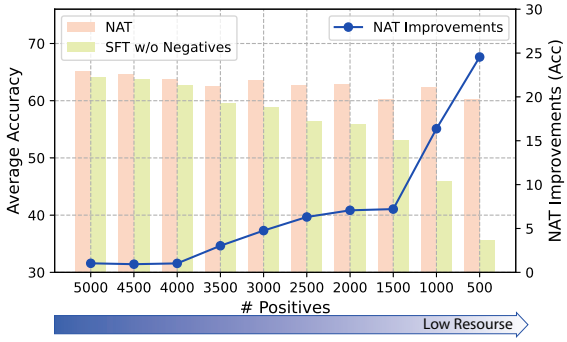


Figure 3: Performance for a fixed number of negative samples (10k) and decreasing number of positive samples. A smaller number of positive samples means a lower-resource scenario. The bars show performance of **NAT** and **SFT** without negative samples. The line shows the improvements using **NAT** compared to **SFT**.

6.1 Impact of Training Sample Quantity

Our initial analysis focuses on the influence of negative sample quantity. We maintain a constant number of positive samples at 2k and 5k, while adjusting the negative samples from 0 to 12k. The results, depicted in Figure 2, illustrate the relationship between the quantity of negative data and the average performance on math tasks. We observe a performance enhancement with an increase in negative data, which plateaus when the volume of negative samples is about 11k in both cases. Due to data availability, we did not experiment with more negatives.

Based on insights from Table 3 and Figure 2, we think that the ideal ratio of negative samples is not fixed. Instead, it is influenced by the number of positive samples, as the improvements are larger for fewer positive samples.

Data	GSM8K	ASDiv	SVAMP	MArith	Avg.
2K positive samples					
Vanilla	35.63	60.55	47.40	80.03	55.90
NAT-bad	32.98	58.72	47.60	71.64	52.74
NAT-good	**46.93**	**66.93**	**60.80**	**83.89**	**64.64**
5K positive samples					
Vanilla	45.87	68.12	58.80	83.89	64.17
NAT-bad	38.59	62.28	52.50	78.52	57.97
NAT-good	**49.05**	**68.66**	**64.40**	**87.58**	**67.42**

Table 5: LLaMA-2 7B model results trained with different quality negative data. We use 10k negative samples and experiment with 2k or 5k positive samples.

We hypothesize that the marginal improvement from negative samples becomes more significant as the availability of positive samples decreases in low-resource settings. To validate this, we maintain a constant number of negative samples while varying the quantity of positive samples from 5,000 to 500. As depicted in Figure 3, there is an increased benefit from the negative samples as the count of positive samples decreases. This showcases the value of **NAT** for low-resource tasks where there is limited positive data. For the second point, we investigate the effects of negative data quality in Section 6.2.

6.2 Impact of Data Quality

We sourced negative data from various models to investigate the impact of negative data quality in NAT. Specifically, we consider the data from GPT-3.5 to be high-quality examples. In contrast, we generated 10k negative examples using a fine-tuned LLaMA-2-7B (Touvron et al., 2023) model to represent low-quality data. For experiments, we paired 2k positive examples with 10k negative examples. The outcomes presented in Table 5 underscore the critical role of data quality in **NAT**. In the 2k positive sample setting, the improvement is -3.16 for low quality compared to $+8.74$ for high-quality negative examples. Similarly, in the 5k positive sample setting, the improvements are -6.20 and $+3.25$, respectively.

6.3 What does the Model Learn with NAT?

Learning Reasoning while Preventing Errors The learnable parts of trajectories are thoughts and actions, where thoughts involve reasoning on the current situation and planning for what to do next. Actions involve selecting which tool to call and the input to that tool. We analyze the trajectories of the GSM8K (Cobbe et al., 2021) test set, generated by LLaMA-2-7B trained with positive examples (**SFT**), **NUT**, and **NAT** respectively. Table 6 shows the accuracy, action error (the percentage of incorrectly calling a tool), and average turns (the average number of steps needed to solve a question). Incorporating negative examples introduces more action errors, resulting in fine-tuned models with more errors compared to **SFT**. However, after incorporating negative examples, both the accuracy of **NUT** and **NAT** increase. This indicates that negative examples mainly work by teaching models with better "thoughts" (i.e. reasoning and planning). Compared to **NUT**, **NAT** achieves significantly fewer action errors and, therefore, better accuracy. This demonstrates that our method works by providing a better trade-off between better "thoughts" and more action errors.

Negative Samples Play a Similar Role as Positive Samples To further explore whether models learn from negative trajectories in the same manner as they learn from positive trajectories, we randomly sample 100 successful trajectories from the training set (as a dev set) and measure the perplexity of models trained with 500 positive examples (not overlapping with the dev set) and varying numbers of negative examples. Figure 4 shows the change in perplexity as the number of negative data increases. The perplexity decreases as more negative data is included, which indicates the model learns to fit successful trajectories with knowledge from failed trajectories. However, this curve seems to be horizontal at the end, and there is still a large gap between the curve with 2,500 positives, which shows that some properties or knowledge from successful trajectories can never be learned from failed trajectories.

6.4 Scale up NAT with Fine-grained Labels

Different negative trajectories contain different degrees of errors. Intuitively, this information also helps models to learn. Therefore, we further propose fine-grained **NAT**, which divides negative trajectories into different groups based on their quality. During training, different groups will be reformatted with different prompts. For HotpotQA, in addition to the EM score, each trajectory has an f1 score, measuring the overlap between the predicted and gold answers. We take this as a fine-grained measurement of data quality, where a trajectory with a higher f1 score has better quality. In this way,

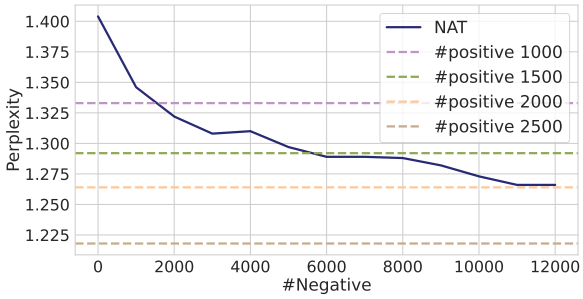


Figure 4: Perplexity for the model trained with 500 positive samples and differing numbers of negative samples. The three dashed lines are perplexity computed on models tuned with differing numbers of positive trajectories (without negatives).

Strategy	Acc	Action Error	#Avg. Turns
SFT	35.63	3.58%	3.12
NUT	44.43	10.47%	3.92
NAT	46.93	7.90%	3.71

Table 6: Accuracy, action error rate, and number of average turns for models with different training strategies. For SFT, the action error in training data is 4.01%. For NUT and NAT, it is 15.33%.

we can differentiate trajectories based on quality by assigning different prompts. For example, the trajectory is prepended "`almost wrong`" if its f1 score is smaller than 0.1, and another trajectory is "`mostly correct`" with an f1 score of 0.9. We denote this **NAT** with different prompting strategies as **NAT-k**, where **k** represents how many classes we divide the negative data into based on quality.

Fine-grained NAT learns more from negative samples For **NAT-2**, we take trajectories with f1 scores equal to 1.0 as positive and assign different prompts for trajectories with f1 scores less than 0.4 and with f1 scores greater than 0.4 less than 1.0. It can be seen from Figure 5 that both **NAT-1** and **NAT-2** outperforms vanilla **SFT**. **NAT-2** shows a better scaling capability, consistently improves the performance when using more negative examples, while **NAT-1** starts to drop with 1,000 negative examples. **NAT-2** shows the best performance among all settings, show its effectiveness and potential for large volume of negative data.

7 Adapting NAT for Chain-of-Thought (CoT)

So far, we have conducted all experiments on agent scenarios with the `ReAct` (Yao et al., 2023) prompting strategy. In this section, we conduct preliminary experiments to explore whether **NAT** works well with Chain-of-Thought (CoT) prompting (Wei et al., 2022). The key difference is that the agent takes an action and receives an observation from the environment iteratively, while CoT generates reasoning steps without taking actions or receiving observations.

We use GPT-3.5-0125 to generate CoT reasoning steps with three in-context learning (Brown et al., 2020) examples on the GSM8k dataset. We then train the model with **NAT**. Table 7 shows the results with CoT prompting. **NAT** achieves a 6.68% improvement compared to no negative data training (**SFT**). **NAT** is still about 1% higher compared to directly including negative samples (**NUT**). The results demonstrate that **NAT** is also applicable and effective for CoT training, showing its broad applicability.

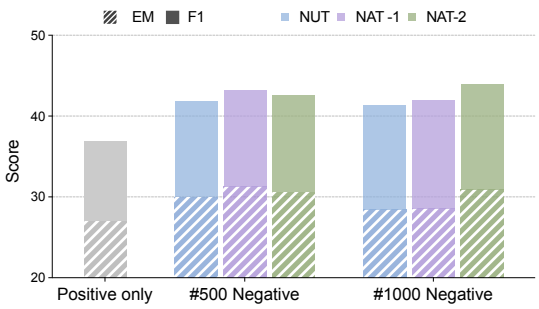


Figure 5: Performance of LLaMA-2-7B on HotpotQA with 500 positive examples and varying numbers of negative examples.

Strategy	GSM8K	ASDiv	SVAMP	MArith	Avg
Vanilla	29.04	55.26	45.60	80.87	52.69
NUT	33.50	**61.69**	52.20	**86.41**	58.45
NAT	**36.24**	61.10	**53.90**	86.24	**59.37**

Table 7: LLaMA-2-7B model CoT results fine-tuned using 2k positive samples and 1.6k negative samples.

8 Conclusion

We introduced **NAT** (Negative-Aware Training), a simple yet effective approach that incorporates negative trajectories into the agent fine-tuning process. Our experiments on mathematical reasoning and question-answering tasks highlight the superior performance of **NAT** over existing methods for agent tuning, particularly in low-resource scenarios. We also conducted extensive analyses to uncover the factors contributing to the success of **NAT**, providing insights into its effectiveness.

Limitations

Despite the promising results demonstrated in our experiments, there are several limitations to our approach. First, like other agent-tuning methods, our approach relies on the availability of ground truth labels, which restricts its applicability in scenarios where such labels are scarce or unavailable. Second, while our experiments indicate that high-quality negative samples contribute significantly to the success of NAT, we were unable to establish a clear boundary or metric for distinguishing between high- and low-quality negative samples. We leave it a further work. Lastly, due to time and budget constraints, we did not fine-tune or evaluate our method on more diverse and larger models (e.g., GPT-3.5), which may limit the generalizability of our findings to more powerful models.

References

Anthropic. 2023. Claude.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *ArXiv*, abs/2310.05915.

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. *ArXiv*, abs/2403.12881.

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. *arXiv preprint arXiv:2110.14168*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Significant Gravitas. 2024. Autogpt. `https://github.com/Significant-Gravitas/AutoGPT`.

Jeevana Priya Inala, Chenglong Wang, Mei Yang, Andres Codas, Mark Encarnación, Shuvendu Lahiri, Madanlal Musuvathi, and Jianfeng Gao. 2022. Fault-aware neural code rankers. *Advances in Neural Information Processing Systems*, 35:13419–13432.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Bin Sun, Xinglin Wang, Heda Wang, and Kan Li. 2023. Turning dust into gold: Distilling complex reasoning capabilities from llms by leveraging negative data. *AAAI 2024*.

Hao Liu, Carmelo Sferrazza, and Pieter Abbeel. 2023a. Chain of hindsight aligns language models with feedback. *arXiv preprint arXiv:2302.02676*.

Na Liu, Liangyu Chen, Xiaoyu Tian, Wei Zou, Kaijiang Chen, and Ming Cui. 2024. From llm to conversational agent: A memory enhanced architecture with fine-tuning of large language models. *ArXiv*, abs/2401.02777.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023b. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing

english math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024. Autoact: Automatic agent learning from scratch via self-planning. *ArXiv*, abs/2401.05268.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2019. Zero: Memory optimizations toward training trillion parameter models. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1743–1752. The Association for Computational Linguistics.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2023. Identifying the risks of lm agents with an lm-emulated sandbox. *ArXiv*, abs/2309.15817.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *International Conference on Learning Representations*.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. `https://api.semanticscholar.org/CorpusID:258833055`.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *arXiv preprint arXiv:2004.09297*.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024a. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2124–2141, Miami, Florida, USA. Association for Computational Linguistics.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024b. Trial and error: Exploration-based trajectory optimization for llm agents.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024c. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600, Bangkok, Thailand. Association for Computational Linguistics.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.

Yongqi Tong, Dawei Li, Sizhe Wang, Yujia Wang, Fei Teng, and Jingbo Shang. 2024. Can llms learn from previous mistakes? investigating llms' errors to boost for reasoning.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. ReFT: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7601–7614, Bangkok, Thailand. Association for Computational Linguistics.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2024. Toolgen: Unified tool retrieval and calling via generation.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed H Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Michael Wooldridge. 1999. Intelligent agents. *Multiagent systems: A modern approach to distributed artificial intelligence*, 1:27–73.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! LLM agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1556–1572, Miami, Florida, USA. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Raghavi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2023. Lumos: Learning agents with unified data, modular design, and open-source llms. *ArXiv*, abs/2311.05657.

Yoheinakajima. 2024. Babyagi. `https://github.com/Significant-Gravitas/AutoGPT`.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *ArXiv*, abs/2310.12823.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, Tulika Awalgaonkar, Juan Carlos Niebles, Silvio Savarese, Shelby Heinecke, Huan Wang, and Caiming Xiong. 2024. Agentohana: Design unified data and training pipeline for effective agent learning. *ArXiv*, abs/2402.15506.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Y. Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. *ArXiv*, abs/2308.10144.

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.

Qinhao Zhou, Zihan Zhang, Xiang Xiang, Ke Wang, Yuchuan Wu, and Yongbin Li. 2024. Enhancing the general agent capabilities of low-parameter llms through tuning and multi-branch reasoning. *ArXiv*, abs/2403.19962.

A Example

Figure 6 shows examples trajectories generated by GPT-3.5. The first turn of each trajectory is the system prompt. Figure 7 shows example inference results of models trained with different settings for the same query.

B Datasets

For mathematical reasoning tasks, we use a dataset of approximately 7k instances from the GSM8K training set as initial seed data, and generate three trajectories with GPT-3.5, as mentioned in Section 3. This process results in a collection of around 9k positive examples and 12k negative examples. Among the positive examples, 5k are unique, indicating that despite multiple attempts, GPT-3.5 fails to solve 2k out of the 7k original questions.

For our experiments, we incorporate 5k unique positive examples from GSM8K to emulate all available positive examples having been generated by GPT-3.5. Additionally, we created a simulated limited dataset using the 2k positive examples generated by ChatGPT. In both scenarios, we include 10k negative examples.

We evaluate different models and training strategies on four test datasets: GSM8K (Cobbe et al., 2021), a high-quality school math word problem dataset containing 1,319 examples (test set), each requiring 2–8 steps to solve; ASDiv (Miao et al., 2020), a math word problem dataset that contains 2,023 examples with diverse language patterns and problem types. SVAMP (Patel et al., 2021), a challenge set of math word problems with 1k examples based on perturbing existing datasets (Miao et al., 2020; Koncel-Kedziorski et al., 2016). MultiArith (Roy and Roth, 2015), a multi-step arithmetic problem dataset with 596 examples.

For question-answering tasks, we collected trajectories based on HotpotQA (Yang et al., 2018) and StrategyQA (Geva et al., 2021). HotpotQA is a Wikipedia-based question-answering dataset where each question requires several steps of reasoning with supporting passages. We use 4k examples from the training set to generate trajectories. StrategyQA is also a multi-step question-answering dataset but the reasoning steps are implicit. The answer to its question is either yes or no. It consists of 2,780 examples, of which 1k is the training set.

Similar to math tasks, we generate three QA trajectories. As discussed in Section 6, the quality of negative samples is important for the effectiveness

Model	Setting	GSM8K	ASDiv	SVAMP	MultiArith	AVG.
Llama-2-7B-Chat	SFT	29.49	58.13	44.50	71.81	50.98
	NAT	35.17	60.01	50.60	77.68	55.87
Llama-2-13B-Chat	SFT	28.35	59.66	50.60	75.34	53.49
	NAT	41.47	63.22	60.90	84.06	62.41
Llama-3-8B-Instruct	SFT	18.12	52.79	45.10	54.19	42.55
	NAT	33.28	63.91	58.90	82.72	59.70
Qwen2.5-3B-Instruct	SFT	57.62	82.60	75.90	95.30	77.86
	NAT	60.58	85.81	79.10	93.79	79.82
Deepseek-LLM-7B-Base	SFT	29.04	53.44	36.70	69.80	47.25
	NAT	36.47	62.53	50.40	78.18	56.90

Table 8: Results for different LLMs trained with SFT and NAT.

Setting	I1-Inst.	I1-Tool	I1-Cat.	I2-Inst.	I2-Cat.	I3-Inst.	AVG.
SFT	34.97	37.76	45.64	34.91	26.77	24.59	34.13
NAT	36.40	38.71	43.57	29.40	35.62	39.89	37.27

Table 9: Results of tool learning for Llama-2-7B-Chat. I1 represents single tool setting, I2 and I3 represent multi-tool setting. Tools in I2 come from the same collection while in I3 come from different collections.

Positive	Negative	Average
Correct	Incorrect	63.55
Incorrect	Correct	63.33
Good	Bad	63.91
A	B	63.15
Random string 1	Random string 2	64.04

Table 10: Results for models trained on prompts with and without interpretability. Strings in the Positive/Negative column represent prompts (prefixes or suffixes) we use for positive/negative trajectories.

of NAT. For HotpotQA, we filter out trajectories that do not give an answer within a certain number of turns or with a zero f1 score. Finally, we obtain 2k unique positive samples and 2k negative samples. However, we find that 2k examples are enough for performance to saturate and that adding more negative samples causes a performance drop. Therefore, we set the number of HotpotQA positive examples to 500 in our experiments.

C Selection of Added Prompts

It has variously been shown that prompts are vital for LLM performance (Brown et al., 2020; Liu et al., 2023b; Sclar et al., 2024). Here, we explore the interpretability of added prompts. More specifically, does the content of the prompt enable LLMs to learn differently from successful and failed trajectories, or simply differentiate these trajectories? We propose two sets of prompts. One set is prompts with interpretability, such as having the model generate a correct or incorrect trajectory. Another set is prompts without interpretability. For example, different letters can be added as prefixes for queries. Table 10 shows the results of models trained with interpretable and uninterpretable prompts. Different prompts do not show a large difference in performance, indicating that the performance boost of NAT comes from simply differentiating positive and negative data.

D More Experiments

NAT for Different LLMs We experiment NAT for different LLMs to validate its effectiveness. As shown in Table 8, NAT improves performance across diverse models and sizes. This also demonstrates the broad applicability for NAT method.

NAT for Tool Usage We have done experiments on math and question-answering (QA) tasks and evaluated the method on 4 math test sets. However, we also experimented NAT for Tool Learning, a more real-world scenario where LLMs use different tools to solve common tasks. We use datasets from ToolBench (Qin et al., 2024) for training and evaluation. The metric is pass rate, representing the percentage that the evaluator believes the answer generated by LLMs can solve the task. We first use GPT-4 to annotate whether the answers of training trajectories are passed. We take those annotated with pass as positive trajectories and others as negative trajectories. The results are shown in Table 9. Even in a more real-world and complex scenario, NAT still improves the performance, surpassing SFT with positive trajectories only.

Math

Solve a math problem with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be two types:
(1) calculate[expression], which calculates the expression and returns the answer.
(2) finish[answer], which returns the answer and finishes the task.

Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?

Thought: To find the total number of clips Natalia sold in April and May, I need to calculate the number of clips sold in May and then add it to the number sold in April. Action: calculate[48 / 2]

Observation: 24

Thought: Now, I need to add the number of clips sold in April to the number sold in May to find the total number of clips sold altogether.\nAction: calculate[48 + 24]

Observation: 72

Thought: The total number of clips Natalia sold altogether in April and May is 72. Action: finish[72]

HotpotQA

Solve a question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be two types:
(1) search[query], which searches a query on google and returns a short snippet containing the answer. Note that sometimes the snippet does not contain the answer, and some alternative search might be needed.
(2) finish[answer], which returns the answer and finishes the task.

Forestville Commonwealth is an example of which 19th century socialist philosophy?

Thought: To answer this question, I need to search for information about Forestville Commonwealth and its association with 19th-century socialist philosophy. Action: search[Forestville Commonwealth 19th century socialist philosophy]

Observation: Owenism is the utopian socialist philosophy of the 19th century social ... Forestville Commonwealth, Lapham's Mills, Coxsackie, Greene County, New York (1826 ... Communitarian Socialism in America: 1663-1829 (Philadelphia, 1950), 62-63. 5. Robert Owen's teachings were the basis for all of the Owenite societies, ...

Thought: The observation suggests that Forestville Commonwealth is associated with Owenism, a utopian socialist philosophy of the 19th century. Action: finish[Owenism]

StrategyQA

Solve a question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be two types:
(1) search[query], which searches a query on google and returns a short snippet containing the answer. Note that sometimes the snippet does not contain the answer, and some alternative search might be needed.
(2) finish[answer], which returns the answer and finishes the task. Remember you should answer Yes or No.

If you were on a diet, would you have to skip lunch at McDonald's?

Thought: Whether someone on a diet would have to skip lunch at McDonald's depends on their specific dietary restrictions and goals. McDonald's offers a variety of menu items, including salads and other healthier options, so it is possible to make a suitable choice while still adhering to a diet plan. Action: search[healthy options for lunch at McDonald's]

Observation: 1. Side Salad: McDonald's offers a side salad that is typically low in calories and fat. 2. Grilled Chicken Sandwich: Opting for a grilled ...

Thought: The observation provides several low-calorie and healthier options available at McDonald's, including salads and grilled chicken sandwiches. This indicates that someone on a diet could indeed find suitable lunch options at McDonald's without having to skip lunch altogether. Action: finish[No]

Figure 6: Example trajectories of GSM8k, HotpotQA, and StrategyQA generated by GPT-3.5.

Figure 7: Inference results of models trained with different settings, where Vanilla model gets wrong because of reasoning error, NUT model gets wrong because of action error, and NAT gets correct.