

Learning From Failure: Integrating Negative Examples when Fine-tuning Large Language Models as Agents

Renxi Wang^{1,2}

Haonan Li^{1,2}

Xudong Han^{1,2}

Yixuan Zhang^{1,2}

Timothy Baldwin^{1,2,3}

¹LibrAI

²MBZUAI

³The University of Melbourne

{renxi.wang, haonan.li, xudong.han, yixuan.zhang, timothy.baldwin}@mbzui.ac.ae

Abstract

Large language models (LLMs) have achieved success in acting as agents, which interact with environments through tools such as search engines. However, LLMs are optimized for language generation instead of tool use during training or alignment, limiting their effectiveness as agents. To resolve this problem, previous work has first collected interaction trajectories between LLMs and environments, using only trajectories that successfully finished the task to fine-tune smaller models, making fine-tuning data scarce and acquiring it both difficult and costly. Discarding failed trajectories also leads to significant wastage of data and resources and limits the possible optimization paths during fine-tuning. In this paper, we argue that unsuccessful trajectories offer valuable insights, and LLMs can learn from these trajectories through appropriate quality control and fine-tuning strategies. By simply adding a prefix or suffix that tells the model whether to generate a successful trajectory during training, we improve model performance by a large margin on mathematical reasoning, multi-hop question answering, and strategic question answering tasks. We further analyze the inference results and find that our method provides a better trade-off between valuable information and errors in unsuccessful trajectories. To our knowledge, we are the first to demonstrate the value of negative trajectories and their application in agent-tuning scenarios. Our findings offer guidance for developing better agent-tuning methods and low-resource data usage techniques.¹

1 Introduction

An agent is a model that has the ability to interact with environments, make decisions, and achieve predefined goals (Wooldridge, 1999). Early work used rule-based or template-based systems to complete tasks in narrow and specialized domains

(Green Jr et al., 1961; Weizenbaum, 1966). Recent work has built off powerful LLMs such as GPT-4 (Achiam et al., 2023), using them as the core of an agent system to process information and make decisions (Gravitas, 2024; Yoheinakajima, 2024). This line of work has resulted in agent systems that are able to perform much more complex and general tasks.

However, these agents generally rely on closed-source LLMs through paid APIs, raising concerns about cost, latency, and reproducibility. Additionally, existing LLMs were not developed for agent use cases (e.g., generating actions or calling tools), and few-shot prompting offers only limited learning support (Chen et al., 2023).

Subsequent work has explored fine-tuning LLMs as agents, typically in three stages: data collection, fine-tuning, and inference (Chen et al., 2023; Zeng et al., 2023; Yin et al., 2023; Qiao et al., 2024). At the data collection stage, a powerful LLM such as GPT-4 is employed to interact with the environment, and the LLM-generated outputs and environment observations are collected as trajectories. In the fine-tuning stage, smaller models are fine-tuned using only successful trajectories. The fine-tuned models then serve as the agent’s core during inference, demonstrating enhanced tool-using and decision-making capabilities, sometimes even surpassing the performance of the original LLM (Yin et al., 2023).

To ensure the agent is being optimized appropriately, previous work has simply discarded trajectories that do not successfully complete the task (i.e. negative examples), using only successful trajectories (i.e. positive examples) in the fine-tuning stage (Zeng et al., 2023; Chen et al., 2023; Qiao et al., 2024). However, in tasks demanding intricate planning, reasoning, or tool usage, the volume of discarded negative samples can exceed 60%, leading to substantial data and computational resource wastage.

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

In this paper, we explore two key questions: (1) Can LLMs learn from these negative examples through fine-tuning? and (2) How can we optimize the use of negative examples to enhance agent performance? To address the first question, we fine-tune LLMs with a mix of positive and negative examples, and observe that incorporating negative examples generally yields benefits. For the second question, we introduce a negative-aware training (NAT) paradigm that explicitly tells the model to differentiate between correct and incorrect interactions by adding prefixes or suffixes. Our experiments demonstrate that NAT outperforms traditional methods by solely using positive examples or naively combining positive and negative ones, enabling better fine-tuning for low-resource data. In addition, we conduct extensive experiments to analyze the learned agents’ behavior after fine-tuning with negative examples. Our contributions can be summarized as follows:

- We demonstrate the value of negative trajectories and introduce a negative-aware training paradigm, allowing LLM-based trained agents to effectively learn from both positive and negative examples. To the best of our knowledge, we are the first to utilize negative examples in agent training.
- 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.
- We find NAT works by providing a better trade-off between useful information and errors in negative examples.

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). 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 (Zeng et al., 2023; Chen et al., 2023; Qiao et al., 2024; Chen et al., 2024; Zhang et al., 2024; Zhou et al., 2024). Zeng et al. (2023) collect trajectories generated by GPT-4 on AgentBench (Liu et al., 2023b) 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 done 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 Results

Learning from negative results can be divided into prompt-based and fine-tuning-based methods. Prompt-based methods enable LLMs to summarize experiences from previous mistakes without updating parameters. Madaan et al. (2023) use LLMs to first generate an output and then refine the output iteratively, while Shinn et al. (2023) employ an evaluator to provide external feedback. Zhao et al. (2023) let the agent compare successful and unsuccessful trajectories, and extract insights based on comparison. The success of these methods relies on the quality of the evaluator used to analyze the trajectories. The performance of fine-tuning-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. Our work focuses on fine-tuning LLMs as agents and is much simpler and more effective.

3 Negative-Aware Training

We first illustrate our motivation for using NAT and then describe our agent framework. We then introduce the whole pipeline of NAT, including data collection, negative-aware reformatting (which is the core part of our method that differentiates it from others), fine-tuning, and inference. Figure 1 outlines previous methods and our NAT paradigm.

3.1 Motivation

Our idea is motivated by two considerations. First, humans learn from mistakes and failures. Failure is

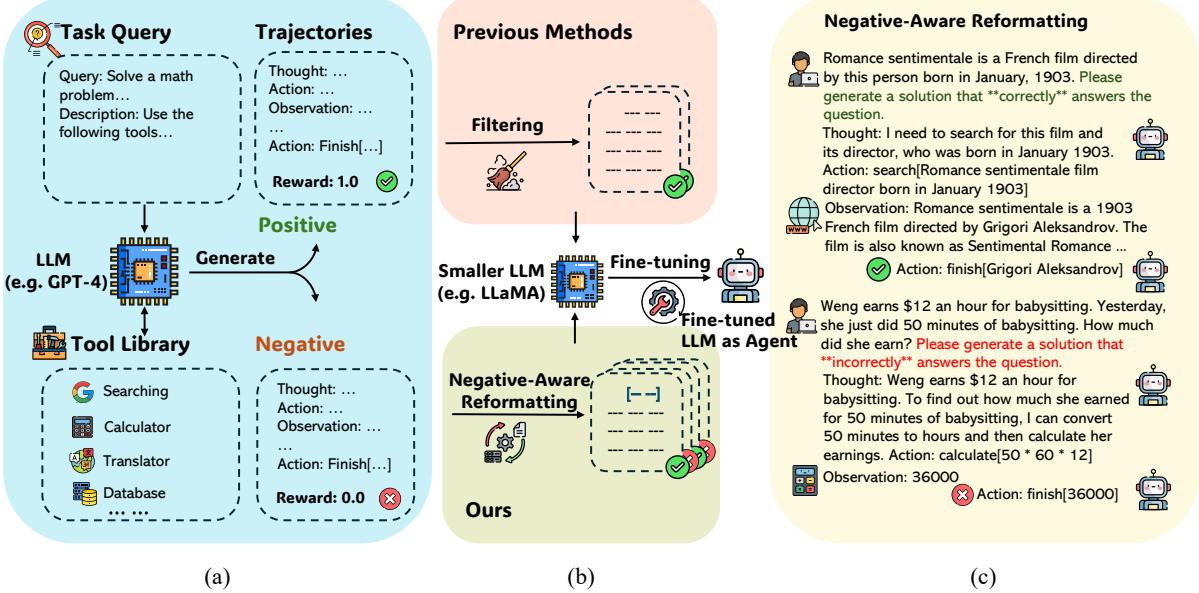


Figure 1: An overview of previous methods and our NAT paradigm. (a) Data collection, where interactions between LLMs and environments (tools) are collected. (b) Data processing, where previous methods simply filter out negative examples, while we reformat trajectories by adding prompts to task queries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

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	<u>29.4</u>
NAT-7B	<u>49.1</u>	<u>64.4</u>	29.8
CodeLlama-13B	<u>36.1</u>	<u>60.0</u>	—
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.

often seen as a stepping stone to success, as it offers insights into what does not work and highlights areas that require change or development. We believe that powerful LLMs can also learn these valuable lessons from unsuccessful trajectories.

Second, it is generally hard for humans to compare and learn from experiences when they do not know which experience is successful. Fine-tuning approaches generally treat all examples equally, and LLMs may learn unwanted errors from negative trajectories if negative examples are incorporated directly. Therefore, we add a prefix or suffix to the query to differentiate positive and negative examples, explicitly telling the model whether the following trajectories they learn are correct.

3.2 Agent Framework

Prompting Strategy As shown in Figure 1, in our agent framework, the process of task resolution 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* 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).

Tools 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 design a search tool with the Serper² API. It takes a search query as input and returns the Google search results. We further re-rank the search results using MPNet (Song et al., 2020) and DPR (Karpukhin et al., 2020).

3.3 The Whole Pipeline

We introduce the whole pipeline of our negative-aware training paradigm here, where negative-aware reformatting is the core part of the paradigm that enables better agent tuning.

Data Collection For each task, we obtain the initial questions and corresponding ground truth answers as seed data. We then use GPT-3.5³ to generate trajectories three times, 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.

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 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/>

³We 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.

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

4 Experiments

4.1 Experimental Setup

Datasets We conduct experiments on mathematical reasoning, multi-hop question answering, and strategic question answering tasks. For math, we collect trajectories using GSM8k (Cobbe et al., 2021) as seed data and test the performance on GSM8k, ASDiv (Miao et al., 2020), SVAMP (Patel et al., 2021), and MultiArith (Roy and Roth, 2015). For question answering, we collect trajectories and test the performance on HotpotQA (Yang et al., 2018) and StrategyQA (Geva et al., 2021), respectively. More details are provided in Appendix B.

Baselines We primarily compare NAT with two baselines. The Vanilla setting uses positive examples only to fine-tune LLMs. This is what previous work (Zeng et al., 2023; Chen et al., 2023; Qiao et al., 2024; Liu et al., 2024) has done. The second setting includes negative examples without adding any prefix or suffix, which we call Negative-Unaware Training (NUT).

Fine-tuning Setup We conduct experiments on LLaMA-2-Chat 7B and 13B models (Touvron et al., 2023). All the models are fine-tuned for 2 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 5×10^{-5} . We train the model with $4 \times$ A100 GPUs with DeepSpeed ZeRO 3 stage (Rajbhandari et al., 2019).

4.2 Results

Math Table 2 presents the overall results of the math tasks, from which we observe: (1) 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 (Vanilla), but also beat the same model trained by directly incorporating negative examples (NUT); and (3) The improvement of NAT is more substantial when there are fewer positive examples or the model is smaller. Specifically, NAT achieves an 8.74% improvement when using a 7B model with 2k positive examples, and a 0.52% improvement when using a 13B model with 5k positive examples. This highlights the value of NAT in data-scarce scenarios, which is common for agent tuning.

It is worth noting that previous work (Zeng et al., 2023) has shown that including negative examples harms model performance. We believe this does

Model	# Positive	Strategy	GSM8K	ASDiv	SVAMP	MultiArith	Average
LLama-2-7B	2k	Vanilla	35.63	60.55	47.40	80.03	55.90
		NUT	<u>44.43</u>	<u>65.69</u>	<u>60.40</u>	<u>83.05</u>	<u>63.39</u>
		NAT	46.93	66.93	60.80	83.89	64.64
LLama-2-7B	5k	Vanilla	45.87	<u>68.12</u>	58.80	<u>83.89</u>	64.17
		NUT	<u>47.54</u>	67.03	<u>63.50</u>	81.71	<u>64.95</u>
		NAT	49.05	68.66	64.40	87.58	67.42
LLama-2-13B	2k	Vanilla	44.43	66.49	65.40	84.40	65.18
		NUT	<u>49.43</u>	<u>67.72</u>	<u>67.60</u>	81.37	<u>66.53</u>
		NAT	50.64	67.92	68.50	83.89	67.74
LLama-2-13B	5k	Vanilla	54.21	71.28	68.30	<u>89.26</u>	<u>70.76</u>
		NUT	51.40	70.34	<u>68.60</u>	86.07	69.10
		NAT	<u>53.75</u>	<u>70.49</u>	70.60	90.27	71.28

Table 2: Overall results for math tasks. Each block is a setting with a specific model and number of positive examples. The best results are **bolded** and second best results are underlined

Strategy	7B		13B	
	EM	F1	EM	F1
Vanilla	27.44	36.41	27.04	36.95
NUT	28.04	40.96	28.24	42.47
NAT	28.80	41.37	28.44	42.45
NAT-2	29.76	42.51	29.60	43.29

Table 3: Results of LLaMA-2-7B and 13B on HotpotQA. All results are reported as the mean score of 5 runs. For the 7B model, we report results using 1,500 negative samples; for the 13B models, we use 2k negative samples. NAT-2 means we divide negative samples into 2 classes based on quality. We discuss this in Section 6.1.

Strategy	7B	13B
Vanilla	55.40	53.40
NUT	62.40	61.80
NAT	65.80	64.60

Table 4: Results of LLaMA-2-7B and 13B on StrategyQA with 1000 positive samples and 500 negative samples.

not contradict our findings: as we discuss in Sections 5.1 and 5.2, performance is determined by both the quantity and quality of the negative data.

Question Answering Tables 3 and 4 show the results on HotpotQA and StrategyQA. Here, NAT-2 is a variant of NAT, where we divide negative data into two classes and use different prompts for each, as detailed in Section 6.1. On HotpotQA, NAT-2 improves performance by more than 2% in EM and 6% in f1 score compared to no negative samples. Compared with NUT, NAT is still about 1% better on EM and f1. On StrategyQA, NAT achieves

more than 8% and about 3% improvements compared to no negative samples and NUT, respectively. This suggests that our method is also effective for question-answering tasks.

5 Analysis

Tables 2 to 4 showcase the capability of LLMs to learn from negative examples when fine-tuned to function as agents. Here, we delve into various factors that could influence the effectiveness of negative-aware training. Specifically, we seek to address the following questions: (1) Given a fixed number of positive examples, how much negative data should be used? (2) What insights does the model gain from negative trajectories? (3) Are all types of negative examples beneficial? and (4) What factors contribute to negative-aware training (NAT) outperforming negative-unaware training (NUT)? Since only the math task contains enough data for our experiments, the analysis is done on the math task.

5.1 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 average math task performance. 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.

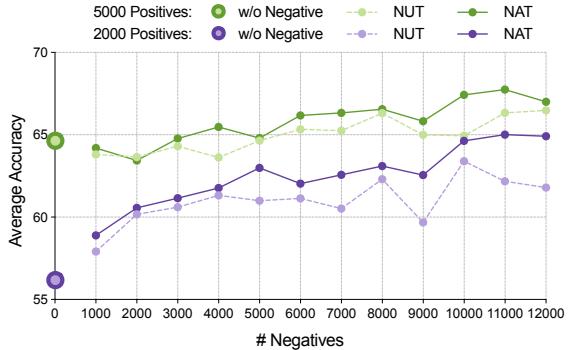


Figure 2: Performance of LLaMA-2-7B for a fixed number of positive samples and variable number of negative samples.

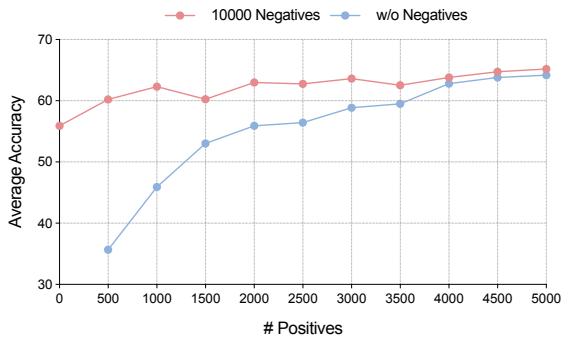


Figure 3: Performance for a fixed number of negative samples (10k) and variable number of positive samples.

Based on insights from Table 2 and Figure 2, we hypothesize that the ideal ratio of negative samples is not fixed. Instead, it is influenced by two main factors: (1) the number of positive samples, as the improvements are larger for fewer positive samples; and (2) the intrinsic quality of the negative samples.

Regarding the first point, we hypothesize that the marginal utility of negative samples diminishes as the quantity of positive samples increases. To test this point, we maintain a constant number of negative samples while varying the quantity of positive samples from 0 to 5k. As depicted in Figure 3, there is a diminishing return on the performance added by negative samples as the count of positive samples rises. For the second point, we investigate the effects of negative data quality in Section 5.2.

5.2 Data Quality Matters

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 as high-quality examples. In contrast, we generated 10k negative examples using a fine-tuned LLaMA-2-7B (Touvron et al., 2023) model to rep-

Data	GSM8K	ASDiv	SVAMP	MAirth	Avg.
2K positive samples					
Vanilla	35.63	60.55	47.40	80.03	55.90
NAT-low	32.98	58.72	47.60	71.64	52.74
NAT-high	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-low	38.59	62.28	52.50	78.52	57.97
NAT-high	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.

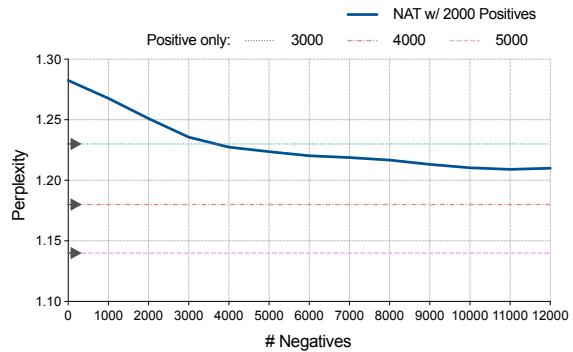


Figure 4: Perplexity for the model trained with 2k positive samples and differing numbers of negative samples. The three dashed lines are perplexity computed on models tuned with differing numbers of positive trajectories (without negatives).

resent 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.

5.3 What does the Model Learn with NAT?

Learning Reasoning Rather Than Acting The learnable part in trajectories are thoughts and actions, where thoughts are the reasoning on the current situation and planning for what to do next. Actions are which tool to call and the input to the tool. We analyze the trajectories of the GSM8K (Cobbe et al., 2021) test set generated by LLaMA-2-7B trained with positive examples (Vanilla), NUT, and NAT respectively. Table 6 shows the accuracy, action error (the percent of incorrectly calling a tool), and average turns (the average number of

Strategy	Accuracy	Action Error	#Avg. Turns
Vanilla	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 Vanilla, the action error in training data is 4.01%. For NUT and NAT, it is 15.33%.

steps needed to solve a question). Incorporating negative examples also introduces more action errors, which can result in fine-tuned models with more action errors compared to Vanilla. 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 2k 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 4k and 5k positives, which shows that some properties or knowledge from successful trajectories can never be learned from failed trajectories.

5.4 Selection of Added Prompts

It has variously been shown that prompts are vital for LLM performance (Brown et al., 2020; Liu et al., 2023a; 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

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 7: 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.

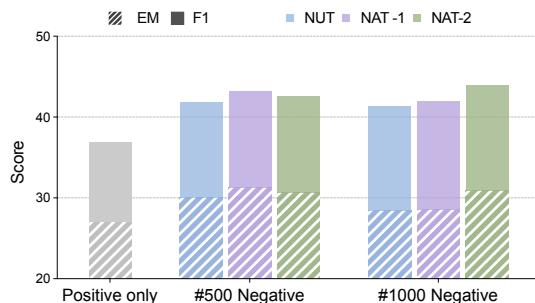


Figure 5: Performance of LLaMA-2-7B on HotpotQA with 500 positive examples and varying numbers of negative examples.

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 7 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.

6 Applications

In this section, we explore some other applications with our NAT method, showcasing its potential and broad applicability.

6.1 Fine-grained NAT

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

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

measurement of data quality, where a trajectory with a higher f1 score has better quality. In this way, 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 Table 3 that the NAT-2 consistently outperforms NAT in all settings, indicating that negative samples associated with finer-grained info are more informative. We investigate how the performance changes with differing numbers of negative samples in Figure 5. When adding negative samples, the performance increases by a margin, consistent with Table 3. However, NAT achieves the best performance with only 500 negative samples, and its performance decreases when adding more negative samples. NAT-2, on the other hand, achieves the best performance with 1k negative samples, consistent with our hypothesis that fine-grained NAT is more beneficial to model training.

6.2 Chain-of-Thought Prompting

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 8 shows the results with CoT prompting. NAT achieves a 6.68% improvement compared to no negative data training (Vanilla). 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.

7 Conclusion

In this paper, we first demonstrated that LLMs can learn from failures when fine-tuned as an agent. On the basis of this finding, we propose NAT, a simple and effective method for integrating failed trajectories in fine-tuning agents. We conduct experiments on math and question-answering tasks, and show the superior performance of our method compared to training directly with positive or negative trajectories across tasks and model sizes.

We are the first to demonstrate the value of negative trajectories in fine-tuning LLMs as agents. Our analysis finds that the quality of negative data is the key to the success of NAT, and models learn similar knowledge to that of positive data (which is much more expensive to attain). NAT is superior because it better utilizes valuable information while restraining learning errors from negative examples.

Negative-aware training is designed to be both agent-agnostic and reasoning strategy-agnostic, making it compatible with various agent strategies, including self-refinement (Madaan et al., 2023), reflection (Shinn et al., 2023), and other agent-tuning frameworks. At the end of this paper, we demonstrated the effectiveness of NAT on Chain-of-Thought (CoT) reasoning in mathematical tasks. Moving forward, we aim to assess the applicability and effectiveness of NAT across a broader spectrum of agent frameworks, strategies, and tasks.

8 Limitations

Although we have conducted extensive experiments to illustrate the effectiveness of our method, there are still limitations. First, similar to previous work in agent tuning, our approach requires the ground truth label of the data, which limits its application. Second, we do not experiment with fine-tuning our method on more diverse and powerful models (e.g. GPT-3.5) due to time and budget limits.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- 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>.
- Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. 1961. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224.
- 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*.
- Na Liu, Liangyu Chen, Xiaoyu Tian, Wei Zou, Kai-jiang 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. 2023a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Yuxian Gu, Hangliang Ding, Kai Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Shengqi Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023b. **Agentbench: Evaluating llms as agents**. *ArXiv*, abs/2308.03688.
- 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 Pąrocki, 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.
- Arkil Patel, Satwik Bhattacharya, 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.

- 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*.
- Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. *Cognitive architectures for language agents*. *arXiv preprint arXiv:2309.02427*.
- 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*.
- 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*.
- Joseph Weizenbaum. 1966. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.
- Michael Wooldridge. 1999. Intelligent agents. *Multilateral 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*.
- 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>.
- 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 Awalgonaonkar, 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, Zuohan 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 3.3. 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 5, the quality of negative samples is important for the effectiveness 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 the following experiments.

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.

Vanilla



Each solid 10-foot section of a redwood tree weighs 400 pounds. Termites ate 30% of this redwood's wood. If the redwood is 200 feet tall, how much does it weigh? Please generate a solution path.



Thought: To find the total weight of the redwood tree, I need to calculate the weight of the wood that was not eaten by the termites and then add the weight of the wood that was eaten. Action: calculate[$200 * 10 * 400 - (200 * 10 * 400 * 0.3)$]



Observation: 560000

Action: finish[560000]



NUT



Each solid 10-foot section of a redwood tree weighs 400 pounds. Termites ate 30% of this redwood's wood. If the redwood is 200 feet tall, how much does it weigh? Please generate a solution path.



Thought: To find the weight of the redwood tree, I need to calculate the weight of the wood that was eaten by the termites and then subtract that from the total weight of the wood. Action: calculate[30% of 400 pounds per foot]



Observation: Error: Sympify of expression 'could not parse '30% of 400 pounds per foot" failed, because of exception being raised: SyntaxError: invalid syntax (<string>, line 1)

NAT



Each solid 10-foot section of a redwood tree weighs 400 pounds. Termites ate 30% of this redwood's wood. If the redwood is 200 feet tall, how much does it weigh? Please generate a **correct** solution path.



Thought: Thought: To find the weight of the redwood tree, I need to calculate the weight of one 10-foot section and then multiply it by the total number of sections in the tree. First, I need to calculate the weight of one 10-foot section after the termites ate 30% of the wood. Action: calculate[$400 * 0.7$]



Observation: 280



Thought: Now, I need to calculate the total weight of the redwood tree by multiplying the weight of one 10-foot section by the total number of sections in the tree. Action: calculate[$280 * 200 / 10$]



Observation: 5600



Thought: The total weight of the redwood tree is 5600 pounds. I have finished the task. Action: finish[5600]

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.