Reinforcement Learning Training for Dynamic Context Management in Mathematical Reasoning

Batu El[1], **Mehmet Hamza Erol**[2], **Hannah Park-Kaufmann**[1]

[1]Institute for Computational and Mathematical Engineering, Stanford University
[2]Department of Computer Science, Stanford University

Extended Abstract

Language models (LMs) face significant limitations in reasoning tasks when their context becomes cluttered with irrelevant information. This can lead to chaotic reasoning traces and degraded performance. Prior work has focused on memory-augmented architectures or prompting-based context controlling mechanisms, but these approaches do not enable the model itself with the ability to manage its own context. Our work addresses this gap by teaching models to explicitly clean their context through learned behavior, enabling better control over their reasoning flow.

We propose a reinforcement learning framework where the model learns to issue a special `<clean>` token that signals context cleaning and resets reasoning. This token provides a lightweight yet effective mechanism for explicit backtracking. For this, we firstly curate an augmented supervised fine-tuning dataset based on expert traces that include both productive and unproductive reasoning paths. The model is fine-tuned to learn the core of these reasoning behaviors and when to clean its context. Then, we further optimize the model using a modified REINFORCE Leave-One-Out (RLOO) objective under a specialized context management pipeline that takes the context cleaning into account effectively. This training setup distinguishes between single-shot answers and multi-phase responses with optional context cleaning.

Using the Qwen-2.5 0.5B model and the Countdown dataset, we build a generation pipeline with a one-shot cleaning interface. The model first generates a response (y_0). If it issues a `<clean>` tag, a fresh response (y_1) is sampled in a cleared context. Our results show that the extension improves reasoning performance across the board. On hard problems (those involving multiplication/division), accuracy improves from 13.61% to 36.94%. Clean usage becomes more efficient after RL training—used less frequently, but with higher average reward—indicating that the model learns not just how but also *when* to clean.

Our results show that language models can learn to manage their own context by issuing a `<clean>` token, leading to improved reasoning and more effective backtracking—without relying on external memory systems. The method is simple and interpretable, yet achieves substantial gains in accuracy, particularly on harder problems. While currently limited to single-use cleaning and no memory retrieval, it lays the groundwork for more advanced context control. Future work will explore multi-step cleaning and memory-aware strategies to support more complex reasoning. This represents a step toward models that can monitor and repair their own reasoning processes.

Preprint. Under review.

Abstract

We introduce a reinforcement learning framework that teaches language models to manage their own context by issuing a special <clean> token, enabling explicit backtracking and improved reasoning without external memory systems. Unlike prior approaches that rely on memory-augmented architectures or prompt-level context control, our method trains models to learn when and how to clean their context through supervised fine-tuning on curated expert traces, followed by a modified REINFORCE Leave-One-Out (RLOO) optimization. Applied to the Qwen-2.5 0.5B model on the Countdown dataset, our approach enables a one-shot cleaning interface where models selectively reset their reasoning. This leads to performance improvements, especially on harder problems, with accuracy on such cases rising from 13.61% to 36.94%. For the other cases, this makes the performance on-par with the default methods. The model also becomes more judicious in its use of the <clean> token post-training, signaling an emergent ability to self-monitor and manage its reasoning flow.

1 Introduction

1.1 Problem & Motivation

Advancements in language models, particularly in reasoning models [OpenAI et al., 2024, DeepSeek-AI et al., 2025], have led to remarkable performance gains in mathematical reasoning tasks. However, these reasoning models still have two major inherent limitations. Firstly, they often have a limited context length, which restricts the extent to which a language model can reason about a problem. Secondly, their ability to effectively use the information within their context window deteriorates as the context length increases [Hosseini et al., 2024, An et al., 2024]. In particular, models have been observed to forget information contained in the middle sections of the context [Liu et al., 2023]. Notably, the performance of the language models degrades significantly with irrelevant information present in the context [Shi et al., 2023, Huang et al., 2025, Chatziveroglou et al., 2025]. Recently, Gandhi et al. [2025] have observed that four cognitive behaviors (verification, backtracking, subgoal setting, and backward chaining) significantly improve a model's ability to learn to reason more effectively during RL training. However, the current literature largely overlooks the potential accumulation of redundant information within the model's context during reasoning, which may degrade the quality of its outputs. We hypothesize that dynamic context management—*specifically, offloading irrelevant information from context and prioritizing the presence of relevant content in the context window*—can be facilitated by RL to enhance reasoning capabilities.

1.2 Proposed Extension

To address the problem of the accumulation of irrelevant information in context, we propose training the model to issue a special <clean> token to clean its context. This can be seen as an explicit form of backtracking, where, when the model makes a mistake or identifies an unproductive line of reasoning, it removes the traces of incorrect reasoning from the context completely instead of resuming from them. Crucially, we do not rely on external tools or memory interfaces. Instead, we train the model to learn when and how to use this context-cleaning behavior as part of its internal decision-making process. In the background section, we discuss a series of works that explore dynamic context management. These typically implement it through *agentic systems*, where language models are *not trained* to manage context themselves, but are instead *prompted* to interact with external managers.

1.3 Contributions

We summarize our contributions as follows:

- We curate an SFT dataset for teaching the model to issue special tokens that allow it to dynamically manage its context (Section 3.2.1).
- We implement a reinforcement learning objective and a generation pipeline that enables our model to leverage dynamic context management (Sections 3.2.2 and 3.2.3).

In our experiments:

- We find that our method improves performance on the difficult examples compared to the default project baseline (Section 4.3.1).
- Comparing our method using SFT to the version using RL, we find that RL training reduces the usage of clean, while improving performance on examples that still use clean (Section 4.3.2).
- We qualitatively assessed that the models we trained use the clean tokens as intended, and we showcase example reasoning steps in Section 4.3.3.

2 Background

2.1 Related Works

Dynamic context management has been explored in the literature from several perspectives, including incorporating memory systems with language models, designing agents that navigate text-based environments, and applying context compression techniques.

LLM + Memory System. Previous works have explored dynamically managing a model's context by incorporating a language model with an external memory system. Several of these studies emphasize the importance of structured memory, where memories are linked via a graph-like structure. Xu et al. [2025] introduce a graph-based memory system in which notes, composed of metadata and embeddings, evolve through semantic linking. Similarly, Rasmussen et al. [2025] present a temporal knowledge graph that maintains episode-wise and entity-centric subgraphs, facilitating long-term reasoning over time. Chhikara et al. [2025] extract natural language facts from conversations to incrementally build a vector-based memory and an optional knowledge graph (Mem0g), supporting long-term consistency in dialogue. Other works, such as MemGPT [Packer et al., 2024], have focused on hierarchical memory management. Notably, these approaches emphasize memory for multi-turn conversations, but do not train the language model to improve its ability to dynamically manage its context. The only work involving training is Wei et al. [2025], which trains a model to memorize its past experiences, rather than teaching it how to manage context dynamically.

Context Compression Another direction for context management is compressing the current context during reasoning. Mu et al. [2024] proposed gisting, which trains models to replace long prompts with compact learned embeddings that preserve performance while significantly reducing context length. Zhang et al. [2025] introduced LightThinker, a method that summarizes and discards intermediate reasoning steps on the fly, enabling efficient multi-step inference with minimal performance loss. InftyThink [Yan et al., 2025] demonstrated that iterative summarization can improve performance under context constraints, though their use of a much larger summarizer model (7B/70B) than the generator suggests that some gains may stem from improved reasoning rather than compression alone.

Continual Learning in Text Environments. Dynamically managing the context has also been the key for developing effecitive agents that can navigate text environments. Majumder et al. [2023] introduces a continually learning agent framework designed to extract contextual information from current episodes for future use, enabling cross-episode generalization in tasks like ScienceWorld [Wang et al., 2022]. Memory as a tool is increasingly operationalized through structured mechanisms. Claude Plays Pokémon [claudeplayspokemon, 2025] showcases how iterative summarization and an `update_memory` tool enable agents to manage long contexts efficiently. Suzgun et al. [2025] treats each problem as another episode in a text environment and learns from previously attempted question to better solve the next ones.

2.2 A Unifying Framework for Dynamic Context Management

We aim to provide a unifying perspective on prior work with our dynamic context management framework. The cognitive architectures [Sumers et al., 2024] in the existing literature typically employ a language model that interacts with an external memory system [Packer et al., 2024, Chhikara et al., 2025, Wei et al., 2025, Xu et al., 2025, Rasmussen et al., 2025, Suzgun et al., 2025]. These systems can

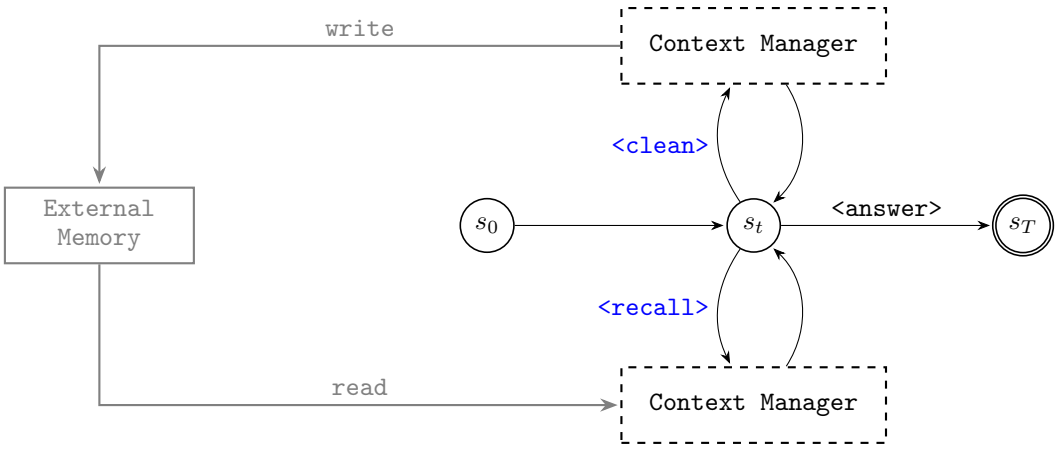

Figure 1: Cognitive architecture for dynamic context management operated by the language model. The language model selects actions that determine the transitions between states. The context manager is a Python program that edits strings and manages the interface with external memory. External memory serves as an information storage unit.

be conceptualized as *finite context machines*, which we define as a system that combines a language model, a context manager, and an external memory module. This system allows the language model to iteratively modify its context and interact with the memory via the context manager, curating its context to facilitate the reasoning needed to produce the desired outputs for the current task. We present our interpretation of such systems in Figure 1.

2.3 Our Extension

In our project, inspired by this idea, we train a language model to dynamically remove information from its context. As illustrated in the Figure 2, we allow the model to issue a `<clean>` token that enables it to clean its context and start afresh. We train our model using reinforcement learning to determine when to issue such clean tokens.

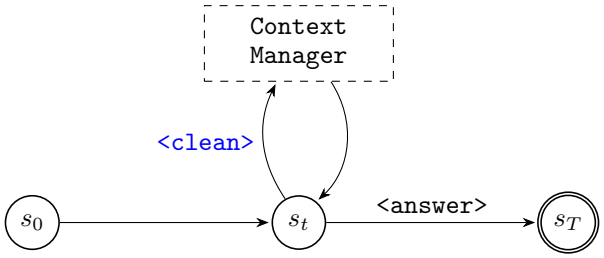

Figure 2: Basic version of the cognitive architecture for Finite Context Machine operated by the Language Model in our work.

This represents a first step toward enabling models to dynamically manage their context, which could further open up extensions for *selective information removal* which allows targeted deletion of specific context snippets, and *information recall*, which stores removed information in external memory for retrieval during later stages of reasoning.

3 Methodology

3.1 Default Project Setup

The goal of the default project setup is to teach language models to perform reasoning, including the abilities to backtrack and search effectively. The training pipeline for this consists of two

stages: Supervised Fine-Tuning (SFT) and Reinforcement Learning, particularly REINFORCE with Leave-one-out (RLOO) Ahmadian et al. [2024].

Supervised Fine-Tuning (SFT): This stage leverages reasoning traces produced by a strong expert model. These traces represent solutions that exhibit cognitive behaviors such as verification, backtracking, subgoal setting, and backward chaining [Gandhi et al., 2025]. Learning these behaviors during the SFT stage facilitates more effective exploration and learning during RL training. Formally, for a given question x, the policy π_θ is optimized to maximize the expected probability of an expert-generated response y as follows:

$$\max_{\theta} \mathbb{E}_{x,y\in D}\left[\frac{1}{|y|}\sum_{t=1}^{|y|}\log \pi_\theta(y_t \mid x, y_{<t})\right] \tag{1}$$

Here, y is expected to be of the *base* format illustrated in the Figure 3, having the in-context instructions for exhibiting the expected problem solving behaviors, and the structured generation with thinking and answering blocks.

Reinforcement Learning (RLOO): After the SFT stage, the policy is transferred to the reinforcement learning stage. Here, the model is exposed to a broader distribution of problems that have not been seen before. Unlike SFT, there are no reasoning traces to learn from, instead only rewards scoring the model's final solution from the useful aspects such as correctness and formatting. By leveraging this weak supervision, the model refines its behavior and learns to optimize the cumulative rewards, which leads to a better policy.

The RLOO algorithm Ahmadian et al. [2024] employed uses the policy to sample multiple completions for a given prompt, and assigns scores to these completions. Based on these, the rewards are adjusted in a leave-one-out fashion, and are used to optimize the policy. Formally, for a given question x, the algorithm utilizes the policy π_θ to generate k responses: $y_1, \ldots, y_k \sim \pi_\theta(\cdot \mid x)$. Given the reward assignment per response $R(x, y_i)$, the advantage per response is calculated in the following format:

$$A(x, y_i) = \left[R(x, y_i) - \frac{1}{k-1}\sum_{j\neq i}^{k} R(x, y_j)\right] \tag{2}$$

Using this advantage formulation, the following gradient is used for the optimization:

$$\frac{1}{k}\sum_{i=1}^{k} A(x, y_i)\frac{\nabla \log \pi_\theta(y_i \mid x)}{|y_i|} \tag{3}$$

This way, the policy is optimized in an unbiased manner with lower variance, increasing the likelihood of responses with higher reward.

3.2 Extension

3.2.1 SFT Data Curation

Building on the expert traces used in the SFT stage, we further augment the data to encourage the model not only to reason and search effectively, but also to manage its context by cleaning up its own generations. For this, we augment the existing expert traces in the form illustrated in the Figure 3.

We begin by applying a post-processing script to standardize all traces into a consistent expected structure shown in the *base* format of the Figure 3. This involves merging repeated `<think>` blocks into a single reasoning segment and retaining only the final `<answer>` block, removing any earlier answer attempts. Additionally, we discard any content generated outside the defined `<think>` or `<answer>` blocks. These steps help create cleaner and more consistent traces by fixing the generation artifacts of the expert model.

Next, we augment these traces in a targeted manner, as shown in the Figure 3. First, for all these traces, we append instructions describing the clean operation and its usage. Then, for the traces ending up with a correct final answer, we preserve the original `<think>` and `<answer>` blocks with the goal of allowing the policy to learn the standard problem-solving behaviors without distraction from the clean instructions. For the traces ending up with an incorrect final answer however, we

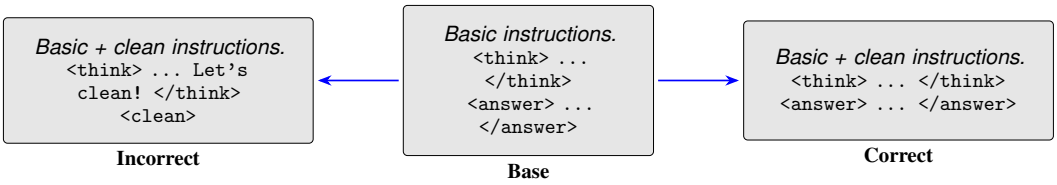

Figure 3: Variants of instruction formatting using <think> and <answer> tags.

assume that they are unproductive, and modify them by inserting an indicator for cleaning right before the end of the think block (e.g. a phrase like "the search is becoming confusing, so I should clean my context"), and insert a special <clean> token right after <think> block, removing the answer tag. This explicitly teaches the model to recognize and recover from flawed or unproductive reasoning traces, and giving itself another chance to operate in a fresh context.

By using this simple dataset curation method, we apply SFT and teach the model both reasoning / problem-solving behaviors and also explicit backtracking by issuing a clean instruction.

3.2.2 Context Management

While the extended SFT stage teaches the model the ability to use <clean> tags, we must define how these instructions are handled during both inference and RLOO training. Based on the conceptual pipeline illustrated in the Figure 1, we build the pipeline shown in Figure 4 for our extension. This version simplifies the conceptual framework by allowing the model to clean its context only once.

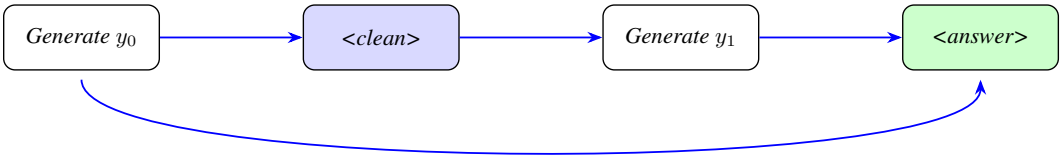

Figure 4: Two-stage generation with intermediate cleaning before final answer generation

To narrate, we first prompt the model with a question x, formatting instructions, and clean instructions. We use this to sample an initial response from the model, y_i^0, and send it to the context manager. The context manager checks if the model has issued a <clean> tag, and if it has, then stores the response, cleans the generations, removes the instructions on cleaning, and re-prompts the policy to generate the next and final response y_i^1. After a generation (y_i^0 or y_i^1), if the context manager detects an <answer> tag in the response, the generations stop, and that answer is used to represent the solution to this question from the policy. Moreover, after that, any generations for the current sample interaction are stored to be utilized by the reinforcement learning stage to calculate the gradients.

3.2.3 Modified RLOO Formulation

We now describe how the RLOO formulation is modified under the context management mechanism introduced in the previous section. Given a question x, let the model generate $y_i^0 \sim \pi_\theta(\cdot \mid x)$ for the ith response. Moreover, optionally, the model can may generate a follow-up response $y_i^1 \sim \pi_\theta(\cdot \mid x)$ in case the context manager detects a clean tag at the end of y_i^0. Then, formally, the reward for this interaction is defined as:

$$R(x, y_i^0, y_i^1) = \begin{cases} R(x, y_i^1), & \text{if } y_i^1 \neq \emptyset \\ R(x, y_i^0), & \text{otherwise} \end{cases} \tag{4}$$

This ensures that the final chosen solution (either y_i^0 or y_i^1) is the one being rewarded, matching the context manager description in the previous section. Given this, the leave-one-out advantage for each sample is computed as:

$$A(x, y_i^0, y_i^1) = R(x, y_i^0, y_i^1) - \frac{1}{k-1} \sum_{j \neq i} R(x, y_j^0, y_j^1) \tag{5}$$

where k being the number of responses from the policy for the given question x. Then, we modify the RLOO gradient to:

$$\frac{1}{k+k_c}\sum_{i=1}^{k} A(x, y_i^0, y_i^1)\left(\frac{\nabla \log \pi_\theta(y_i^0 \mid x)}{|y_i^0|} + \mathbb{1}[y_i^1 \neq \emptyset] \cdot \frac{\nabla \log \pi_\theta(y_i^1 \mid x)}{|y_i^1|}\right) \tag{6}$$

where $k_c = \sum_{i=1}^{k} \mathbb{1}[y_i^1 \neq \emptyset]$, indicating the number of times the policy cleans its context. Here, both y_i^0 and y_i^1 share the same advantage value, since the final reward is determined by the outcome of the complete trajectory. The gradients are computed for each segment separately and normalized by their respective lengths with appropriate response masking. In case the model proceeds directly to an answer, the learning signal is based only on the initial response y_i^0 whereas the case of cleaning and retrying backpropagates the cleaning signal through both y_i^0 (for learning when to clean) and y_i^1 (for learning how to respond after cleaning in a single attempt). This effectively and cleanly aims to model different modes, ultimately teaching the model to use clean effectively by deciding when to and when not to use, also adapt its response when the model cannot clean anymore.

4 Experiments

4.1 Setup

In our experiments, we target math reasoning task by using the Countdown dataset Gandhi et al. [2024, 2025], and we initialize our policy with the Qwen-2.5 0.5B model. For both the default and extension settings, we first train a model through supervised fine-tuning (SFT) by leveraging the expert reasoning traces from Gandhi et al. [2025], then apply reinforcement learning (RLOO) Ahmadian et al. [2024] on top of the trained model. We conduct the SFT experiments for 20 epochs with a learning rate of 2e-5, an effective batch size of 64, and a weight decay of 0.1. We apply a cosine learning rate scheduler with 5% warmup steps and train using cross-entropy loss. We conduct RLOO experiments for 700 total steps with a learning rate of 1e-6, an effective batch size of 16, and a weight decay of 0.01. We sample 4 responses per prompt and focus only on their correctness, ignoring formatting-related rewards when backpropagating gradients. For both the SFT and RLOO runs, we cap the maximum gradient norm at 1 and allow the model to generate up to 1024 new tokens, with temperature and top-p values set to 1. We evaluate SFT runs every 50 steps and RLOO runs every 10 steps on the milestone leaderboard validation set, keeping the model with the best validation score. We score the generations by considering the formatting and correctness, rewarding 0.1 and 1.0 respectively for the satisfactory ones. During both evaluation and training in both settings, we apply greedy decoding when generating responses.

4.2 Main Results

The evaluation results of the picked models are shared in tables 1 and 2. The results in Table 1 indicate that the extension RLOO run achieves the highest accuracy and score. Moreover, the extension RLOO also achieves the best score for the hard questions whose final expressions contain at least a division or multiplication operator. The extension models use significantly longer total context lengths compared to the default models. Finally, RLOO improves on top of the SFT in both default and extension setups. Table 2 on the other hand shows the evaluation results on the final leaderboard dataset. These results indicate that the extension SFT model still achieves a noticeably better performance compared to the default SFT model, while the extension RLOO model achieves on-par performance. We also observe that the extension models have longer average completion lengths compared to the default models, which aligns with the expectations from our extension that the model can leverage a longer horizon of context by effectively cleaning its context when necessary.

4.3 Findings

4.3.1 Extension vs. Default

Figure 5 provides a detailed view of the improvements introduced by the extension setting on top of the default. When comparing the accuracy and score metrics, we observe that the majority of the gains come from the SFT stage, indicating that the extension contributes most during supervised

	Default		Extension	
	SFT	**RLOO**	**SFT**	**RLOO**
Accuracy	41.50	76.00	59.50	**79.00**
Score	47.00	77.10	62.65	**80.25**
Easy Score	54.33	**90.18**	70.98	89.76
Hard Score	13.61	17.50	24.72	**36.94**
Average Completion Length	916.56	676.22	**1255.55**	1117.95

Table 1: Comparison of results from Default and Extension methods on the milestone leaderboard dataset. The dataset consists of 164 hard and 36 easy samples; hard samples include at least one division or multiplication in the final expression.

	Default		Extension	
	SFT	**RLOO**	**SFT**	**RLOO**
Accuracy	23.60	49.30	29.20	**49.80**
Score	30.15	51.63	34.64	**52.13**
Average Completion Length	1066.50	1046.10	1551.75	**1658.68**

Table 2: Performance for the evaluations on the final dataset.

fine-tuning. Looking at the improvements across question difficulty, we see a similar pattern for easy tasks as they mostly benefit during the SFT stage. In contrast, hard tasks show a different trend as the RLOO stage contributes significantly more, which suggests that reinforcement learning is especially helpful in improving performance on more difficult problems. For the final leaderboard dataset improvements, we observe similar trends where the SFT stage dominates the improvements by our extension.

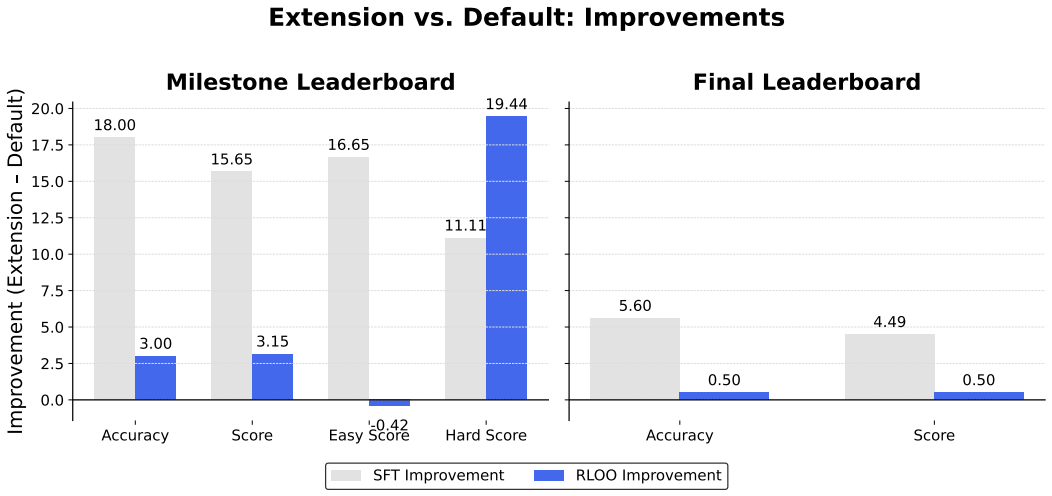

Figure 5: Comparison of extension vs. default results on the Milestone and Final leaderboard datasets (%). The bars indicate the performance difference between extension and default models per run type (SFT and RLOO).

4.3.2 Clean Usage

Figure 6 compares the usage rate of the `<clean>` token and the associated score when cleaning is used, for both the Extension SFT and Extension RLOO model under the milestone and final leaderboard datasets. We observe that in both datasets, the SFT model uses the `<clean>` token at a higher rate than the RLOO model. Furthermore, the RLOO model achieves a higher average score

when cleaning is invoked compared to SFT. This indicates that the model refines and improves the effectiveness of its <clean> token usage throughout the reinforcement learning training.

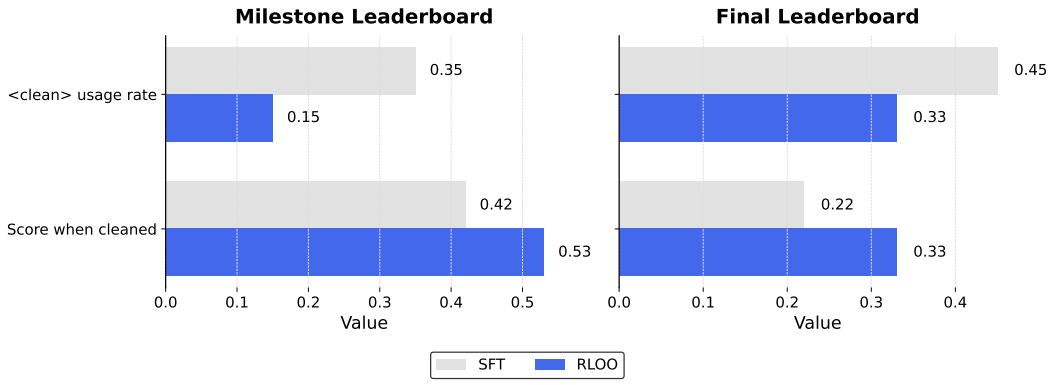

Figure 6: Clean usage rate and the expected score when the mode cleans its context. We observe that the model improves its usage of cleaning throughout the RL training by using it less and more effectively.

4.3.3 Reasoning Samples

In this section, we present examples of reasoning traces from the final trained policy. Figure 7 illustrates a typical chain-of-thought reasoning trace and a case where the model issues a <clean> token. We observe that the model often enumerates its sequence of attempts while searching for the correct expression and sometimes adds commentary on the results to guide its subsequent steps. These observations highlight that the model effectively learns cognitive behaviors from the expert traces, such as sub-goal setting and backtracking. Furthermore, when the model issues a <clean> token, it typically precedes this with a phrase indicating that the current line of reasoning is unpromising. These qualitative results suggest that the model has effectively learned a method of search and problem-solving, along with a mechanism for managing its context by cleaning it when necessary, due to the SFT and RLOO stages.

```
Let me analyze this step by step:
1. First, let's look for numbers
close to 68 or factors/multiples
that could help:
...
4. Let's try multiplication:
- 27 * 2 = 54 (close to 60)
- Not very helpful
5. New approach:
- 60 + 19 = 79 (too high)
- 60 - 27 = 33 (not helpful)
...

This works! Let's verify:
60 + 27 = 87
87 - 19 = 68
</think>
<answer> (60 + 27) - 19 </answer>
```

[60, 27, 19] → 68 **Example CoT**

```
...
After multiple attempts, I'm
having trouble reaching exactly
55 with these numbers using only
the allowed operations. Let me try
one final approach:

12. Let's try:
- 63 / 57 ≈ 1.08 (not helpful)
- 61 + 6 = 67
- 67 - 6 = 61 (not 55)

None of these attempts reach
exactly 55 as requested.
I realize my search is getting
confusing.
Let me reset and start fresh.
</think>
<clean>
```

[61, 63, 57] → 55 **Example Cleaning**

Figure 7: Examples of chain-of-thought with <think> and <answer> tags, and context reset via <clean> tag.

5 Discussion

Our work, while demonstrating a novel mechanism for context management, has several limitations that also highlight its broader implications and directions for future research.

Trade-off Between Persistence and Resetting. Training the model to decide when to clean introduces a trade-off between continuing a difficult reasoning path and choosing to reset. In some cases, this may lead the model to give up on problems that could still be solved with more steps, making it less robust in challenging settings.

Loss of Information. Cleaning the context removes all previous content, which means the model cannot use earlier attempts to guide future reasoning. This limits the opportunity to recover from mistakes or build on partial progress. Future work could explore ways to summarize or retain useful parts of earlier reasoning.

Limited Task Complexity. The benefits of context cleaning are likely to be more useful in tasks with longer or more complex reasoning chains. In our experiments, many examples are short enough that the gains from cleaning may be smaller than what could be achieved in other domains.

Single-Use Cleaning. For simplicity, we allow the model to clean its context at most once. While this helps reduce complexity, it also limits the flexibility of the model. A more advanced system could support multiple cleaning steps or selective removal of context, enabling better control in multi-stage reasoning.

6 Conclusion

Overall, we propose a framework that enables models to explicitly backtrack by issuing a clean token, enhancing the models' ability to manage context effectively. Our setup extends a default configuration by curating SFT data, applying a context manager to guide generation, and adapting the RL formulation. Both quantitative and qualitative results demonstrate the effectiveness of our approach. While we acknowledge areas for improvement, we also highlight future directions informed by both the limitations and strengths of our work. In future research, we plan to explore training models equipped with memory mechanisms that support read/write operations. This would enable more advanced and dynamic context management, allowing models to maintain state, track previously explored trajectories, and avoid revisiting unpromising ones. We believe a similar reinforcement learning approach can be leveraged to support this behavior effectively.

7 Contributions

Below is a breakdown of the work done in the project.

Default:

- Implementing SFT objective using Qwen 2.5 0.5B Base: Batu, Hamza
- Implementing RLOO algorithm: Batu, Hamza
- Building verifier-based dataset loader for Countdown (format + verification reward): Batu, Hamza
- Constructing prompts dataset for RLOO: Hannah
- Validating dataloader correctness and efficiency, and ensuring proper formatting: Hannah

Extension:

- Proposing the extension approach (ideation): Batu
- Augmenting existing expert traces used in the SFT stage to encourage the model to manage its context by cleaning up its own generations: Hamza, Hannah
- Defining the modified RLOO formulation: Batu, Hamza
- Implementing the context manager for inference and RLOO: Hamza
- Validating extension model performance on the Countdown task: Batu, Hannah
- Conducting experiments to compare default and extension methods: Batu, Hannah
- Conducting experiments to compare performance improvement/reduction across datasets, and usage rate of the <clean> token for SFT and RLOO models: Batu, Hamza
- Running experiments to produce reasoning samples demonstrating model reasoning process: Batu, Hannah
- Clarifying limitations of current work and directions for future research: Batu

Logistics:

- Writing the project proposal: Batu, Hamza, Hannah
- Reviewing the literature to gather knowledge on relevant prior work: Batu, Hamza, Hannah
- Writing the milestone report: Batu, Hamza, Hannah
- Making the poster: Batu, Hamza, Hannah
- Presenting at the poster session: Batu, Hamza
- Writing the final report: Batu, Hamza, Hannah
- Doing the leaderboard submissions with initial checkpoints for countdown task and with extension run: Hamza

Reason for adjustments from original breakdown proposed:

The allocation of tasks evolved naturally from the initial proposed breakdown simply due to the iterative nature of the new-idea-developing process, where at any point, the person(s) who had the capacity to step in to address emergent work or refine the scope based on intermediate findings did so.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms, 2024. URL https://arxiv.org/abs/2402.14740.

Chenxin An, Jun Zhang, Ming Zhong, Lei Li, Shansan Gong, Yao Luo, Jingjing Xu, and Lingpeng Kong. Why does the effective context length of llms fall short?, 2024. URL https://arxiv.org/abs/2410.18745.

Giannis Chatziveroglou, Richard Yun, and Maura Kelleher. Exploring llm reasoning through controlled prompt variations, 2025. URL https://arxiv.org/abs/2504.02111.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

claudeplayspokemon. Pokémon gameplay and chat interaction, June 2025. URL https://www.twitch.tv/claudeplayspokemon.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (sos): Learning to search in language, 2024. URL https://arxiv.org/abs/2404.03683.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars, 2025. URL https://arxiv.org/abs/2503.01307.

Peyman Hosseini, Ignacio Castro, Iacopo Ghinassi, and Matthew Purver. Efficient solutions for an intriguing failure of llms: Long context window does not mean llms can analyze long sequences flawlessly, 2024. URL https://arxiv.org/abs/2408.01866.

Yue Huang, Yanbo Wang, Zixiang Xu, Chujie Gao, Siyuan Wu, Jiayi Ye, Xiuying Chen, Pin-Yu Chen, and Xiangliang Zhang. Breaking focus: Contextual distraction curse in large language models, 2025. URL https://arxiv.org/abs/2502.01609.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL https://arxiv.org/abs/2307.03172.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization, 2023. URL https://arxiv.org/abs/2310.10134.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens, 2024. URL https://arxiv.org/abs/2304.08467.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. URL https://arxiv.org/abs/2501.13956.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context, 2023. URL https://arxiv.org/abs/2302.00093.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents, 2024. URL https://arxiv.org/abs/2309.02427.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. URL https://arxiv.org/abs/2504.07952.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader?, 2022. URL https://arxiv.org/abs/2203.07540.

Jiale Wei, Xiang Ying, Tao Gao, Fangyi Bao, Felix Tao, and Jingbo Shang. Ai-native memory 2.0: Second me, 2025. URL https://arxiv.org/abs/2503.08102.

Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. Inftythink: Breaking the length limits of long-context reasoning in large language models, 2025. URL https://arxiv.org/abs/2503.06692.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. Lightthinker: Thinking step-by-step compression, 2025. URL https://arxiv.org/abs/2502.15589.