

Extended Abstract

Motivation In this project, we tackle the Countdown math reasoning problem, which often requires models to generate long, multi-step reasoning traces. It has been shown that language models reason better over shorter contexts, and our aim is to train a model to use a tool which summarizes its own reasoning context, thereby compacting its reasoning trace. We hypothesize that the summarize tool will help the model reason better by preserving only the most relevant content from its previous context, while alleviating the problems caused by long context windows.

Method We train the model to use the summarize tool using both an SFT warm-start and further RLOO finetuning. For the SFT warm-start, we create our own custom dataset that includes demonstrations of when to call the summarize tool, as well as how to continue after a summary. The summarize tool is implemented by running inference on a larger external summarizer model, prompted to produce concise, relevant summaries. During rollouts, the model runs in a loop that either ends with a summarize tool call or the model submitting its answer, allowing the model to summarize its context whenever it chooses to. After our SFT warm-start teaches the model to use the summarize tool, we finetune it using RLOO with a custom reward function. This reward function encourages tool-calling with a positive "tool bonus", as well as a length penalty that adds a negative reward per token of context length. Combined, these two phases of training teach the model how and when to call the summarize tool for optimal context compression.

Implementation Our summarizer tool calls the Qwen 2.5 7B Instruct model, and the model calls the tool by using a special token. We constructed our SFT warm-start dataset based off of the original SFT dataset. In particular, we modified each entry to produce a pair of entries: an "invoke" example that shows the model when to call the summarize tool, and a "continue" example that demonstrates how to continue reasoning after a summary. For RLOO finetuning with our custom reward function, we experimented with various hyperparameters, including the values for the tool bonus and length penalty multiplier in our reward function. We trained for 100 steps with a constant learning rate, a positive KL penalty, and positive entropy coefficient. We found that using a small positive tool bonus and length penalty multiplier led to more stable training.

Results Our summarizer methods achieved slightly lower but comparable accuracy to the RLOO baseline, doing so using a lower average rollout context length. In particular, the RLOO baseline achieved the highest pass@1 of 0.55, but our SFT warm-start summarizer achieved a pass@1 of 0.47 with an average context length of about 100 less tokens. We found that our summarize tool was effective in preserving important context in a compact, bullet-point fashion, reducing the length of summarized text by a factor of around 3.5x on average.

Discussion Our summarize tool successfully shortened the model's average reasoning context while maintaining a comparable accuracy on Countdown problems. However, one challenge we faced was that many of the problems we used for our SFT warm-start dataset and those in the evaluation dataset were not intended for especially long reasoning traces, while the summarize tool is best suited for complex, multi-step reasoning. This led to cases when the model redundantly calls the tool when it doesn't need to, such as when it is already close to finding an answer. Another challenge was that constraints on training time and compute made it difficult to train to convergence with our desired parameters. We had to limit the model to only one tool call per rollout because of this issue.

Conclusion Through our experiments, we showed that language models have the potential to manage their own reasoning traces, achieving comparable performance on the Countdown math reasoning task with a shorter average context length. For future work, we would like to train the model to generate its own summaries instead of calling an external summarizer, as well as perform a more thorough hyperparameter sweep and train our RLOO summarizer to convergence.

Reinforcement Learning for Self-Guided Context Compression in Mathematical Reasoning

Ryan Wang

Department of Computer Science
Stanford University
wangrj@stanford.edu

Jerry Wang

Department of Computer Science
Stanford University
jwang028@stanford.edu

Abstract

In this project, we train a language model to use a tool that summarizes its own reasoning to aid it in the Countdown math reasoning task. It has been shown that models reason better over short contexts, and we hypothesize that our summarize tool can help shorten the model’s reasoning context while preserving the most important information. We train the model to use our tool using our own SFT warm-start dataset and RLOO finetuning with a custom reward function, showing that it achieves comparable performance to the RLOO baseline while requiring a shorter average context length. Our experiments demonstrate that the summarize tool is effective in helping the model reduce its reasoning length, and that reinforcement learning can be used to teach the model when the best time is to summarize. For future work, we hope to extend this project by finetuning the model to produce summaries tailored specifically for Countdown-style problems.

1 Introduction

Large language models (LLM) have shown success across a wide variety of natural language reasoning tasks, and is nowadays widely used in code generation, fact recalling, and mathematical reasoning. Prior to the development of reinforcement learning (RL), supervised fine-tuning (SFT) has been the dominant way to adapt these models. However, reinforcement learning has recently become one of the most important paradigms for LLM fine-tuning. Instead of relying on the imitating of human-labeled data, RL methods can allow the model to optimize directly for desired outcomes through feedback. This enables the model to self-correct errors, refine their responses, and potentially capture more nuanced patterns, enhancing their performances as compared to regular fine-tuning.

Two RL methods are especially relevant. The first method is training on preference data, which involves comparing the qualities of two different responses on the same query. In this method, a reward function is defined implicitly to try to maximize the reward margin between the preferred and dispreferred responses. The second method is training with verifier-based rewards, which involves comparing against objective truth, such as correctness and consistency. This method is especially useful for mathematical reasoning, where there are objective and verifiable answers.

This project focuses on Countdown problems, where the model is given sets of integers and asked to construct equations with arithmetic operations to reach specific target values. These problems often involve multiple steps and sometimes complex mathematical reasoning, therefore testing the model’s ability to plan, decompose, and execute the required sequence to solve problems. This problem allows for a controlled environment to evaluate different model’s performances using different methods.

In addition to these existing RL approaches, we propose an extension based tool-integrated reasoning with a summarization tool. Countdown problems can involve complex and lengthy mathematical reasoning, causing the context to grow substantially. The context would contain some correct reasoning in the path to the target, but also a lot of incorrect reasoning paths that should be discarded.

In addition, it has been shown that models perform better when they reason over shorter contexts. The summarization tool introduces a way for the model to selectively retain useful information while discarding irrelevant paths, simultaneously shortening the context. We hypothesize that RL will prove useful in learning when summarization becomes useful, thereby improving reasoning performance. Furthermore, shorter context lengths would increase efficiency and lower computation costs. To evaluate this new approach, we will compare models trained with and without the summarization tool on both task accuracy as well as average tokens per episode.

2 Related Work

Wei et al. (2023) first introduced Chain-of-Thought Reasoning, showing that intermediate reasoning steps significantly boost performance on complex, multi-step tasks like math problems. This motivates our baseline approach of finetuning a model to approach Countdown problems using reasoning. Using reinforcement learning to train reasoning was first demonstrated by Guo et al. (2025), which demonstrated that GRPO with sparse rewards enables models to self-discover effective reasoning behaviors.

One challenge faced by reasoning models is that excessively long reasoning traces can negatively impact performance. Liu et al. (2023) found that models struggle when relevant information is in the middle of a long context (as opposed to at the beginning or end), with some performing worse than a closed-book baseline. This is especially pertinent for a challenging task like Countdown which requires many intermediate steps, as crucial information could be buried in the middle of long reasoning traces.

Dai et al. (2025) addressed long reasoning traces by adding a length penalty to the GRPO reward function, resulting in a significant reduction in sequence length and more stable training. However, this approach does not consider giving the model any mechanism to preserve important information. Our tool-based approach attempts to fill this gap by giving the model control over when and what to preserve via summarization. Zhang et al. (2026) tackle long inputs by allowing an LLM to recursively call itself, shortening the context that each model needs to handle. In doing so, they find success on long-context reasoning tasks. Similarly, our approach enables a model to break down long reasoning traces. Prior work such as Feng et al. (2025) has shown that RL has potential to allow models to autonomously discover optimal tool invocation patterns.

These related work show two insights. First, reinforcement learning is an effective framework for enhancing reasoning capabilities and learning complex behaviors such as tool use. Second, context management is important for maintaining performance on long-horizon reasoning tasks. Our approach combines these two directions by training a model with RL to directly invoke a summarization tool during reasoning. In doing so, the model must learn when summarization is beneficial, which helps us investigate whether RL can learn effective reasoning context-compression strategies that improve reasoning performance and computational efficiency.

3 Method

Supervised Fine-Tuning (SFT) The first baseline model is a supervised fine-tuning of the base model. SFT uses standard next-token prediction, identical to the pre-training objective; however, the loss is applied only to completion tokens, not on the query tokens. The objective

$$\max_{\theta} \mathbb{E}_{x,y \in D} \sum_{t=1}^{|y|} \log \pi_{\theta}(y_t | x, y_{<t}),$$

where x are queries and y are expert completions, is optimized over examples in dataset D .

Implicit Preference Optimization (IPO) The second baseline model learns from a preference dataset to maximize the reward margin between the preferred and dispreferred responses. For a query x and response y , the reward is defined as

$$r_{\theta}(x, y) = \beta(\log \pi_{\theta}(y|x) - \log \pi_{\text{ref}}(y|x)).$$

We denote y_w as the preferred response and y_l as the dispreferred response, so we want to maximize the reward margin

$$r_{\theta}(x, y_w) - r_{\theta}(x, y_l).$$

Azar et al. (2023) relaxes the Bradley-Terry modelling assumptions in Direct Preference Optimization (DPO) that human preferences are generated according to a logistic model of reward differences to reduce overfitting to preference labels. IPO optimizes the policy by encouraging the relative log-likelihood margin between preferred and dispreferred responses to match a target value. It optimizes the objective

$$\mathcal{L}_{\text{DPO}}(\pi_{\theta}; \pi_{\text{ref}}) = \|h_{\pi_{\theta}}^{y_w, y_l} - (2\beta)^{-1}\|_2^2,$$

where

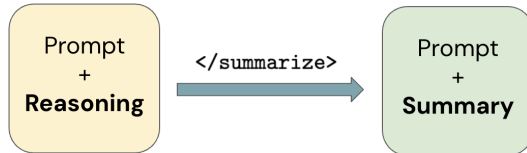
$$h_{\pi_{\theta}}^{y_w, y_l} = \log \frac{\pi_{\theta}(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_{\theta}(y_l|x)}{\pi_{\text{ref}}(y_l|x)}.$$

Note that reference policy π_{ref} uses the SFT model.

REINFORCE Leave-One-Out (RLOO) The third baseline model is an online reinforcement learning algorithm that directly optimizes the policy using a rule-based verifier reward. Unlike IPO, RLOO samples responses from the current policy and updates the model based on the reward. To reduce the variance of the policy gradient estimate, RLOO uses a leave-one-out baseline, where the reward of the current response is evaluated against the average reward of other responses. The policy is then updated to encourage responses with above-average rewards and discourage responses with below-average rewards. This objective is written as

$$\frac{1}{k} \sum_{i=1}^k \left[R(y_{(i)}, x) - \frac{1}{k-1} \sum_{j \neq i} R(y_{(j)}, x) \right] \nabla \log_{\pi}(y_{(i)}|x), \text{ for } y_{(i)} \sim \pi_{\theta}(\cdot|x).$$

Summarization Tool We created a summarization tool, which takes all the context so far and calls a stronger external model, Qwen 2.5 7B Instruct, using a custom prompt to summarize the existing context. The prompt asks the model to summarize the context in at most 8 bullet points, stating what was attempted and the reason for failure. The model is also asked to retain useful intermediate values. The tool is invoked whenever a `</summarize>` token is emitted. The resulting summary replaces the full reasoning trace in the context window, which reduces the context length while helping the model retain useful information.



SFT Warmup with Summarization To enable the model to use the summarization tool, we created a custom dataset based on the SFT Warm-start dataset containing examples where the model emits `</summarize>` tokens. There are two types of training examples: "Invoke" and "Continue". "Invoke" examples demonstrate when the model should invoke the summarization tool and emit the `</summarize>` token, while "Continue" examples demonstrate how the model should respond to a summary. This stage teaches the model when to call the summarization tool as well as the continuation after the tool is used. This stage is necessary to encourage the model to use the summarization tool.

Below are a pair of examples from our warmup dataset. The first is an "invoke" example. In this entry, the "completion" contains reasoning up until the point when the summarize tool is invoked, indicated by the entry ending with the special `</summarize>` token. The second example is a "continue" example, which demonstrates how to continue reasoning after a summary. Notice that this entry contains the reasoning summary in the "prompt" (before "completion"), between the `<summary>` tags. To ensure that training on this dataset encourages the correct distribution, these summaries are generated using the same external model used for rollouts, Qwen 2.5 7B Instruct. The "completion" that follows the summary is an example of how to continue reasoning after seeing a summary, starting a new `<think>` and eventually providing an answer to the problem.

```
step by step.</im_end>><im_start>assistant</im_start>, "completion": "<think>Let me try to solve this systematically.</im_start>. First, let's try to get close to 78 using the larger numbers:</im_start>\n86 - 2 = 84 (same as one of our numbers)</im_start>\n84 - 2 = 82 (too high)</im_start>\n88 + 2 = 70 (too low)</im_start>. Let's work backwards from 78:</im_start>\n78 = 156/2</im_start>\n78 = 39*2</im_start>. Looking at our numbers, we can try to get to 156 first:</im_start>\n84 + 68 = 152 (close to 156)</im_start>\n88 + 68 = 154 (even closer)</im_start>. Let's try a different approach:</im_start>\n84 - 68 = 16</im_start>\n86 - 2 = 84</im_start>. What if we try division first:</im_start>\n84/2 = 42</im_start>\n68/2 = 34</im_start>. I notice that 78 is between 68 and 86. Let me try:</im_start>\n86 - 2 = 84</im_start>\n84 - 68 = 16</im_start>\n68 + 16 = 84 (not helpful)</im_start>. New approach:</im_start>\n86 - 84 = 2</im_start>.</summarize>", "type": "invoke"}
```

Figure 1: An "invoke" example from our warmup dataset.

```
</answer> tags, for example <answer> (1 + 2) / 3 </answer>.Assistant: Let me solve this step by step.</im_end>><im_start>assistant</im_start></summary>\n- 84 + 68 = 152, too low</im_start>\n- 88 + 68 = 154, close but too high</im_start>\n- 84 - 68 = 16, not useful directly</im_start>\n- 86 - 2 = 84, not helpful</im_start>\n- 84/2 = 42, useful intermediate</im_start>\n- 68/2 = 34, useful intermediate</im_start>\n</summary></im_start>, "completion": "<think>86 + 2 = 78</im_start>\n78 + 8 = 78 (we need to get 8 somehow)</im_start>. Final attempt:</im_start>\n84 - 68 = 16</im_start>\n16 + 2 = 18</im_start>\n86 - 32 = 54</im_start>\n54 + 24 = 78</im_start>. Let me try one more sequence:</im_start>\n86 - 84 = 2</im_start>\n2 + 68 = 136</im_start>\n136 - 58 = 78</im_start>.</think></im_start> (86 - 84) + 68 </answer>", "type": "continue"}
```

Figure 2: A "continue" example from our warmup dataset.

We constructed the "invoke" and "continue" examples in pairs, by choosing a split point in each of the entries of the original SFT warm-start dataset and summarizing the pre-split reasoning to create an "invoke" example, while building a corresponding "continue" example using the post-split reasoning. To ensure that our dataset is representative of how the summarize tool should be used, we only used entries longer than 25 lines and chose split points longer than 20 lines. This reflects that the summarize tool should only be called after some amount of reasoning. We also used string matching to ensure that the correct answer never appears before the split point, as there is no point in calling summarize if the model has already finished the problem.

RLOO with Summarization We further train this model using RLOO with the summarization tool enabled during rollout. In practice, to enable the summarize tool, we run the model in a loop. If the model emits the `</summarize>` token, the external summarizer is called and the current reasoning trace is replaced with the summary. In the next iteration of the loop, the model is called again with the summarized context. This loop repeats until the model emits the special `<answer>` token indicating that it is ready to finish the problem and submit its answer. For our RLOO training, due to time and compute constraints, we limit the model to call the summarize tool at most once per rollout. While this is a significant limitation, we found it was necessary to keep training time in check and for training to remain stable. Moreover, it still allows the model to learn the best time to summarize its context, keeping context short.

For RLOO with summarization, we trained the model with a custom reward function that rewards tool use and penalizes context length, given by the equation

$$R = R_{\text{task}} + b_{\text{format}} + b_{\text{tool}} - \alpha \cdot l,$$

where R_{task} is the verifier reward, b_{format} is a reward for correctly formatted responses, b_{tool} is a reward for tool usage, l is the maximum length of the model's context, and α is a hyperparameter. With a positive α value, the last term encourages the model to keep its reasoning trace concise. Along with the other terms, this aims to encourage the model to summarize when its context becomes too long. Though the length-penalizing term is already encouraging tool use, we found empirically that using a small positive tool bonus b_{tool} keeps tool-use rates more stable throughout training, and makes sure that the model doesn't start to "forget" about using the tool. Overall, this stage refines the model's tool-calling capabilities by teaching it to limit its context lengths.

4 Experimental Setup

Model and Libraries The base model used as the initialization for fine-tuning is the Qwen 2.5 0.5B Base model. Hugging Face libraries were used for infrastructure tasks, including model loading, tokenizer loading, and dataset loading.

Supervised Fine-Tuning (SFT) The dataset used to train the SFT model is the public Warm-start SFT dataset (https://huggingface.co/datasets/Asap7772/cog_behav_all_strategies). We implemented SFT with a batch size of 16, a gradient accumulation step count of 1, a gradient clipping threshold of 1.0, a constant learning rate of 5×10^{-6} , and an AdamW optimizer.

Implicit Preference Optimization (IPO) The dataset used to train the IPO model is the pairwise preference dataset (https://huggingface.co/datasets/asingh15/countdown_tasks_3to4-dpo). We implemented IPO for 1 epoch with a batch size of 64, a gradient accumulation step count of 16, a gradient clipping threshold of 1.0, a β of 0.1, a constant learning rate of 5×10^{-6} , and an AdamW optimizer.

REINFORCE Leave-One-Out (RLOO) The dataset used to train the RLOO model is the prompt/evaluation dataset (https://huggingface.co/datasets/asingh15/countdown_tasks_3to4). We implemented RLOO with a group size of $K = 16$ rollouts per prompt, where the baseline of each response is the average reward of the 15 other responses. We trained for 100 steps with a constant learning rate of 1×10^{-5} , a KL divergence coefficient of 0.001, an entropy coefficient of 0.001, a batch size of 128, and a gradient clipping threshold of 1.0.

Summarizer Tool Dataset We created a custom dataset to train the model to use the summarizer tool based on the SFT Warm-start dataset, as described in the previous section. The dataset contains explicit demonstrations of tool usage for imitation learning, consisting of examples of when to summarize, as well as how to continue reasoning after a summary.

Evaluation Metrics Our main evaluation metrics were Pass@1 (accuracy) on the Countdown evaluation dataset and the average context length per episode in tokens. These metrics allowed us to determine which methods solved the Countdown task most effectively while using a short context length.

5 Results

5.1 Quantitative Evaluation

Algorithm	Variant	Pass@1	Average Context (tokens)
SFT	Baseline	0.27	282
IPO	Baseline	0.35	527
RLOO	Baseline	0.55	477
SFT	Summarizer	0.47	379
RLOO	Summarizer	0.42	425

Table 1: Accuracy and average context length for various methods.

We computed the accuracy (Pass@1) and average rollout context length for each of our baseline and summarizer methods, shown in Table 1. The baseline RLOO was the method that achieved the highest Countdown accuracy, at 0.55. with the SFT summarizer and RLOO summarizer at 0.47 and 0.42 respectively. Notably, the average context length for our summarizer methods was significantly less than that for our IPO and RLOO baselines, with the SFT summarizer at an average rollout context length of 379 compared to 527 and 477 for the baselines. Both summarizers achieved accuracy comparable to our best baseline (RLOO) while having a lower average context, demonstrating that the summarize tool has potential in shortening context while preserving necessary information.

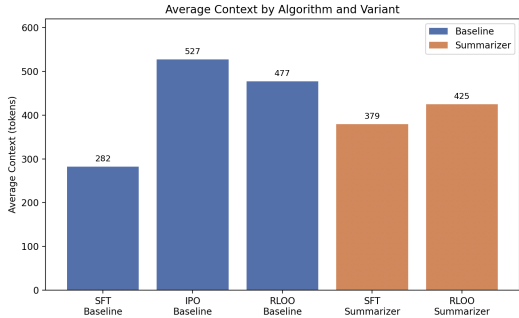


Figure 3: Average rollout context length for each method.

Figure 1 provides an additional visual comparing average context length across our methods. Though the SFT baseline has a much shorter average context, it also has the lowest accuracy at only 0.27, suggesting that a context that is too short is insufficient to preserve important reasoning traces. We note that the RLOO summarizer has a longer average context than the SFT summarizer, which may suggest that RLOO tends to teach the model to be more selective in when it calls the summarize tool, or that it tends to nudge the model toward longer reasoning traces in general.

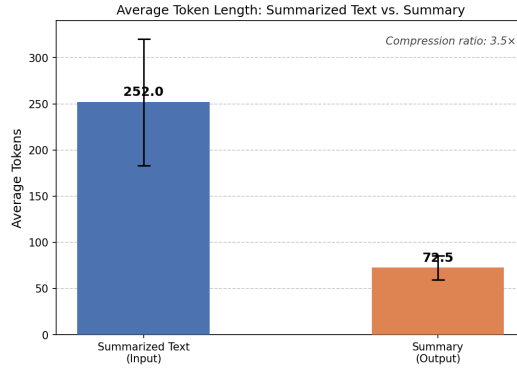


Figure 4: Average length of summarized text and summary text in our SFT warmup dataset

To verify that the summarize tool is indeed the reason behind shorter contexts, we analyzed the average lengths of summarized (pre-tool) text and summary text in our SFT warmup dataset. Since the same summary model used in rollouts was used to construct the dataset, this provides an accurate estimate of the rough ratio between the lengths of summarized and summary text. This comparison is shown in Figure 2. The summarized text in our dataset was 252 tokens long on average, and the summaries were only 72.5 tokens on average, indicating that the summarize tool provides a roughly 3.5x compression ratio. This large compression rate shows that the summarize tool is shortening context as intended, and that the short, bullet-point format specified by our prompt is successful in generating a concise summary.

5.2 Qualitative Analysis

```
tags, for example <answer> (1 + 2) \ / 3 </answer>.\nAssistant: Let me solve this step by
step.<|im_end|>\n<|im_start|>assistant\n\n<summary>\n- 66 - 48 = 18, still too large\n- 80
- 66 = 14, too small\n- 80 - 48 = 32, too small\n- 80 - 36 = 44, too small\n- 66 - 48 = 18,
reduce further\n- 48 - 36 = 12, combine steps\n- 8\n</summary>\n<think>\nLet me try to
reach 62 using these operations.\n\nFirst attempt:\n80 - 66 = 14\n14 + 48 = 62!\n\nThis
works! Let me verify:\n(80 - 66) + 48 = 14 + 48 = 62\n</think>\n<answer> (80 - 66) + 48 </
answer>","<|im_start|>system\nYou are a helpful assistant.<|im_end|>\n<|im_start|>user\nA
```

Figure 5: Example of the model successfully using a summary

We found that in many cases, the model was able to use tool-generated summaries to aid in solving the Countdown problem. Figure 3 shows a rollout where the model has already called the summarize tool, and soon goes on to find the correct answer. In this example, the summary generated by the tool is between the `<summary>` tags, and includes intermediate values like 18, 14, and 32. In its following reasoning, the model is able to use the value 14 from the summary, correctly reaching the target number 62 by discovering that $14 + 48 = 62$.

```
answer>.\nAssistant: Let me solve this step by step.<|im_end|>\n<|im_start|
>assistant\n\n<summary>\n- 44 + 19 = 63, too small\n- 63 + 35 = 98, correct\n- 44 * 19 =
836, too large\n- 35 * 19 = 665, too large\n- 44 * 35 = 1540, too large\n</
summary>\n<think>Let me try to find a way to get to 98.\nFirst, let me try working with the
larger numbers:\n35 + 44 = 79\n79 + 19 = 98!\n\nThis works! Let me verify:\n35 + 44 = 79\n79
+ 19 = 98\n</think>\n<answer> (35 + 44) + 19 </answer>","<|im_start|>system\nYou are a
```

Figure 6: Example of a redundant summary.

In some cases, the summarize tool was called redundantly, as in the example shown in Figure 4. Here, the model had already reached the target number 98 before calling the summarize tool, as evidenced by the bullet point "- 63 + 35 = 98, correct". Yet, it continues reasoning after the summary, rediscovering the answer instead of immediately concluding. We found that this issue occurred often for simpler problems where the reasoning required to solve the problem was relatively short. Since our SFT warmup dataset and RLOO summarizer training explicitly encourage the use of the summarize tool, the model can call the summarize tool even when it doesn't need to, and sometimes when it has already found the answer, as in the above example.

6 Discussion

Our experiments showed that a summarize tool can effectively shorten the model's reasoning context while still maintaining a comparable accuracy. The summary was able to preserve important information needed to solve the task, and was significantly shorter than the summarized text on average.

One challenge we faced was that it was difficult to train our model specifically for problems that required longer reasoning traces. The summarize tool provides greater benefit and makes sense to use mostly after a reasoning trace has already accumulated significant length, but many Countdown problems in both the SFT dataset and the evaluation dataset were intended to be solved with a shorter reasoning trace. Specifically, though we filtered for longer reasoning outputs when constructing our custom SFT dataset, the average output length of an entry in the dataset was still only roughly 30 lines. This may have meant that the model picked up on behaviors like summarizing too early or when an answer is already close to being found. More experimentation is required to determine the length threshold for reasoning traces above which summarization produces a significant accuracy boost, but we hypothesize that the benefit comes for relatively longer reasoning traces on average.

Another big challenge was the compute and training time required in generating rollouts with the summarization tool. Even with batching and a multi-GPU setup, generating rollouts was significantly slower due to the external summarizer being called multiple times per episode. Though we attempted to optimize our inference setup using vLLM, the RLOO summarizer training proved too time- and compute-intensive to run to full convergence. This was one of the main reasons that we limited the model to one summarize call per episode, but we believe that more complex behaviors can likely be learned by allowing multiple tool invocations per episode, which could lead to longer reason-summarize-reason cycles for especially difficult problems.

7 Conclusion

In this project, we showed that a language model has the potential to manage its own reasoning context in solving Countdown math puzzles. We trained the model to use a tool that summarizes its own context, achieving comparable accuracy to a RLOO baseline while lowering the average context length of the model's reasoning traces. This demonstrates that a summary can preserve important context while lessening the need for long context windows, alleviating the problems caused by long context.

For future work, we hope to extend this project by finetuning the model to generate its own summaries instead of calling an external summarizer. We hope that this can help the model decide on a summary format that is the most optimal for Countdown-style problems, which may be beneficial for performance. We also hope to further improve our training setup by collecting a larger SFT Warm-start dataset featuring more difficult problems that require longer reasoning traces, as well as by sweeping more hyperparameters for our RLOO summarizer training. If possible, we would also like to optimize our compute setup and attempt to train to convergence for our RLOO summarizer while allowing the model multiple tool invocations per rollout.

8 Team Contributions

- **Ryan:** Extended Abstract, Abstract, Results, Discussion, Conclusion, SFT warm-start dataset construction, RLOO training with custom reward function

- **Jerry:** Introduction, Related Work, Method, Experimental Setup, Summarization tool, SFT Warmup with summarization

Changes from Proposal We slightly changed the format of the reward function for training stability and simplicity of implementation, and decided to implement our summarizer using an external model and leave the finetuned summarizer for future work.

References

- Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. 2023. A General Theoretical Paradigm to Understand Learning from Human Preferences. arXiv:2310.12036 [cs.LG] <https://arxiv.org/abs/2310.12036>
- Muzhi Dai, Shixuan Liu, and Qingyi Si. 2025. Stable Reinforcement Learning for Efficient Reasoning. arXiv:2505.18086 [cs.AI] <https://arxiv.org/abs/2505.18086>
- Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. ReTool: Reinforcement Learning for Strategic Tool Use in LLMs. arXiv:2504.11536 [cs.CL] <https://arxiv.org/abs/2504.11536>
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, 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, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, 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, Mingxu Zhou, 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, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, 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, Yudian Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, 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. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature* 645, 8081 (2025), 633–638. doi:10.1038/s41586-025-09422-z
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172 [cs.CL] <https://arxiv.org/abs/2307.03172>
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] <https://arxiv.org/abs/2201.11903>
- Alex L. Zhang, Tim Kraska, and Omar Khattab. 2026. Recursive Language Models. arXiv:2512.24601 [cs.AI] <https://arxiv.org/abs/2512.24601>