Extended Abstract

Motivation In our previous project, we developed a retrieval-based method, RINGER [1], utilizing abstract rules and achieved encouraging results in math reasoning. However, merging rules requires extensive searching time to find optimal rules with low NLLs, and thus, the method is hard to scale up. To unlock the full potential of RINGER and enable it to scale to larger Chain-of-Thought (CoT) datasets, we propose fine-tuning a compact, efficient model that learns to perform rule merging directly. By using Direct Preference Optimization (DPO) to align this smaller model with high-quality rule merges, we aim to preserve RINGER's explainability and generalization benefits while improving efficiency. This integration not only reduces compute requirements by over $7\times$ but also sets the stage for scaling RINGER across broader domains and problem distributions.

Method Our method follows a two-step framework. First, we fine-tune a lightweight rule-merging model using DPO. This process unfolds in two phases: Phase I focuses on merging pairs of Chains-of-Thought (CoT–CoT), while Phase II extends this to merging an existing rule with a third CoT (Rule–CoT). In phases I, a powerful pretrained LLM generates multiple candidate merged rules, including a special "[NO MERGE]" option. Each candidate is scored using a fixed judge model based on its average Negative Log-Likelihood (NLL) over associated problems, penalized for verbosity. All pairwise comparisons of candidates are turned into preference tuples, which are used to fine-tune a smaller model with LoRA. The resulting Phase II merger model replaces the original merger model in the RINGER system. During inference, the model proposes a single merged rule, which is evaluated against original CoTs using NLL across multiple judge models. The rule is accepted into the database only if it improves generalization under three predefined filters. This approach enables scalable rule training with reduced computational cost.

Implementation We used the OpenR1-Math-220k dataset and constructed a 350-problem training set and 70-problem test set focused on competition-level algebra questions. For DPO training, we generated preference data in two phases: CoT–CoT merging and Rule–CoT merging, yielding 10,762 total preference tuples. Candidate merges were produced using Qwen3-32B (Phase I) and scored with Qwen3-14B using a reward function combining NLL and a length penalty. We fine-tuned a Qwen3-4B model via DPO with LoRA. The final Phase II merger model replaced the original large merger in the RINGER pipeline. Inference-time rule evaluation was done using a judge ensemble of Qwen3-14B and Phi-4 to ensure cross-model robustness.

Results Our analysis reveals three key qualitative improvements from DPO fine-tuning. First, the fine-tuned Qwen-3-4B model generates 50% fewer singleton rules (103 vs. 199) and more multi-CoT rules (e.g., 21 rules covering 3 CoTs vs. 10), indicating stronger generalization and reuse capabilities. Second, with shorter and more predictive rules. Compared to the 32B RINGER model and pre-trained conterpart, the fine-tuned Qwen-3-4B model reducing the longest rule by 39% (1798 vs. 2927 tokens) and lowering average NLL (8.12 vs. 8.70) respectively. Third, It completes all training problems without Out Of Memory (OOM) exception (vs. 28 failures for the 32B model) and maintains comparable accuracy (44/70 vs. 41/70) while requiring over $7\times$ less compute.

Discussion Beyond the quantitative improvements, we observe that many generated rules include explicit token or character counts despite never being prompted to do so. This behavior indicates that the fine-tuned model has an implicit understanding of conciseness as a quality signal. This suggests that the model has effectively learned from the reward design. However, two limitations remain. First, although "[NO MERGE]" was included as an abstention option during training, the model never outputs it, indicating that relative preference training (as used in DPO) may be insufficient to enforce abstention behavior. Second, we observe occasional overgeneralization, where the model merges conceptually unrelated CoTs into a single divergent rule. These incoherent merges reduce explainability and make such rules difficult to further merge and unreliable during inference, posing a major limitation to scaling the framework across large and diverse CoT dataset.

Conclusion This work demonstrates that a DPO fine-tuned Qwen-3-4B model can effectively replace a much larger 32B rule merger in the RINGER framework, achieving comparable accuracy (44/70 vs. 41/70) while reducing computational power by over $7\times$. This breakthrough addresses RINGER's primary bottleneck, its rule database construction cost. This, in turn, enables scaling to large CoT datasets and evaluating generalization on benchmarks like MATH-500 and AIME. Qualitative analysis confirms that the fine-tuned model produces more concise and generalizable rules. To further improve RINGER framework, future work should filter out mismatched problem types during merging and incorporate DPO with supervised fine-tuning (SFT) to explicitly teach abstention behavior in unproductive cases. These enhancements will support the construction of cleaner, more explainable rule databases for generalizable mathematical reasoning.

Reinforce Rule Merging for Rule-Guided LLM Reasoning

Yifan Zhang
Department of Computer Science
Stanford University
yzh123@stanford.edu

Abstract

In the previous project RINGER, we present a framework for distilling abstract reasoning rules from diverse chain-of-thought (CoT) trajectories and applying them via retrieval-augmented generation. While direct preference methods like Direct Preference Optimization (DPO) and Group Relative Policy Optimization (GRPO) have recently been used to incentivize Large Language Models (LLMs) to produce CoTs before answering, the cost of collecting high-quality reasoning data and the difficulty of enhancing reasoning generalization and robustness remain major challenge to their broader applicability. Inspired by human abstract reasoning, we first prompt a large LLM to merge multiple CoTs into concise rules that minimize negative log-likelihood (NLL). We then curate these low-NLL rules into preference pairs and train a compact "rule-merger" model using DPO, which reframes alignment as a stable, closed-form classification loss. At inference, this small model can merge CoTs into abstract rules based on the preference, and enhance the efficiency of RINGER framework. On our curated math benchmarks, the DPO-trained rule-merger matches the performance of the original RINGER system while achieving significant improvement in efficiency, demonstrating a practical path toward scalable, generalizable LLM reasoning.

1 Introduction

LLMs have demonstrated emerging reasoning capabilities across various domains, including arithmetic, logic, and symbolic manipulation, essentially unlocked by Chain-of-Thought prompting, where models generate intermediate reasoning steps before providing final answers. Recently, researchers have used reward or preference-based methods, such as DPO [2] or GRPO [3], to push the model to generate a thinking CoT before generating the response. This research direction has presented promising performance [4, 5]. However, collecting reasoning data incurs a significant cost, and the models still struggle to generalize to multiple domains. Inspired by human cognition, we studied whether these reasoning CoT data can be merged into more abstract rules that can be applied to multiple types of questions. Then, these merged rules are used during LLM reasoning via retrieval augmentation generation. We hypothesize that abstract reasoning rules, distilled from diverse CoT trajectories, can generalize reasoning patterns more effectively than raw chains of thought. In our previous project, we developed a retrieval-based method, RINGER [1], utilizing abstract rules and achieved encouraging results in math reasoning. However, merging rules requires extensive searching time to find optimal rules with low NLLs, and thus, the method is hard to scale up.

To address these challenges, we utilize the DPO method to train a small model that learns the rule-merging preference, enabling us to scale the RINGER system in advance. DPO reframes alignment as a closed-form classification loss, eliminating the need for explicit reward modeling and sampling during training. DPO has proven competitive with PPO-based RLHF in controlling

generation quality and alignment, while being far more stable and computationally efficient. Here, our goal is to train a small model that focuses on merging reasoning CoTs, which can generate answers with lower NLLs. Then, the small model is used for searching for better reasoning rules.

Our DPO training method proceeds in three stages: (1) prompt a large LLM to merge multiple CoTs into concise abstract rules that minimize NLL, (2) extract and curate these rules with lower NLLs and length penalty to construct preference pairs, and (3) train a compact "rule-merger" model with DPO to reproduce rule merging behavior. At inference, this small model retrieves and applies rules to guide QA, combining the interpretability of symbolic rules with the flexibility of neural networks.

Empirical results on our curated math benchmarks demonstrate that our DPO-trained rule-merger achieves comparable performance to the original RINGER model, while offering more than 7 times better efficiency.

2 Related Work

Reinforcement Learning from Human Feedback (RLHF) has been instrumental in aligning LLMs to human preferences, typically by training a reward model on ranked outputs and fine-tuning via PPO [6]. Despite its success in chat and summarization, RLHF requires careful reward-model fitting and hyperparameter tuning, often leading to instability. DPO addresses these issues by deriving an analytical optimal policy under a preference-based loss, enabling direct fine-tuning of the LLM with a simple classification objective [2]. Recent extensions include Calibrated DPO (Cal-DPO) [7], which further refines reward scaling for improved alignment, and a comprehensive survey categorizing DPO variants across data strategies, learning frameworks, and constraint mechanisms.

Chain-of-Thought (CoT) prompting [8] elicits intermediate reasoning by demonstrating step-by-step solutions in few-shot examples, leading to substantial improvements on math benchmarks (e.g., PaLM 540B achieved 57% on GSM8K). A recent meta-analysis [9] confirms CoT's outsized benefits for math and logic tasks while noting diminishing returns on purely linguistic tasks. Variations such as Self-Consistency aggregate multiple CoT samples to improve robustness, while Tree-of-Thought explores branching reasoning paths. Retrieval-augmented Generation (RAG) integrates external knowledge, such as textbook snippets, to ground LLM responses and reduce the likelihood of hallucinations. In educational contexts, RAG improves the trade-offs between preference and correctness for middle-school math QA. Other work investigates multi-step retrieval and backtracking corrections to enhance multi-hop reasoning in QA systems [10]. Many researchers have also combined CoT with reward and preference training to create reasoning models [11]. Despite this progress, collecting reasoning data requires considerable effort and is challenging to generalize across diverse domains in a single model. Our approach utilizes DPO to train a small model to learn how to merge CoT into more generalized and abstract rules, enabling these rules to better guide the model in generating the correct answers.

3 Preliminary Work

Our previous work, RINGER [1], introduced a novel framework to enhance the reasoning capabilities of LLMs by applying generalized rules extracted from CoTs. Unlike traditional inference-time reasoning or reinforcement learning (RL)-based tuning, RINGER provides a transparent and data-efficient mechanism for guiding model reasoning while reducing inference cost and improving explainability.

As shown in Figure 1, RINGER operates in two main phases: (1) Rule Generalization, which transforms CoT explanations into abstract, reusable rules; and (2) Rule-Based Inference, which retrieves and applies those rules to guide model reasoning at inference time.

RINGER's rule generalization phase consists of four components: (1) a **Data Formatter**, which standardizes both problem statements and CoTs into structured formats for consistency and conceptual retrieval; (2) a **Rule Database**, which stores generalized rules indexed by problem type and tracks rule performance via average NLL; (3) a **Rule Merger**, which uses an LLM to combine retrieved rules and new CoT into candidate generalized rules; and (4) a **Rule Selector**, which scores and filters these candidates using multi-model NLL evaluations to retain only high-quality generalizations.

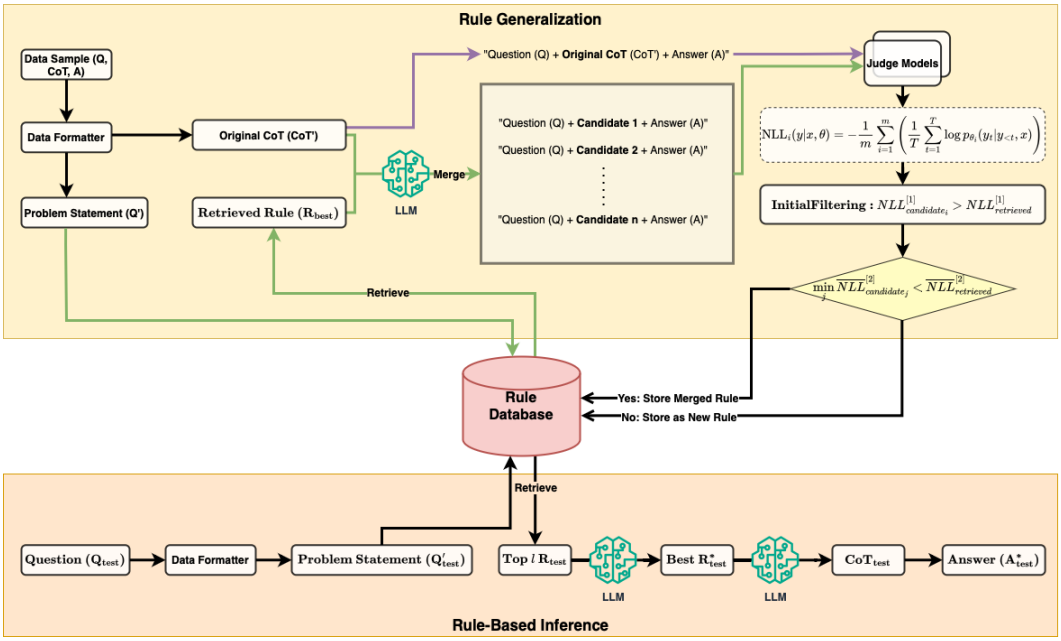

Figure 1: Overview of **R**easoning at **IN**ference time with **GE**neralized **R**ules (RINGER). It involves two main phases: Rule Generalization and Rule-Based Inference.

During inference, RINGER applies two components to solve new problems. The **Rule Retriever** reformats the test query and retrieves the most conceptual relevant rules from the database. The top-ranked rule is passed to the **Rule-Guided Reasoner**, which prompts an LLM to generate a structured reasoning trace and final answer, guided by the selected rule's steps and heuristics.

RINGER demonstrated encouraging empirical performance and interpretability advantages over standard prompting baselines. On a test set of 70 competition-level algebra problems, it achieved 58.6% accuracy, outperforming all benchmarks, while reducing output token length by over $4\times$, highlighting its efficiency. Rule generalization successfully consolidated reasoning patterns, with several rules reused across up to 10 problems, confirming high cross-problem applicability. Moreover, RINGER enhanced explainability, as each rule is transparent and editable. While RINGER framework is promising, merging CoTs into rules requires heavy computational time and power. To address this issue, in this project, we utilize DPO training to fine-tune a small model that merges CoTs based on NLL preference.

4 Method

Our method consists of a two-step framework designed to fine-tune a rule merger using DPO and subsequently integrate the fine-tuned rule merger into the RINGER system with some modifications. As illustrated in Figure 2, the framework includes:

- **DPO-Based Rule Merger Fine-Tuning**
- **Integration with the RINGER Framework**

Below, we describe each step of the framework in detail.

4.1 DPO-Based Rule Merger Fine-Tuning

Our DPO-based fine-tuning step consists of two sequential phases. Phase I focuses on training the model to merge pairs of CoTs into generalized rules. Phase II builds on the model from Phase I, enabling it to handle both CoT-CoT and Rule-CoT merging. Together, these two stages fine-tune the model to develop broader rule generalization capabilities: the **Phase I Merger Model** is used as the starting point for Phase II, and the final **Phase II Merger Model** is then integrated into the RINGER framework.

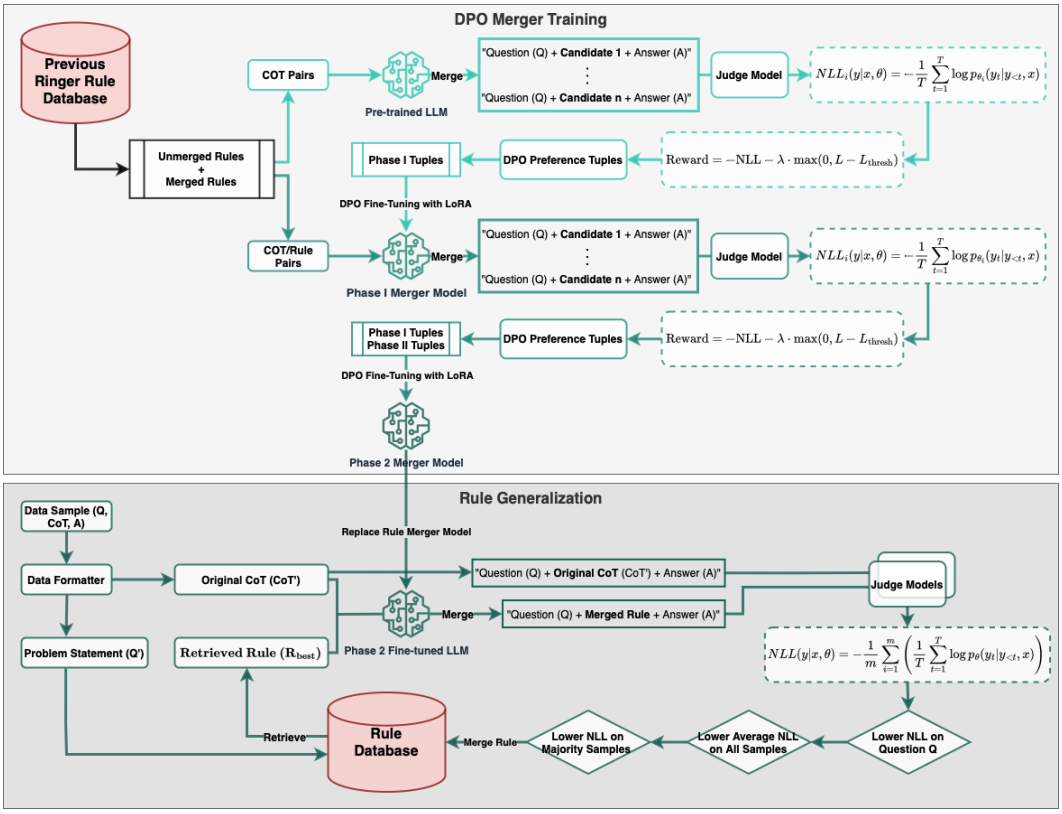

Figure 2: Overview of DPO-Based Rule Merger Fine-Tuning and its Application in RINGER

4.1.1 Candidate Rules Generation. From the Rule Database constructed in preliminary work, we extract training samples in two phases to support DPO fine-tuning. The goal is to construct a dataset for fine-tuning a rule-merging model that can synthesize generalized reasoning rules from multiple CoTs. In both phases, the model that generates rule candidates is prompted using a structure template designed to encourage conciseness and to allow abstention via a special token "[NO MERGE]" when a merge is unproductive.

In *Phase I*, we construct a balanced dataset of CoT pairs that are either (1) derived from the same underlying rule (potentially mergeable), or (2) from unrelated reasoning paths (potentially unmergeable). This 50/50 mix enables the model to learn when to merge and when to perform no merge. For each CoT pair $[(Q_1, \text{CoT}_1, A_1), (Q_2, \text{CoT}_2, A_2)]$, we apply a powerful pre-trained LLM with strong generalization capabilities to synthesize n candidate generalized rules $\{\text{Rule}_i\}_{i=1}^{n}$. Each candidate aims to express a generalized reasoning pattern that supports solving both Q_1 and Q_2.

In *Phase II*, we build on the CoT–CoT (Phase I) merging setup by introducing a third CoT to support further generalization. Specifically, we select rule groups from the Rule Database that were previously created by merging at least three distinct CoTs. For each group, we enumerate all CoT pairs and apply the **Phase I Merger Model** to synthesize a candidate rule $([Q_1, Q_2], \text{Rule}_{\{1,2\}}, [A_1, A_2])$. We then pair this rule with a third CoT (Q_3, CoT_3, A_3) from the same group (not used in the initial merge), forming an input pair of the form $[([Q_1, Q_1], \text{Rule}_{\{1,2\}}, [A_1, A_2]), (Q_3, \text{CoT}_3, A_3)]$. Using the fine-tuned **Phase I Merger Model**, we generate n candidate rules $\{\text{Rule}_i\}_{i=1}^{n}$ aimed at generalizing reasoning across all three questions $[Q_1, Q_2, Q_3]$.

4.1.2 DPO Rule Selection. Each candidate rule $\{\text{Rule}_i\}_{i=1}^{n}$ is evaluated by its ability to generate correct answers across the relevant question set. For each question, we format the input prompt as:

```
Question: {Q}\nChain of Thought: {Rule}\nFinal Answer:
```

Let x_j be the prompt constructed from Q_j and Rule_i, and let y_j be the ground truth answer of length T_j for $j = 1, 2$ in Phase I and $j = 1, 2, 3$ in Phase II. The average NLL of the rule is computed as:

$$\text{NLL}_i = -\frac{1}{k}\sum_{j=1}^{k}\frac{1}{T_j}\sum_{t=1}^{T_j}\log p_\theta(y_{j,t}|y_{j,<t}, x_j),$$

where $k = 2$ in Phase I and $k = 3$ in Phase II, and θ is the fixed judge model.

To discourage overly verbose rules, we apply a length-aware reward function:

$$R_i = -\text{NLL}_i - \lambda \cdot \max(0, L_i - L_{\text{thresh}}),$$

where L_i is the character length of Rule$_i$, L_{thresh} is a predefined length threshold, and λ is a length penalty coefficient. This reward function encourages concise, reusable rules.

We also compute a baseline reward corresponding to the special abstention candidate "[NO MERGE]", where no rule merging is performed. To compute it, we format the CoTs using the same prompt structure as candidate rules and evaluate their weighted average NLL. In *Phase I*, we compute it as:

$$\text{NLL}_{\text{baseline}} = -\frac{1}{2}\left(\frac{1}{T_1}\sum_{t=1}^{T_1}\log p_\theta(y_{1,t}|y_{1,<t}, x_1^{\text{CoT}}) + \frac{1}{T_2}\sum_{t=1}^{T_2}\log p_\theta(y_{2,t}|y_{2,<t}, x_2^{\text{CoT}})\right)$$

In Phase II, we reuse the prompts x_1, x_2 generated from Rule$_i$ and compute the third using the original CoT:

$$\text{NLL}_{\text{baseline}} = -\frac{1}{3}\left(\sum_{j=1}^{2}\frac{1}{T_j}\sum_{t=1}^{T_j}\log p_\theta(y_{j,t}|y_{j,<t}, x_j) + \frac{1}{T_3}\sum_{t=1}^{T_3}\log p_\theta(y_{3,t}|y_{3,<t}, x_3^{\text{CoT}})\right)$$

The corresponding baseline reward $R_{\text{baseline}} = -\text{NLL}_{\text{baseline}} - \lambda \cdot \max(0, L_i - L_{\text{thresh}})$ is used for the "[NO MERGE]" candidate.

All $n + 1$ candidates (including "[NO MERGE]") are scored and ranked by their reward values. To construct DPO preference data, we enumerate all $\binom{n+1}{2}$ unordered candidate pairs. For each pair (i, j), if $R_i > R_j$, we record a preference tuple:

$$\texttt{chosen} = \text{Rule}_i, \quad \texttt{rejected} = \text{Rule}_j$$

This ensures the model is trained to prefer concise, high-quality rules and to abstain when merging is unproductive, with the resulting preference tuples used for downstream DPO fine-tuning.

4.1.3 DPO Training with Preference Pairs. Using the collected preference tuples, we fine-tune a smaller rule-merging model via DPO. For each prompt and preference tuple $(\texttt{chosen}, \texttt{rejected})$, the DPO objective encourages the model to assign higher likelihood to the preferred output, enabling it to learn when and how to merge reasoning traces effectively.

We implement fine-tuning using Low-Rank Adaptation (LoRA), a parameter-efficient method that inserts trainable low-rank matrices into selected layers of the model. Only these injected parameters are updated during training, while the base model remains frozen—allowing for efficient and scalable adaptation.

In *Phase I*, we first use a powerful, pre-trained LLM to generate candidate rules for CoT–CoT merges. The resulting preference tuples are used to fine-tune a smaller model from scratch, producing the **Phase I Merger Model**, a lightweight rule merger capable of synthesizing generalized rules from two reasoning traces.

In *Phase II*, we reuse the fine-tuned **Phase I Merger Model** to generate candidate merges between an existing rule and a third CoT. These new preference tuples, combined with those from Phase I, are then used to fine-tune a model from scratch, equipping it with the ability to handle increasingly abstract Rule–CoT merges. The resulting **Phase II Merger Model** is then integrated into the RINGER framework.

The following pseudocode 1 outlines the overall DPO merger training process:

Algorithm 1 DPO Fine-Tuning for Rule Merging

Inputs: CoT-CoT pairs or Rule-CoT pairs from rule database
LLM: Rule-merging model (pre-trained or fine-tuned)
JudgeModel: NLL-based evaluation model
n: Number of candidate merges per example
Z: Number of training samples

for $i = 1$ **to** Z **do**
  Sample (Q_1, CoT_1, A_1), (Q_2, CoT_2, A_2) from rule database ▷ Phase I
  or **construct** $([Q_1, Q_2], \text{Rule}_{\{1,2\}}, [A_1, A_2])$, (Q_3, CoT_3, A_3) ▷ Phase II
  Use LLM to generate n candidate merged rules $\{\text{Rule}_i\}_{i=1}^{n}$
  Append "[NO MERGE]" as the baseline candidate
  for each candidate rule Rule_i **do**
    Compute average NLL over question-answer pairs
    Compute reward $R_i = -\text{NLL}_i - \lambda \cdot \max(0, L_i - L_{\text{thresh}})$
  end for
  Rank all $n + 1$ rules by reward
  for each pair (j, k) among $n + 1$ rules **do**
    if $R_j > R_k$ **then**
      Store preference tuple $(\texttt{prompt}, \texttt{chosen} = \text{Rule}_j, \texttt{rejected} = \text{Rule}_k)$
    end if
  end for
end for
Train the LLM using DPO objective on all collected preference tuples

4.2 Integration with the RINGER Framework

In this step, we modify the RINGER system to leverage the smaller, fine-tuned **Phase II Merger Model** for efficient and accurate inference. While the overall structure of the RINGER Rule Generalization pipeline remains the same, we update the Rule Merger and Rule Selector components to reflect the capabilities of the **Phase II Merger Model**.

4.2.1 Rule Merger. For each new question-CoT-answer tuple (Q, CoT, A), we replace the large, pre-trained LLM previously used for rule merging with the fine-tuned **Phase II Merger Model**. This smaller model is trained specifically for rule generalization and has shown strong performance on the task. Instead of generating n candidate rules as in the original RINGER system, we now generate only one candidate rule due to the high reliability of the fine-tuned model.

4.2.2 Rule Selector. Like the original RINGER system, we evaluate both the merged candidate rule and the original CoT using judge models. We construct prompts in the same format:

```
Question: {Q}\nChain of Thought: {Rule}\nFinal Answer:
```

The average NLL across m judge models is computed as:

$$NLL(y|x, \theta) = -\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{T} \sum_{t=1}^{T} \log p_{\theta_i}(y_t | y < t, x) \right)$$

To ensure the quality of rule generalization, we apply three filters before accepting a merged rule into the Rule Database:

1. The merged rule must have a lower NLL than the original CoT on the new question.
2. The merged rule must have a lower average NLL than the existing rule across its previously associated samples.
3. More than half of the previously associated samples must show reduced NLL under the merged rule.

If the merged rule passes all three filters, it replaces the retrieved rule in the Rule Database. Otherwise, the reformatted new CoT is stored as a new rule. This adjustment enables efficient training while preserving or improving the quality of rule generalization.

5 Experimental Setup

5.1 Data

5.1.1 Preliminary Work. In our preliminary work, we utilized the `open-r1/OpenR1-Math-220k` dataset [12], which contains 220,000 mathematical problems, each accompanied by 2–4 reasoning traces generated by DeepSeek R1 [4] and verified by Math Verify [13]. From this corpus, we selected 5,000 Algebra problems sourced from Olympiad competitions, each with exactly two correct and verified traces. These samples, represented as (Q, CoT, A), form our working dataset. We randomly chose 70 problems as our test set. Due to computational limitations, we did not train on the full remaining 4,930 problems. Instead, for each test problem, we identified its 5 most relevant problems from the remaining pool, resulting in a targeted 350-example training set. This subset was used to build an initial Rule Database in our preliminary work.

5.1.2 DPO-Based Rule Merger Fine-Tuning. We constructed preference data in two stages to support DPO fine-tuning of the rule-merging model.

5.1.2.1 Phase I. We focused on CoT–CoT merging. We began with the RINGER Rule Database, which contains 113 unmerged rules (each linked to a single problem) and 64 merged rules (each covering two or more problems). From this database, we sampled 430 potentially mergeable pairs and 430 potentially unmergeable pairs. These 430 pairs per category were split into 380 training pairs and 50 test pairs. For each pair, we generated 4 candidate merged rules using a large pretrained LLM with the updated prompt A.1, along with a "[NO MERGE]" baseline. The candidates were scored using the reward function, and all $\binom{4+1}{2} = 10$ pairwise preferences were recorded. This yielded 2,530 preference tuples for Phase I training.

5.1.2.2 Phase II. We extended this setup to Rule–CoT merging. We selected 21 merged rules from the Rule Database, each constructed from at least 3 distinct CoTs. For every merge trial, 4 candidate rules and one "[NO MERGE]" baseline were evaluated using the same reward function, and 10 pairwise preference tuples were recorded. This produced 8,232 preference tuples for Phase II.

Together, Phases I and II yielded a total of 10,762 preference tuples, which were used to fine-tune the final Phase II merger model.

5.1.3 Integration with the RINGER Framework. To evaluate the impact of the Phase II merger model on end-to-end performance, we removed the original RINGER training and test sets and re-sampled 70 test problems and 350 training problems from the remaining pool of 4,580 Algebra problems, using the same procedure as in the original RINGER evaluation.

5.2 Evaluation Setting

To assess the effectiveness of the fine-tuned **Phase II Merger Model** and its integration into the RINGER framework, we adopt and extend the original RINGER. Specifically, we evaluate the model's performance on a newly sampled test set of 70 problems to ensure a fair and controlled comparison. Our evaluation focuses on the following four dimensions:

Answer Accuracy: We extract the model's predicted final answer and compare it against the ground-truth label. Each response is manually reviewed to ensure correctness. Our primary requirement is that the integration of the DPO-trained merger must not degrade accuracy.

Rule Consolidation Rate: We analyze the distribution of how many CoTs are consolidated into generalized rules. A meaningful rule-merging mechanism should actively merge similar CoTs. If most rules remain unmerged, the RINGER framework provides limited generalization benefits.

Rule Conciseness: We compute the number of output tokens of the longest generated rule. This reflects the conciseness and interpretability of the rules. With our reward function, we expect the fine-tuned model to not produce longer, more verbose reasoning steps.

Rule Quality: To evaluate generalization quality, we compute the weighted average NLL of all rules on associated problems. Lower NLL indicates stronger alignment between the rule and the solution. This also aligns with our DPO training objective, which favors rules with predictive alignment.

Together, these metrics provide a comprehensive view of both functional performance and structural quality, allowing us to assess whether the **Phase II Merger Model** offers meaningful improvements over the original RINGER baseline.

5.3 Experimental details

Below, we provide the implementation details of our DPO fine-tuning and its application in RINGER, which follows the approach design illustrated in Section 4.

5.3.1 DPO Fine-Tuning.

5.3.1.1 Phase I. We used the large pretrained model **Qwen3-32B** [14] from Alibaba, with maximum token limit of 22,000, temperature of 0.1, and top-k of 50, to generate 4 candidate merged rules per CoT–CoT pair. Candidate rules were scored using a judge model (**Qwen3-14B**) based on NLL, with a length-aware penalty (length threshold of 7,000 characters and length penalty coefficient of 0.00025) applied.

Using the resulting preference tuples, we fine-tuned a smaller model, **Qwen3-4B**, from scratch via DPO, with a maximum sequence length = 32,768, $\beta = 0.02$, learning rate = $1e$–5, and 10 training epochs. LoRA was applied for efficient tuning with $r = 8$, $\alpha = 16$, dropout = 0.1

5.3.1.1 Phase II. We followed the same setup, but used the fine-tuned **Phase I Merger Model** to generate candidate rules. DPO fine-tuning was then conducted on a new **Qwen3-4B** instance.

5.3.2 Integration into RINGER. For Phase III, we adopted the original framework's settings with one key change: the rule-merging LLM was replaced by our **Phase II Merger Model**. All other components and parameters remained consistent with the original RINGER system. For scoring, we used a judge model ensemble consisting of **Qwen3-14B** and **Phi-4** [15] by Microsoft to improve robustness and cross-model agreement.

6 Results

6.1 Quantitative Evaluation

# of CoTs Merged per Rule	1	2	3	4	5	6	7	8	10+	OOM	NLL	Longest Rule
QWen-2.5-32B[*]	113	43	8	5	1	0	3	1	3	28		2927
QWen-3-4B	199	41	10	3	4	0	1	0	0	0	8.70	1924
QWen-3-4B Fine-Tuned	103	52	21	6	7	1	1	1	0	0	8.12	1798

Table 1: Distribution of CoTs' consolidation into Rules.

	Test Set Accuracy	**Compute Time**	**GPU**
QWen-2.5-32B[*]	41/70 (58.6%) [+]	36 hr	H200 140GB
QWen-3-4B	42/70 (60%)		
QWen-3-4B fine-tuned	44/70 (62.86%)	5 hr	H100 94GB

Table 2: OpenR1 Mathematics Test Set Accuracy Score

[*] Results from preliminary project RINGER.
[+] Accuracy is not directly comparable, as the test sets differ. However, both were randomly sampled from the same problem pool.

Note: All models listed in the tables are used for rule database construction. For accuracy evaluation, we do not modify RINGER's inference pipeline. The model used in RINGER's inference pipeline is Qwen-2.5-14B-Instruct.

6.2 Qualitative Analysis

Rule Consolidation Behavior. As shown in Table 1, the DPO fine-tuned QWen-3-4B model generates approximately 50% fewer singleton rules (103 vs. 199) compared to its pre-trained counterpart, indicating improved consolidation of CoTs into shared rules. Notably, it forms more rules with three or more associated

CoTs (e.g., 21 rules covering 3 CoTs vs. 10), showing better generalization across samples. This consolidation behavior closely mirrors that of the prior QWen-2.5-32B model, despite the fine-tuned model being 8x smaller.

Rule Quality. The DPO fine-tuned model also improves in terms of rule conciseness and NLL. Its longest rule is 1798 tokens, which is shorter than the baseline Qwen-3-4B model's longest rule (1924 tokens), and nearly 39% shorter than the prior QWen-2.5-32B model's longest rule (2927 tokens). The average weighted NLL (8.12) is also lower than the baseline (8.70 from pre-trained QWen-3-4B model), indicating that the reward function successfully encouraged the model to generate higher-quality rules. These trends suggest the model is not only merging more CoTs but doing so in a more consistent and efficient manner.

Robustness and Efficiency. Unlike the 32B baseline, which encountered OOM issues on 28 problems, the DPO fine-tuned QWen-3-4B models successfully completed all 350 training data without exception. Although the test sets differ, making direct accuracy comparisons inappropriate, both were randomly sampled from the same problem pool. In this context, the DPO fine-tuned model achieved 44/70 accuracy on the new test set, compared to 41/70 reported for the RINGER baseline. This suggests that accuracy has not degraded, indicating that the fine-tuned model has the strong capability in rule generalizes despite its smaller size. (Note that The pre-trained Qwen-3-4B achieves 42/70. Because 80% of the CoTs are unmerged, so essentially it just extracts the most relevant CoT and uses that to solve the new problem, similar to our Retrieval_CoT benchmark in the preliminary work, which had 39/70 accuracy). More importantly, it achieves this with significantly improved efficiency: requiring only 5 GPU hours on a 94GB H100 GPU, compared to 36 GPU hours on a 140GB H200 GPU for the 32B baseline. These results demonstrate that with targeted fine-tuning, smaller models can serve as stable and efficient alternatives to much larger models for complex rule-based reasoning.

7 Discussion

Beyond quantitative results, our analysis surfaced several behavioral patterns and limitations in the DPO fine-tuned merger model that offer valuable insights for future development.

First, manual inspection revealed that the model has internalized the length penalty from the reward function. Many generated rules include explicit character or word counts, despite such tokens never appearing in the prompts or training data. This suggests the model has learned to treat conciseness as a proxy for quality, successfully aligning with the reward design. Moreover, because the model's reasoning capability is constrained by token limits, generating more concise rules makes them more effective and efficient to apply during inference and to further merge during training.

Second, while we included "[NO MERGE]" as a special token to discourage inappropriate merges, the model never outputs it. This likely reflects a limitation of DPO, which optimizes relative preferences between outputs rather than teaching absolute behaviors. Since abstaining requires a more categorical shift in generation, DPO alone may not provide strong enough signals to enforce it. To address this, we propose supplementing DPO with supervised fine-tuning (SFT), which can explicitly teach the model when to output "[NO MERGE]" through labeled examples.

Third, although many merged rules were valid and generalized well across structurally similar problems (see Appendix A.2 for an example), we also observed some cases where conceptually different problems were inappropriately merged (see Appendix A.3). The model occasionally produced divergent rules of the form "if A $\rightarrow$ do step X_1-X_2, if B $\rightarrow$ do step Y_1-Y_2", combining unrelated reasoning paths into a single rule. This behavior undermines interpretability and coherence, which are the core goals of the RINGER framework. It suggests that the current training setup may not sufficiently penalize incoherent generalization.

8 Conclusion

This work demonstrates that a DPO fine-tuned Qwen-3-4B model can effectively replace a much larger 32B model in the RINGER framework without compromising performance. Despite being trained and evaluated on data from the same rule database, the smaller model achieves comparable accuracy (44/70 vs. 41/70) while offering 7$\times$ faster computational time and more than 7$\times$ lower compute requirements. This scalability breakthrough addresses RINGER's primary bottleneck, its computational expense in rule construction, and opens the door to applying RINGER to larger and more diverse CoT datasets. This, in turn, allows us to evaluate its generalization and effectiveness on standard mathematical benchmarks such as MATH-500 and AIME. Qualitative analysis further confirms that the fine-tuned model produces concise and generally valid rules, significantly improving consolidation behavior over the pre-trained baseline.

Looking forward, two areas warrant further development. First, to prevent overgeneralized or divergent rules, future experiment should filter out problem pairs involving dissimilar mathematical concepts, ensuring conceptual consistency in rule merging. Second, to improve abstention behavior, DPO training should be augmented with SFT, especially for `[NO MERGE]` cases where rule merging is inappropriate. These refinements will help build

cleaner, more interpretable rule databases and advance the goal of scalable, generalizable reasoning across diverse mathematical tasks.

9 Team Contributions

- **Yifan Zhang**: Yifan Zhang was the primary contributor of this project. He was responsible for the full research and engineering cycle, including preparing and processing the datasets, developing the RL framework for fine-tuning the LLM, implementing the customized reward function (e.g., based on Negative Log Likelihood and rule conciseness), running the model training and experiments, and evaluating the system on mathematical reasoning benchmarks. Yifan was responsible for conducting ablation studies, analyzing model performance, and producing all written deliverables, including the proposal, milestone, poster, and final project report.
- **Kezhen Chen**: Kezhen Chen was this project's external mentor, offering high-level feedback on RL framework design, reviewing experimental methodology, and providing technical advice on RL and LLM fine-tuning.

Changes from Proposal No change from proposal

References

[1] Yifan Zhang and Kezhen Chen. Ringer: Reasoning at inference time with generalized rules. `https://drive.google.com/file/d/1sn8xVPNJgzcH2Lt6mDOprrXRC5p1PnNu/view?usp=sharing`, 2025. Stanford CS224N Custom Project Report.

[2] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024.

[3] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

[4] 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.

[5] Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown, November 2024.

[6] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[7] Teng Xiao, Yige Yuan, Huaisheng Zhu, Mingxiao Li, and Vasant G Honavar. Cal-dpo: Calibrated direct preference optimization for language model alignment, 2024.

[8] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[9] Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning, 2025.

[10] Xinjie Zhao, Fan Gao, Xingyu Song, Yingjian Chen, Rui Yang, Yanran Fu, Yuyang Wang, Yusuke Iwasawa, Yutaka Matsuo, and Irene Li. Reagent: Reversible multi-agent reasoning for knowledge-enhanced multi-hop qa, 2025.

[11] Xuan Zhang, Chao Du, Tianyu Pang, Qian Liu, Wei Gao, and Min Lin. Chain of preference optimization: Improving chain-of-thought reasoning in llms, 2024.

[12] open r1. Openr1-math-220k. https://huggingface.co/datasets/open-r1/OpenR1-Math-220k.

[13] huggingface. Math-verify. https://github.com/huggingface/Math-Verify.

[14] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

[15] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 technical report, 2024.

A Appendix

A.1 DPO Merge Rule Prompt

You are provided with two generalized problem-solving plans:

Retrieved Theory: a plan summarized from multiple samples or a plan derived from a single data sample.
Existing Theory: a plan derived from a single data sample.

Retrieved Theory: {}
Existing Theory: {}

Your task is to **merge these two plans into one improved, structured theory that is strictly better than either original plan.** The merged theory should be **comprehensive, logically structured, and adaptable** to different variations of the problem.

Key Merging Principles:
1. **Check Domain Alignment**: Only merge these plans if they address the **same or closely related** problem types (e.g., both are Quadratic Equations, both are Logarithmic Equations).
2. **Branching for Different Domains**: If the plans solve **different** problem types (e.g., Algebraic Factorization vs. Trigonometric Identities), keep them in **separate sub-plans** rather than forcing a single unified flow.
3. **Avoid Over-Generalization**: If no meaningful overlap is found, do not merge. If only partial overlap exists, unify **only** the steps that actually apply to both domains.
4. **Introduce Decision Points**: If multiple methods exist for the same type of problem, specify **when** to use each approach.
5. **Concrete Steps**: For the domain you decide to merge, unify steps with **specific details**, avoiding vague or overly broad instructions.

6. **Step Limit**: Ensure the total number of steps **does not exceed 20**. If a merged plan exceeds 20 steps, **prioritize key steps** and remove redundant details. The returned theory should **NOT** exceed 1200 words
7. **Prefer Conciseness:** The merged rule should be **no more than 5000 characters**. Any extra characters beyond this limit detract from effectiveness—keep it as brief as possible while still correct and complete.
8. **Reject Unproductive Merges:** If merging the two rules does not lead to a clearer, more effective, or logically coherent unified plan, do not force the merge. Instead, output exactly "[NO MERGE]"

Format:
Mathematical Category: [e.g., Algebra, Geometry, Probability, Calculus]
Solution Strategy: [Merged strategy, incorporating key aspects of both theories]
Mathematical Problem Scope: [Clearly state what type of problem this theory helps solve]
General Heuristic: [Summarized core approach that underlies the merged solution]

Core Steps
Each step should clearly state:
1. **What action is performed**
2. **Why it is necessary**
3. **Alternative approaches, if applicable**

Merged Steps:
<Step1> [Action Name]
- [What to do]
- [Why this step is necessary]
- **Alternative Approach (if applicable)**

<Step2> [Action Name]
- [What to do]
- [Why this step is necessary]
- **Alternative Approach (if applicable)**

Decision Logic for Alternative Methods:
- **If Condition A $\rightarrow$ Use Method 1**
- **If Condition B $\rightarrow$ Use Method 2**

Dependencies: [Define step relationships clearly]

Edge Cases Special Considerations:
- [Describe any special conditions where this theory needs modification]

Return only the newly formatted theory. Do not include explanations or extra comments.

A.2 Valid Rule Merging Example

original_questions:
1. **5.1.** From point A to point B, which are 10 km apart, a car departed at 7:00. After traveling $\frac{2}{3}$ of the distance, the car passed point C, from which a cyclist immediately set off for point A. As soon as the car arrived at B, a bus immediately departed from B in the opposite direction and arrived at A at 9:00.

At what distance from B did the bus catch up with the cyclist, if the cyclist arrived at point A at 10:00 and the speed of each participant in the movement was constant?

2. Given that location C is the midpoint between locations A and B. At 7:00 AM, car A departs from A heading towards B, while car B and car C depart from B and C respectively, heading towards A. When car A and car C meet, car B has completed exactly $\frac{3}{8}$ of the total distance.

Car C arrives at A at 10:00 AM, and at 10:30 AM, when car B reaches A, car A is still 84 kilometers away from B.

What is the distance between A and B in kilometers?

rule:
Mathematical Category: Algebra
Solution Strategy: Relative Motion and System of Equations for Multi-Object Motion Problems

[TRUNCATED]

Core Steps
Step 1: Define Variables and Known Quantities
What to do: Assign variables to unknowns (e.g., total distance D, speeds of objects). Note all given numerical and relational data.
Why: Establishes a clear framework for modeling.
Alternative: Use substitution or elimination directly if relationships are highly structured.

Step 2: Establish Equations Based on Motion Relationships

[TRUNCATED]

Dependencies Step 1 → Step 2 → Step 3 → Step 4 → Step 5 → Step 6

[TRUNCATED]

Final Output Word Count: 498
Final Character Count: 1,495

A.3 Invalid Rule Merging Example

original_questions:
1. Given the sequence $\{a_n\}$, where $a_1 = 1$, $a_2 = \frac{1}{4}$, and $a_{n+1} = \frac{(n-1)a_n}{n-a_n}$ $(n = 2, 3, 4, \cdots)$. (1) Find the general term formula for the sequence $\{a_n\}$; (2) Prove that for all $n \in \mathbb{N}_+$,

$$\sum_{k=1}^{n} a_k^2 < \frac{7}{6}.$$

2. (a_n) is an arithmetic progression with a common difference of 1. It is known that S_{2022} is the smallest among all S_n (less than the sum of the first n terms for any other value of n). What values can the first term of the progression take?
rule:
Mathematical Category: Sequences, Series, and Calculus
Solution Strategy: Pattern Recognition, Inductive Proofs, Series Comparison, and Calculus Optimization

[TRUNCATED]

Core Steps
Step 1: Compute Initial Terms and Observe Patterns
What to do: Calculate the first few terms using the recursive formula.
Why: Identifies potential patterns or closed-form expressions.
Alternative: Use generating functions or characteristic equations.

Step 2: Hypothesize General Formula

[TRUNCATED]

Decision Logic for Alternative Methods
If Condition A → Use Method 1:
Sequence is recursive with a clear pattern ⇒ Use pattern recognition and induction.
If Condition B → Use Method 2:
Minimize sum of arithmetic progression ⇒ Use calculus and optimization.
If Condition C → Use Method 3:
Bound sum of squares ⇒ Use comparison test or integrals.

Dependencies
Step 1 → Step 2 → Step 3 → Step 6
Step 7 → Step 8 → Step 9 → Step 10 → Step 11

[TRUNCATED]

Final Word Count: 900 Character Count: 5901