Data Diversification Methods In Alignment Enhance Math Performance In LLMs

Anonymous ACL submission

Abstract

While recent advances in preference learning have enhanced alignment in human feedback, mathematical reasoning remains a persistent challenge. We investigate how data diversification strategies in preference optimization can improve the mathematical reasoning abilities of large language models (LLMs). We evaluate three common data generation methods—temperature sampling, Chain-of-Thought prompting, Monte Carlo Tree Search (MCTS), and introduce Diversified-ThinkSolve (DTS), a novel structured approach that systematically decomposes problems into diverse reasoning paths. Our results show that with strategically diversified preference data, models can substantially improve mathematical reasoning performance, with the best approach yielding gains of 7.1% on GSM8K and 4.2% on MATH over the base model. Despite its strong performance, DTS incurs only a marginal computational overhead (1.03×) compared to the baseline, while MCTS is nearly five times more costly with lower returns. These findings demonstrate that structured exploration of diverse problem-solving methods creates more effective preference data for mathematical alignment than traditional approaches.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across a wide range of tasks, but mathematical reasoning remains a particularly challenging domain (Luo et al., 2023; Lightman et al., 2023). While recent work has shown that Reinforcement Learning from Human Feedback (RLHF) (Stiennon et al., 2020) and preference optimization techniques like Direct Preference Optimization (DPO) (Rafailov et al., 2023) can substantially improve LLM performance on general tasks, their application to mathematical reasoning has received less attention.

In standard preference optimization scenarios, datasets typically consist of unmodified preference pairs drawn from human annotations or model-generated evaluations. While such datasets can yield performance improvements (Guo et al., 2024b; Tunstall et al., 2023; Xia et al., 2024) in alignment with human preference, we hypothesize that more structured and diverse preference data can lead to significantly better performance specifically tailored to mathematical reasoning (Liu et al., 2024b).

Our work explores how strategically designed data generation and diversification methods can enhance the effectiveness of preference optimization for mathematical reasoning. We propose several approaches to generate preference data that incorporate diverse reasoning strategies, problem reformulations, and solution methodologies. By leveraging techniques such as Chain-of-Thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022), Monte Carlo Tree Search (MCTS) (Silver et al., 2016; Feng et al., 2023), and specialized thought-reflection mechanisms, we create datasets that expose LLMs to a richer space of mathematical problem-solving strategies during preference optimization.

Among these approaches, we introduce Diversified-ThinkSolve (DTS), a novel structured method that systematically decomposes problems into diverse problem-solving approaches before generating solutions. DTS explicitly separates the thought generation process from solution execution, enabling exploration of multiple problem-solving strategies while maintaining computational efficiency. This approach addresses a fundamental limitation of traditional sampling methods—their inability to systematically explore diverse thinking pathways.

We conduct a comprehensive comparative analysis of these strategies across standard mathematics benchmarks. Our DTS approach yields significant improvements in both GSM8K and MATH over the base model, while incurring only marginal computational overhead. Our findings highlight that structured exploration of analytical approaches creates more effective preference data for mathematical alignment than traditional approaches, and that data quality and diversity can be more crucial than optimizing algorithmic approaches.

2 Background

In this section, we provide the necessary background and information regarding alignment training for LLMs. We start by providing a background on the RLHF process and then we discuss post-training alignment techniques utilized in this paper.

2.1 Reinforcement Learning from Human Feedback

Often after we pre-train a model we want to further adapt it to meet certain needs or specifications (Stiennon et al., 2020; Bai et al., 2022a; Ouyang et al., 2022). Reinforcement Learning from Human Feedback (RLHF) has become a standard approach for aligning large language models with human preferences and values (Christiano et al., 2017; Leike et al., 2018). RLHF emerged as a solution to the challenge of aligning AI systems with human values and preferences when these values were difficult to specify mathematically yet easy to judge. While RLHF requires relatively small amounts of comparison data to be effective compared to other approaches, sourcing high-quality preference data remains an expensive process. This technique has become particularly crucial for LLMs, where it helps guide these powerful systems toward producing outputs that humans find helpful, harmless, and honest (Bai et al., 2022a,b).

The RLHF process typically consists of three stages:

1. **Supervised Fine-Tuning (SFT):** The model is first fine-tuned on demonstrations that exemplify desired behavior, producing a model π^{SFT}.

2. **Reward Modeling:** Human annotators compare model responses, and these comparisons train a reward model $r_\phi(x, y)$ that predicts human preferences. The reward model is trained using maximum likelihood on preference pairs (x, y_w, y_l) using the Bradley-Terry Model (Bradley and Terry, 1952; Plackett, 1975) to model the preference probability.

3. **RL Optimization:** The language model is then optimized further using reinforcement learning, typically with Proximal Policy Optimization (PPO), to maximize the reward while maintaining proximity to the reference model (Jaques et al., 2017, 2020; Schulman et al., 2017).

2.2 Preference Optimization Methods

Recent work has introduced more efficient alternatives to the full RLHF pipeline. Direct Preference Optimization (DPO) (Rafailov et al., 2023) eliminates the need for an explicit reward model and RL training by directly optimizing a policy from preference data:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\big(\beta(r_w - r_l)\big)\right]$$

where r_w and r_l are the log probability ratios of the preferred and dispreferred responses relative to a reference model. This approach has shown comparable or superior performance to RLHF while being more computationally efficient and stable.

More recent methods include Simple Preference Optimization (SimPO) (Meng et al., 2024), which eliminates the need for a reference model while maintaining strong performance:

$$\mathcal{L}_{\text{SimPO}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta(s_w - s_l) - \gamma\right)\right]$$

where s_w and s_l are length-normalized log probabilities, β controls preference signal strength, and γ is a target margin.

We also compare with Odds Ratio Preference Optimization (ORPO) (Hong et al., 2024), which combines supervised fine-tuning with preference optimization through a log odds ratio term, enabling effective alignment without a reference model. ORPO's loss function balances a supervised term for the preferred completion with a preference term based on log odds ratios.

3 Data Diversification Methods

In this section, we describe our proposed data diversification strategies on creating high-quality preference data for fine-tuning and preference optimization.

3.1 Baseline Strategy

Our baseline strategy follows standard practice in preference optimization, generating multiple completions from the base model with only temperature sampling for diversity. During generation, we set the `max_tokens` to 1,024, the `temperature` to 2, `top_p` to 0.75, and `top_k` to 50. We generate 5 completions from the base model π^{SFT} using the following system prompt template:

> "You will be given a math problem. Provide a step-by-step solution, clearly showing all calculations and reasoning. Ensure that each step is explained and justified. After your detailed solution, on a new line, give the final numerical answer in the format: 'Final Answer: [number]'. Do not include any units in the final answer. Double-check your calculations to ensure accuracy."

3.2 Chain-of-Thought Strategy

Chain-of-Thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022) encourages LLMs to generate step-by-step reasoning before producing a final answer. This approach has shown significant improvements in mathematical problem-solving, particularly for complex multi-step problems (Havrilla et al., 2024). For generation, we used OptiLLM's cot-reflection inference proxy to illicit chain of thought reasoning for our model during inference time[1]. This method implements chain-of-thought reasoning with <thinking>, <reflection>, and <output> section tags in the prompt. We set our `temperature` to 0.7 and `max_tokens` to 1,024 to avoid context length issues with increased token counts from chain-of-thought.

3.3 MCTS Strategy

Methods incorporating search algorithms like Monte Carlo Tree Search (MCTS) have shown promise for enhancing mathematical reasoning (Feng et al., 2023; Yao et al., 2023; Liu et al., 2024a). These approaches explore multiple solution paths and can identify effective reasoning strategies through simulation. For this strategy, we leverage MCTS through the OptiLLM inference proxy (codelion, 2024) to systematically explore the solution space[2]. For each mathematical problem, we initialize a dialogue-based MCTS search with the problem as the initial query and a structured solution prompt as the system prompt. We set our `exploration_weight` to 0.2, `num_simulations` to 2, and our `simulation_depth` to 1, which is the default configuration for the MCTS approach, and set `temperature` to 0.7 and `max_tokens` to 1,024 for our generation configuration. At the end, the N (in our case 5) most promising complete solution paths are picked.

This lightweight MCTS approach enables efficient yet effective exploration of the solution space, finding diverse high-quality solutions that may not be discovered through simpler sampling approaches.

3.4 Diversified-ThinkSolve (DTS) Strategy

While the previously described strategies offer certain improvements, they exhibit key limitations in generating truly diverse mathematical reasoning approaches. Temperature sampling produces variations that often follow similar reasoning patterns, and Chain-of-Thought, despite encouraging step-by-step reasoning, tends to converge on a single solution path. MCTS explores alternative branches but incurs substantial computational costs. To address these limitations, we introduce Diversified-ThinkSolve (DTS), a novel strategy specifically designed to generate diverse, high-quality mathematical reasoning paths with minimal computational overhead.

DTS leverages DSPy, a declarative programming paradigm for language models, that enables modular and structured reasoning (Khattab et al., 2023b,a). Unlike traditional prompting approaches that produce variations of the same solution or chain-of-thought strategies that follow a single reasoning flow, DTS explicitly decomposes the mathematical problem-solving process into two distinct phases: multiple approach generation followed by targeted execution. This decomposition enables systematic exploration of the solution space while maintaining reasoning coherence.

We implement DTS through two specialized modules. First, a `ThoughtGenerator` construct generates $N = 5$ distinct reasoning approaches using the following prompt template:

> "Given the math problem: {problem}, provide 5 possible approaches or initial thoughts on how to solve it, including any relevant mathematical concepts, formulas, or techniques that may be applied. Consider multiple perspectives and po-

---

[1] https://github.com/codelion/optillm/blob/main/optillm/cot_reflection.py

[2] https://github.com/codelion/optillm/blob/main/optillm/mcts.py

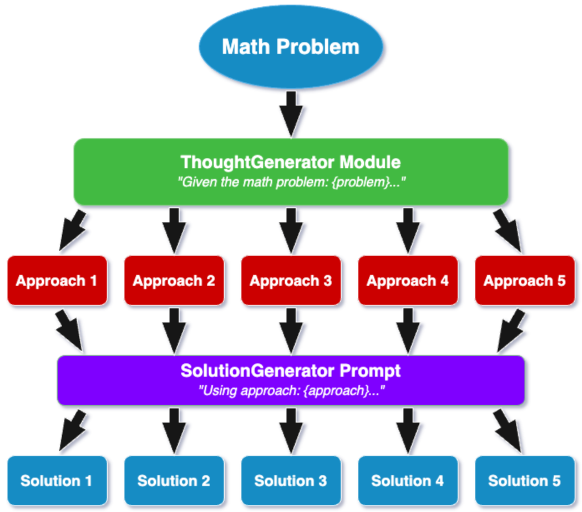

Figure 1: Diversified-ThinkSolve (DTS) modular reasoning pipeline for generating diverse mathematical problem solutions. Each math problem is first processed by a ThoughtGenerator to propose multiple solution approaches. Then utilizing the SolutionGenerator with each approach, we are given multiple solutions, contributing to a diverse set of preference data.

> tential solution paths, and describe each thought in 1-2 sentences.”

Then, for each generated approach, a `SolutionGenerator` module produces a complete solution following that particular reasoning pathway: “Given the math problem: {problem} Using this approach: {approach} Please provide a detailed solution showing all work and steps.”

Figure 1 illustrates this process. By decoupling reasoning approach generation from solution implementation, DTS systematically explores diverse problem-solving strategies while ensuring each solution maintains consistent reasoning flow. The modularity of this approach allows for easy adjustment of the number and type of reasoning paths without modifying the entire pipeline.

A key advantage of DTS over other strategies is its explicit promotion of strategic diversity—it doesn’t merely produce different ways to present the same solution, but fundamentally different problem-solving approaches. This structured diversity creates more informative preference pairs that expose the model to a richer set of mathematical reasoning patterns during alignment.

4 Experiment Setup

4.1 Datasets and Models

We conduct our experiments using the MetaMathQA dataset (Yu et al., 2023) composed of 395k training examples which are all augmented from the training sets of GSM8K and MATH. Since this dataset consists of duplicated problems with distinct queries, we decided to use a deduplicated version containing 13,929 unique mathematical queries and solutions. For evaluation, we use GSM8K’s test set of 1,319 problems and the MATH-500 test subset.

For training, we use *meta-llama/Llama-3.1-8B-Instruct* (Meta AI, 2024) as our base model for all experiments. For scoring completions, we use Nvidia’s *Llama-3.1-Nemotron-70B-Reward-HF* (NVIDIA NeMo Team, 2024), which demonstrated the highest accuracy in our reward model evaluation (Section 4.2).

4.2 Reward Model Selection

We evaluated several candidate reward models from the top models on RewardBench (Lambert et al., 2024) by having them score both model-generated completions and ground truth solutions on the GSM8K test set. We tracked four key metrics: `correct_higher` (model’s correct output received higher reward than ground truth), `correct_lower` (model’s correct output received lower reward than ground truth), `incorrect_higher` (model’s incorrect output received higher reward than ground truth), and `incorrect_lower` (model’s incorrect output received lower reward than ground truth). An effective reward model should minimize `incorrect_higher` cases, which represent instances where incorrect solutions are scored above correct ones.

As shown in Figure 2, *Nvidia’s Llama-3.1-Nemotron-70B-Reward-HF* demonstrated the lowest rate of `incorrect_higher` judgments, with an average inaccuracy rate of 3.11% on the GSM8K test set. We further validated this model on a random sample of 4,919 problems from the harder MetaMathQA dataset, finding a comparable inaccuracy rate of 2.76%. More details can be found in Appendix A.2.

4.3 Preference Data Generation and Filtering

For each data generation strategy described in Section 3, we generated preference data from the MetaMathQA dataset. We applied a "mixed correctness"

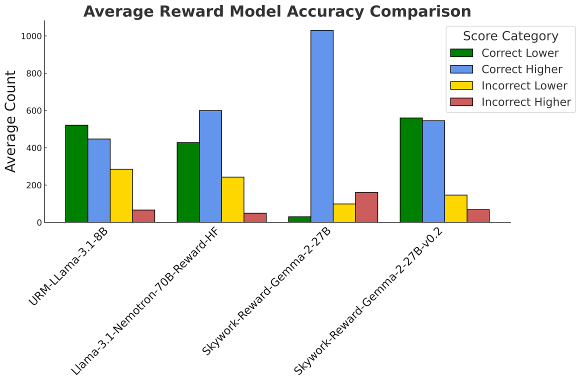

Figure 2: Reward Model Accuracy Comparison. Bars represent average counts of prediction outcomes for different reward models.

Method	GSM8K		MATH	
	Best	Avg	Best	Avg
Llama-3.1-8B-IT	*76.1%*	–	*48.2%*	–
Llama-3.1-70B-IT	85.4%	–	61.6%	–
Llama-3.2-1B-IT	44.9%	–	23.4%	–
Llama-3.2-3B-IT	73.1%	–	44.8%	–
Baseline+SFT	76.7%	74.2%	48.4%	47.7%
Baseline+ORPO	77.6%	76.5%	51.8%	49.1%
Baseline+DPO	77.6%	76.8%	52.2%	49.6%
Baseline+SimPO	80.7%	78.9%	50.8%	48.0%
CoT+SFT	76.7%	73.7%	50.6%	49.0%
CoT+ORPO	77.4%	77.1%	51.2%	48.2%
CoT+DPO	77.6%	74.5%	50.8%	48.1%
CoT+SimPO	77.6%	53.0%	50.4%	48.1%
MCTS+SFT	76.7%	76.6%	48.8%	47.6%
MCTS+ORPO	79.0%	77.9%	50.8%	49.0%
MCTS+DPO	77.8%	77.3%	49.2%	42.8%
MCTS+SimPO	78.0%	59.7%	50.6%	24.2%
DTS+SFT	76.7%	75.7%	49.6%	48.8%
DTS+ORPO	77.2%	76.8%	50.6%	49.7%
DTS+DPO	81.2%	79.3%	**52.4%**	**50.2%**
DTS+SimPO	**83.2%**	**81.5%**	52.0%	49.6%

Table 1: Mathematical reasoning accuracy (%) on GSM8K (0-shot) and MATH-500 across different data generation strategies and preference optimization methods. We report both the best and average performance across 5 epochs for the optimal hyperparameter setting for each fine-tuning method and data generation strategy.

filtering approach, selecting only cases where 2-3 out of 5 model generations were correct, ensuring the model learns to distinguish between correct and incorrect reasoning patterns. We then used our reward model to select the highest-scored correct completion as y_w and the highest-scored incorrect completion as y_l for preference training.

For each strategy, we created preference datasets of comparable size: Baseline (1,097 samples, 30.4% filtered rate from 3,610 problems), DTS (1,293 samples, 17.2% from 7,500 problems), Chain-of-Thought (1,247 samples subsampled from 2,493, 17.9% from the full dataset), and MCTS (1,586 samples subsampled from 3,172, 22.8% from the full dataset). For training consistency, we used comparable dataset sizes across all strategies, sampling half of the subsets from the larger CoT and MCTS datasets.

5 Results

We present results comparing our different data generation strategies across various preference optimization methods.

5.1 Analysis of Data Generation Strategies

As shown in Table 1, DTS consistently outperforms other methods, achieving a 7.1% improvement over the base Llama-3.1-8B-IT model on GSM8K and a 4.2% improvement on MATH. The optimal fine-tuning method varies by benchmark, with SimPO yielding the best results for GSM8K and DPO performing best for MATH.

Baseline Strategy: The standard approach of generating completions with temperature sampling showed moderate improvements over the base model, particularly when combined with SimPO, achieving 80.7% accuracy on GSM8K. This suggests that even simple diversity through temperature sampling can enhance performance. However, this strategy was consistently outperformed by more sophisticated diversification methods except when paired with SimPO, indicating that temperature sampling alone provides insufficient diversity for optimal mathematical reasoning.

Chain-of-Thought Strategy: CoT showed mixed results with significant stability issues, particularly with SimPO where average performance dropped to 53.0% on GSM8K despite reasonable best-case performance (77.6%). Analysis revealed that incorrect CoT responses often contained repetitive patterns and low-quality reasoning, creating preference pairs with extremely poor rejected completions that may have hindered learning.

MCTS Strategy: While MCTS showed promising results with ORPO (79.0% on GSM8K), it exhibited considerable instability with SimPO, where performance degraded substantially across epochs (average 59.7% on GSM8K, 24.2% on MATH). Despite MCTS's systematic exploration capabilities, its high computational cost and inconsistent performance make it less practical than DTS.

DTS Strategy: The DTS thought-based approach demonstrated substantial improvements across all fine-tuning methods, with the highest scores in both benchmarks. By explicitly generating multiple solution approaches before solving problems, this method effectively exposes the model to diverse reasoning paths. The structured exploration of different mathematical strategies appears to provide the model with a richer learning signal during preference optimization. When combined with SimPO, this approach achieved the highest average (81.5%) and best (83.2%) GSM8K scores, while pairing with DPO yielded the best MATH performance (52.4%). This suggests that decomposing mathematical reasoning into distinct thought processes creates more effective preference data for alignment.

Interestingly, the baseline strategy with SimPO outperformed both the CoT and MCTS strategies on average, highlighting that sophisticated data generation methods must be carefully integrated with the appropriate preference optimization technique. The clear winner across both benchmarks is the DTS approach, which consistently produced high-quality, diverse preference data that translated to substantial improvements in mathematical reasoning capabilities.

5.2 Hyperparameter Sensitivity Analysis

Understanding the impact of hyperparameter choices on model performance is crucial when optimizing preference learning for mathematical reasoning. While prior work has explored hyperparameter tuning for general preference learning (Tang et al., 2024), the unique challenges of mathematical reasoning tasks may require different optimal configurations. Additionally, different data generation strategies might interact with hyperparameters in unexpected ways, potentially requiring strategy-specific tuning. We conduct this analysis to identify the most effective hyperparameter configurations for each data generation method and to provide practical guidance for researchers applying preference optimization to mathematical domains.

As shown in Table 2, SimPO consistently demonstrates superior performance across most data generation strategies, particularly with the DTS approach where it achieves remarkable performance (83.2% on GSM8K). The performance advantage of SimPO is particularly pronounced with the DTS strategy, where all hyperparameter configratuions yield strong and stable results, consistently outperforming other fine-tuning methods. Notably, for both CoT and MCTS strategies, the performance margin is more modest, and in the case of MCTS, ORPO actually provides the best results for both GSM8K (79.0%) and MATH (50.8%).

SFT: For supervised fine-tuning, we examined learning rates of 1e-5 and 3e-5, with 2e-5 being the standard default in most SFT implementations. Our results indicate minimal differences between these learning rates on GSM8K performance, with all configurations yielding identical accuracy (76.7%). However, the lower learning rate of 1e-5 consistently produced slightly better results on the more challenging MATH benchmark across all data generation strategies, improving performance by 0.6-2.4%.

ORPO: For ORPO, we found that the standard learning rate of 8e-6 recommended in the original work was excessive for mathematical reasoning tasks, significantly degrading model performance (see Appendix B.3). Our experiments with lower learning rates (5e-7, 2e-7, and 7e-8) revealed distinct optimal configurations for different data generation strategies. For MCTS, higher learning rates performed better, while the other strategies benefited from progressively lower learning rates. The best overall ORPO performance was achieved with MCTS at a learning rate of 5e-7, yielding 79.0% on GSM8K and 50.8% on MATH—the highest scores for any MCTS configuration across fine-tuning methods. In our ORPO ablations, we set the λ weighing parameter by default to 1 which remained constant.

DPO: Following recent findings that lower learning rates are beneficial for reasoning-intensive domains (Shen et al., 2024), we conducted a thorough grid search across learning rates (1e-7, 3e-7, 5e-7, 7e-7) and beta values (0.01, 0.05, 0.1). Our results show that smaller learning rates (e.g., 5e-7) are more suitable for mathematical reasoning, with the optimal configuration varying by data generation strategy. DPO showed particularly strong performance on the MATH benchmark, achieving the highest overall MATH scores for both baseline (52.2%) and DTS (52.4%) strategies, representing

Method	η	β	γ	Baseline		CoT		MCTS		DTS	
				GSM8K	MATH	GSM8K	MATH	GSM8K	MATH	GSM8K	MATH
SFT	1×10^{-5}	—	—	**76.7**	**48.4**	**76.7**	**50.6**	**76.7**	**48.8**	**76.7**	**49.6**
	3×10^{-5}	—	—	76.7	48.2	76.7	48.0	76.7	48.2	76.7	49.0
DPO	7×10^{-7}	0.01	—	75.8	**52.2***	76.0	47.8	76.1	48.2	80.5	50.6
	5×10^{-7}	0.01	—	76.9	51.6	**77.6**	47.6	77.0	**49.2**	**81.2**	50.4
	3×10^{-7}	0.01	—	76.9	50.6	76.8	50.4	77.0	48.4	79.2	**52.4***
	1×10^{-7}	0.01	—	77.6	48.2	77.5	50.2	**77.8**	48.4	77.5	49.2
	3×10^{-7}	0.05	—	76.7	51.6	77.3	50.4	76.9	49.0	78.6	51.2
	1×10^{-7}	0.05	—	**77.6**	51.2	77.0	**50.8**	77.3	48.8	77.7	49.0
	3×10^{-7}	0.1	—	77.2	48.6	76.7	48.8	77.3	48.2	78.2	51.6
	1×10^{-7}	0.1	—	77.2	50.8	77.3	49.2	77.1	48.6	77.3	49.0
ORPO	5×10^{-7}	—	—	76.7	49.0	76.7	51.0	**79.0***	**50.8***	76.7	**50.6**
	2×10^{-7}	—	—	**77.6**	**51.8**	77.0	**51.2***	77.3	49.0	76.8	50.4
	7×10^{-8}	—	—	76.7	49.0	**77.4**	48.6	77.2	49.6	**77.2**	48.8
SimPO	1×10^{-6}	10	0.3	**80.7***	49.4	75.7	48.6	77.9	49.6	82.8	48.8
	8×10^{-7}	10	0.3	79.5	49.8	77.3	46.4	77.6	48.8	83.2	49.0
	5×10^{-7}	10	0.3	78.0	49.6	77.4	**50.4**	77.9	49.2	**83.2***	**52.0**
	8×10^{-7}	10	0.5	78.5	50.2	**77.6***	49.8	77.6	47.2	82.9	50.6
	1×10^{-6}	2.5	0.55	77.7	**50.8**	76.7	50.2	77.9	49.6	82.5	50.6
	8×10^{-7}	2.5	0.55	78.2	49.4	77.5	49.4	**78.0**	**50.6**	82.5	47.0

Table 2: Unified hyperparameter sweep across fine–tuning methods and data–generation strategies. For every hyperparameter setting we report the best–epoch accuracy (%) on **GSM8K** and **MATH**. The highest score for each fine-tuning method's data-generation strategy is **bold**. Overall best result for each data-generation strategy is *.

a 4.2% improvement over the base model. The optimal beta value was consistently 0.01 across strategies, suggesting that a mild KL constraint is preferable for mathematical reasoning tasks.

SimPO: Recent work has shown that when using online data with a reward model for preference data creation, increasing beta to 10 can substantially improve performance with the right learning rate (Meng et al., 2024). Our results strongly support this finding, as the best hyperparameters for baseline, DTS, and CoT all featured higher beta values (10) combined with carefully tuned learning rates. We also examined $\beta = 2.5$ and $\gamma = 0.55$, which were found promising in the original SimPO work for Llama 3 Instruct. Interestingly, while this configuration performed well, it was consistently outperformed by the higher beta configurations. The most striking result was achieved with DTS+SimPO at $\eta = 5e-7$, $\beta = 10$, and $\gamma = 0.3$, which produced the highest overall performance on GSM8K (83.2%) while also maintaining strong MATH performance (52.0%).

5.3 Training Dynamics

Figure 3 illustrates performance evolution across training epochs for each data generation strategy. For GSM8K, we observe distinct patterns: DTS shows strong and consistent improvement, rising sharply after epoch 1 (78.2% to 81.4%) and maintaining growth through epoch 4 (83.2%). In stark contrast, CoT performance degrades heavily after epoch 1, dropping from 77.6% to 72.6% by epoch 3, indicating significant instability. Baseline and MCTS follow similar trajectories with steady improvements until epoch 4 followed by slight regression.

For MATH, all strategies exhibit substantial variability. Baseline and MCTS display a "dip-and-recover" pattern, with performance decreasing in the middle epochs before climbing to their peaks at epoch 5 (52.2% and 50.8% respectively). DTS shows similar volatility, achieving its highest performance at epoch 2 (52.4%) before dipping and partially recovering. CoT exhibits the opposite behavior, with performance increasing until epoch 4 (51.2%) before declining sharply at epoch 5 (47.6%).

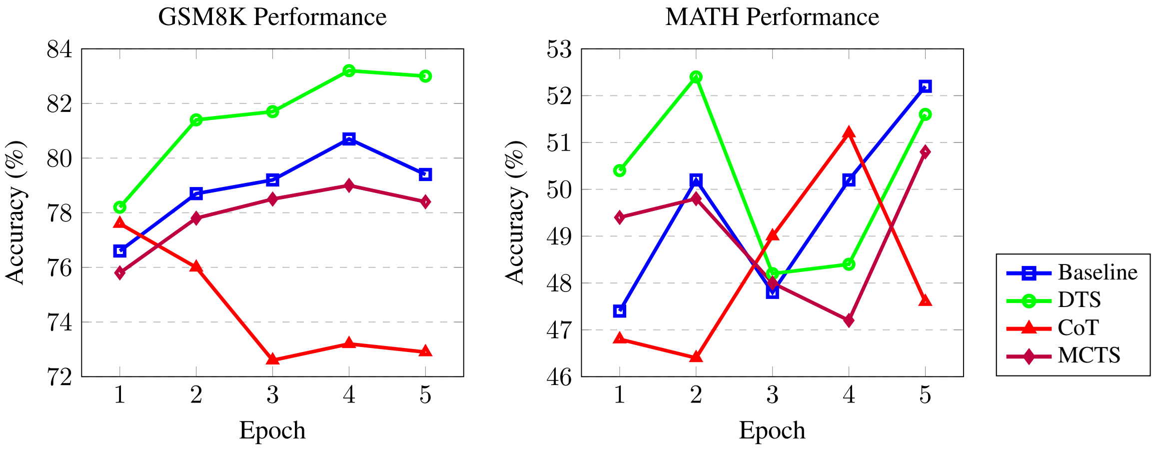

Figure 3: Performance progression across training epochs for different data generation strategies using optimal hyperparameters.

These dynamics highlight that DTS offers the most stable improvements for GSM8K, while all strategies demonstrate significant epoch-to-epoch variability on the more challenging MATH benchmark.

5.4 Computational Efficiency Considerations

Each data generation pipeline incurs a distinct token budget that corresponds to GPU hours and cost. We look at the cost of a strategy given the expected number of generated tokens per problem. The computation for the relative compute for each data generation strategy can be found in Appendix A.5.

Despite MCTS requiring significantly more computational resources, it does not yield proportional performance improvements, failing to match either DTS or even the baseline approach with SimPO. The DTS strategy offers an exceptional balance between performance and computational efficiency with only a 1.03x compute overhead compared to baseline, making it highly suitable for resource-constrained scenarios. Even with minimal additional computation, DTS achieves the best performance on both GSM8K (83.2%) and MATH (52.4%).

CoT occupies a middle ground at 1.99x baseline compute, but its unstable training dynamics and inferior performance make it less attractive despite its moderate computational requirements. The baseline approach, while computationally efficient, cannot match DTS's performance despite extensive hyperparameter optimization.

Strategy	**Relative Compute**	**GSM8K**	**MATH**
Baseline	1.00x	80.7%	52.2%
DTS	1.03x	83.2%	52.4%
CoT	1.99x	77.6%	51.2%
MCTS	4.85x	79.0%	50.8%

Table 3: Computational requirements and best performance for different data generation strategies combined with their respective optimal fine-tuning method.

6 Conclusion

Our findings demonstrate that strategic diversification of preference data can substantially enhance mathematical reasoning capabilities in LLMs. Several key insights emerge from our experiments:

Diversity of reasoning paths is crucial: Strategies that explore multiple problem-solving approaches consistently outperformed the baseline, indicating that exposure to diverse reasoning paths develops more robust mathematical capabilities.

Data quality trumps optimization algorithm: While SimPO and DPO performed best, the differences between optimization methods were smaller than those between data generation strategies, suggesting that research should prioritize data quality and diversity over algorithm selection.

Structured exploration outperforms random sampling: DTS's superior performance highlights that systematic exploration of the solution space is more effective than random variations through temperature sampling for generating high-quality preference data.

7 Limitations

7.1 Benchmark Scope and Generalizability

Our study demonstrates improvements on GSM8K and MATH benchmarks, which, while representative, capture only a subset of mathematical reasoning tasks. The effectiveness of our strategies may vary across different mathematical domains, complexity levels, or applications. Future work should evaluate these methods on a broader range of mathematical reasoning tasks and real-world applications.

7.2 Reward Model Dependencies

Despite our careful selection process (achieving error rates below 3%), our reliance on automated reward models introduces potential biases in preference data generation. These models occasionally make incorrect judgments, which could impact the quality of preference pairs and subsequent model training. Developing more robust mathematical evaluation methods remains an important avenue for future research.

7.3 Model Scale Considerations

Our experiments focused on a single model size (8B parameters). The relative effectiveness of different data diversification strategies might vary with model scale, potentially yielding different patterns of improvement in larger or smaller architectures. Extending this analysis to diverse model scales would provide valuable insights into the scalability of our approaches.

7.4 Computational Efficiency Tradeoffs

The computational requirements of more sophisticated strategies, particularly MCTS (4.85× baseline compute), limit their practical applicability in resource-constrained environments. While our DTS approach achieves an excellent balance between performance and efficiency (1.03× baseline compute), further work on optimizing data generation pipelines could improve accessibility.

8 Ethical Considerations

Our research aims to improve mathematical reasoning capabilities in language models, which has broadly positive applications in education, scientific research, and various technical domains.

We have made deliberate efforts to ensure research accessibility by providing comprehensive methodology details and implementation guidance. This openness helps democratize advanced mathematical capabilities across the research community and prevents the concentration of such capabilities in well-resourced organizations.

While enhanced mathematical reasoning could potentially enable more sophisticated applications in sensitive domains like finance or cryptography, we believe the educational and scientific benefits significantly outweigh potential risks. Mathematical reasoning fundamentally supports objective problem-solving rather than inherently harmful capabilities.

Our data generation methods rely on existing language models, which may contain biases. However, we focused specifically on mathematical problem-solving, which operates in a relatively objective domain with well-defined evaluation criteria, reducing (though not eliminating) the risk of perpetuating harmful biases.

We view our research as augmenting rather than replacing human mathematical reasoning, with the goal of creating more useful tools that complement human capabilities in educational and scientific contexts.

References

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, John Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. 2022a. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, John Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jamie Mueller, Jared Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nova Schiefer, Nelson Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Thomas Lanham, Thomas Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom

Brown, and Jared Kaplan. 2022b. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Ralph Allan Bradley and Milton E. Terry. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Howard Chen, Zexuan Rong, Kwan Li, Percy Liang, and Omar Khattab. 2024a. Assisting in writing wikipedia-like articles from scratch with large language models. *arXiv preprint arXiv:2402.14207*.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024b. Alpagasus: Training a better alpaca with fewer data. *arXiv preprint arXiv:2307.08701*.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30.

codelion. 2024. Optillm: A framework for optimizing llm generations. `https://github.com/codelion/optillm`.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.

Shunyu Feng, Lisa Yuan, Aditya Sharma, Xiang Li, Antonio Torralba, Leslie Kaelbling, Joshua Tenenbaum, and Lerrel Pinto. 2023. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*.

Luca Gallo and Sandiway Karmakar. 2024. A comparative study of dspy teleprompter algorithms for aligning large language models evaluation metrics to human evaluation. *arXiv preprint arXiv:2405.10345*.

Saujas Guo, Karan Gupta, Sanjeev Arora, Angela Fan, and Danqi Chen. 2024a. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.00330*.

Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, Johan Ferret, and Mathieu Blondel. 2024b. Direct language model alignment from online ai feedback.

Suchin Gururangan, Xiaoxuan Wang, Percy Liang, Kai-Wei Chang, and Nanyun Peng. 2024. In-context learning for extreme multi-label classification. *arXiv preprint arXiv:2401.12178*.

Alex Havrilla, Sharath Raparthy, Christoforus Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, and Roberta Railneau. 2024. Glore: When, where, and how to improve llm reasoning via global and local refinements. *arXiv preprint arXiv:2402.10963*.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Orpo: Monolithic preference optimization without reference model. *arXiv preprint arXiv:2403.07691*.

Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, José Miguel Hernández-Lobato, Richard E Turner, and Douglas Eck. 2017. Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. In *International Conference on Machine Learning*, pages 1645–1654. PMLR.

Natasha Jaques, Judy Hanwen Shen, Asma Ghandeharioun, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Shane Gu, and Rosalind Picard. 2020. Human-centric dialog training via offline reinforcement learning. *arXiv preprint arXiv:2010.05848*.

Omar Khattab, Amereen Alagesan, Ziqiao Zhuang, Cameron Harris, Percy Liang, and Matei Zaharia. 2024. Fine-tuning and prompt optimization: Two great steps that work better together. *arXiv preprint arXiv:2407.00183*.

Omar Khattab, Christopher Potts, Percy Liang, and Matei Zaharia. 2023a. Dspy assertions: Computational constraints for self-refining language model pipelines. *arXiv preprint arXiv:2312.13382*.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2023b. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2412.15298*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35:22199–22213.

Nathan Lambert, Valentina Pyatkin, Jacob Daniel Morrison, Lester James Validad Miranda, Bill Yuchen Lin, Khyathi Raghavi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hanna Hajishirzi. 2024. Rewardbench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. 2018. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Chen Liu, Kun Zhang, Shuai Sun, Kehan Chen, Qingxing Ye, Yingjun Wu, Chin-Yew Lin, and Ming Zhou. 2024a. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00676*.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024b. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. *arXiv preprint arXiv:2312.15685*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward.

Meta AI. 2024. Llama 3.1 8b instruct. `https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct`. Accessed: June 2024.

NVIDIA NeMo Team. 2024. Llama-3.1-nemotron-70b-reward-hf. `https://huggingface.co/nvidia/Llama-3.1-Nemotron-70B-Reward-HF`. Accessed: June 2024.

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 E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*.

Robin L. Plackett. 1975. The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Yue Shen, Junxian He, Yuhuai Wu, Tsung-Hsien Kuo, and Xiang Lisa Li. 2024. Unveiling the secret recipe: A guide for supervised fine-tuning small llms. *arXiv preprint arXiv:2406.09778*.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021.

Wenpin Tang, David D. Yao, Shi-Xiong Zhang, and Sambit Sahu. 2024. A survey on human preference learning for large language models. *arXiv preprint arXiv:2406.11191*.

Together AI. 2024. Llama 3.1 8B Instruct API. `https://www.together.ai/models/llama-3-1`. Accessed: May 19, 2025.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Shengyi Huang, Kashif Rasul, Alexander M. Rush, and Thomas Wolf. 2023. The alignment handbook. `https://github.com/huggingface/alignment-handbook`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*.

Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. 2024. Is dpo superior to ppo for llm alignment? a comprehensive study. *arXiv preprint arXiv:2404.10719*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*.

Jeffrey Zhou, Ki Ren Cheong, Alison Wang, and Karthik Narasimhan. 2024. Prompt-based monte carlo tree search for mitigating hallucinations in large language models. *arXiv preprint arXiv:2403.11315*.

A Additional Experimental Details

A.1 Implementation Details

We implemented all models and training procedures using the HuggingFace Transformers library (version 4.43.1). For preference optimization, we used the DPO and ORPO implementations from the TRL library (version 0.9.6), which provide optimized implementations of these algorithms. All training procedures were conducted on a compute cluster with 8 NVIDIA A100 80GB GPUs using mixed-precision training (bfloat16) to accelerate training while maintaining numerical stability for mathematical operations.

For baseline model inference and data generation, we accessed the Llama-3.1-8B-Instruct model through the Together AI API (Together AI, 2024) with consistent generation parameters across experiments (unless otherwise specified). All model evaluations on the test sets were performed with greedy decoding (`temperature` = 0) to ensure deterministic outputs and fair comparisons across methods. For the different data generation strategies, we used OptiLLM (version 0.1.8) for MCTS and CoT implementations, and developed our custom DTS pipeline using core components from the DSPy framework (version 2.6.16). For reproducibility, we set random seeds consistently (42) across all experiments.

A.2 Reward Model Analysis

Selecting an appropriate reward model is crucial for effective preference data creation, as it directly affects the quality of paired examples used during optimization. An ideal reward model should consistently assign higher scores to correct mathematical solutions than to incorrect ones, ensuring that the preference signal aligns with mathematical accuracy.

We conducted a comprehensive evaluation of several reward models using the GSM8K test set. For each problem, we generated solutions using various LLMs and compared the reward scores assigned to these solutions against those assigned to ground truth solutions. We tracked four key metrics, as defined in Section 4.2: `correct_lower` (CL), `correct_higher` (CH), `incorrect_lower` (IL), and `incorrect_higher` (IH). The most critical metric is IH, which represents cases where an incorrect solution received a higher reward than the ground truth—these cases directly undermine the preference learning objective.

As shown in Table 4, *Llama-3.1-Nemotron-70B-Reward-HF* demonstrated the highest reliability, achieving the lowest error rate of 3.11% when evaluating Llama-3.2-3B-IT outputs. The *URM-LLama-3.1-8B* model also performed well with error rates below 3.5% for the Llama-3 series, though it struggled more with Mistral-7B outputs. In contrast, the original *Skywork-Reward-Gemma-2-27B* model showed the highest error rates (>12%), frequently assigning higher rewards to incorrect solutions, though its v0.2 iteration showed substantial improvement.

To further validate our reward model selection, we extended our evaluation to the more challenging MetaMathQA dataset, sampling 4,919 problems. The Llama-3.1-Nemotron-70B-Reward-HF model maintained consistent performance with a 2.76% error rate (136 IH cases out of 4,919 total evaluations), confirming its robustness across different mathematical problem distributions. Based on these results, we selected *Llama-3.1-Nemotron-70B-Reward-HF* as our reward model for all preference data generation in our experiments.

A.3 Judgment and Completion Scoring Setup

Accurate assessment of mathematical solutions requires a robust scoring mechanism that can evaluate both the correctness of final answers and the quality of intermediate reasoning steps. To achieve this, we implemented a structured judgment framework using Nvidia's *Llama-3.1-Nemotron-70B-Instruct-HF* model as our scoring engine.

A.3.1 Scoring Protocol

We normalized scores on a 0-100 scale, where 100 represents completely correct solutions with high-quality reasoning, and 0 indicates entirely incorrect solutions with flawed reasoning paths. To ensure consistent and meaningful evaluations, we designed a comprehensive system prompt that instructs the judge model to:

1. Evaluate correctness relative to reference solutions
2. Award partial credit for correct reasoning steps (up to 60 points)
3. Reserve scores of 80+ for completely correct solutions
4. Provide detailed explanations for point deductions

The full judgment prompt is structured as follows:

Reward Model	Generator Model	CL	CH	IL	IH	Error (%)
URM-LLama-3.1-8B	Mistral-7B-IT-v0.1	345	222	657	95	7.20%
	Gemma-2-9B-IT	313	853	73	80	6.07%
	Llama-3.2-3B-IT	691	349	235	44	3.34%
	Llama-3.1-8B-IT	735	364	175	45	3.41%
Llama-3.1-Nemotron-70B-Reward-HF	Gemma-2-9B-IT	479	455	322	63	4.78%
	Llama-3.2-3B-IT	398	647	233	41	**3.11%**
	Llama-3.1-8B-IT	406	697	173	43	3.26%
Skywork-Reward-Gemma-2-27B	Llama-3.2-3B-IT	28	1012	119	160	12.13%
	Llama-3.1-8B-IT	32	1048	78	161	12.21%
Skywork-Reward-Gemma-2-27B-v0.2	Llama-3.1-8B-IT	560	545	146	68	5.16%

Table 4: Reward Model Evaluation on the GSM8K Test Set. We evaluate various reward models against different generator models, tracking: **CL** (Correct Lower)—model's correct output received lower reward than ground truth; **CH** (Correct Higher)—model's correct output received higher reward than ground truth; **IL** (Incorrect Lower)—model's incorrect output received lower reward than ground truth; **IH** (Incorrect Higher)—model's incorrect output received higher reward than ground truth. The **Error** rate shows the percentage of incorrect outputs receiving higher rewards than ground truth, calculated as IH/(CL+CH+IL+IH).

> Here is a math question: {question}
> Here is the gold answer: {gold_answer}
> Here is a student answer: {generated_answer}
> You are a math teacher grading a student's answer. You need to judge if the student answer is correct based on the gold answer. You need to follow the following rubrics: 1. The score should be between 0 and 100. 2. If the student answer is not correct based on the gold answer, deduct points from the score for each wrong step. Add points to the score for each correct step, up to a maximum of 60 points. 3. If the student answer is correct based on the gold answer, please give a final score above 80. 4. Please give a detailed explanation in bullet points for each point deducted. In the end, the score and explanation should be in the following format. Note that the final output should be parsed as a json object.
> <explanation>
> {"correct": true/false, "score": integer}

A.3.2 Implementation Details

To ensure scoring consistency and determinism, we set the generation parameters to `temperature=0.0` and `max_tokens=4,096`. The structured JSON output format ({`correct`, `score`}) facilitated automated extraction and processing using regular expressions. In rare cases where the judge model produced malformed outputs or failed to follow the required format, we assigned a score of -1 and excluded these samples from subsequent analysis to maintain data quality.

The MetaMathQA dataset provided high-quality reference solutions that served as our gold standard for comparison. As noted in our reward model analysis (Table 4), we observed occasional cases where reference solutions received lower scores than incorrect model-generated solutions.

In our preliminary analysis, we found that this judgment approach provided more nuanced and informative scores compared to simple binary correctness checks, enabling finer distinctions between solutions with similar final answers but different reasoning quality. The detailed explanations produced by the judge model also provided valuable insights for qualitative analysis of model performance patterns and failure modes.

A.4 Preference Optimization Configuration

Effective preference optimization requires careful configuration of training parameters to balance learning dynamics, computational efficiency, and model stability. We implemented a consistent training infrastructure across all fine-tuning methods, varying only the specific hyperparameters detailed in our ablation study (Section 5.2).

Our training infrastructure leveraged DeepSpeed

```
ACCELERATE_LOG_LEVEL=info accelerate
    launch \
    --config_file deepspeed_zero3.yaml \
    --dataset_name dpo_dataset \
    --model_name_or_path meta-llama/
        Llama-3.1-8B-Instruct \
    --learning_rate 3.0e-7 \
    --beta 0.01 \
    --lr_scheduler_type cosine \
    --bf16 true \
    --num_train_epochs 5 \
    --per_device_train_batch_size 1 \
    --gradient_accumulation_steps 16 \
    --gradient_checkpointing \
    --gradient_checkpointing_kwargs '{"
        use_reentrant": false}' \
    --logging_steps 25 \
    --eval_strategy 'no' \
    --optim adamw_torch \
    --attn_implementation
        flash_attention_2 \
    --save_strategy epoch \
    --seed 42 \
    --warmup_ratio 0.1 \
    --no_remove_unused_columns
```

Figure 4: Representative DPO training configuration used for fine-tuning. We maintained this base configuration across all preference optimization methods, adjusting only the method-specific hyperparameters (e.g., learning rate, beta, gamma) according to our ablation studies.

ZeRO-3 for memory optimization (Rasley et al., 2020), FlashAttention-2 for efficient attention computation (Dao, 2024), and mixed-precision training (bfloat16) to accelerate training while maintaining numerical stability. We employed gradient checkpointing to reduce memory requirements, enabling us to process longer mathematical reasoning sequences without compromising batch size.

For all preference optimization methods (DPO, ORPO, SimPO), we maintained a global batch size of 16, configured as per GPU batch size of 1 with 16 gradient accumulation steps. This batch size was selected based on prior work (Meng et al., 2024) suggesting that moderate batch sizes (16-32) achieve optimal performance for preference learning across diverse domains. Each training run was executed for 5 epochs with a cosine learning rate schedule and 10% warmup ratio to ensure stable optimization dynamics.

Figure 4 shows a representative configuration for DPO training. When implementing other methods, we maintained this base configuration while adjusting method-specific parameters:

- **DPO**: Varying learning rates (1e-7 to 7e-7) and beta values (0.01, 0.05, 0.1)
- **ORPO**: Varying learning rates (7e-8 to 5e-7) with lambda fixed at 1.0
- **SimPO**: Varying learning rates (5e-7 to 1e-6), beta values (2.5, 10), and gamma values (0.3, 0.5, 0.55)

For each method-strategy combination, we conducted a grid search over these hyperparameters as detailed in Section 5.2, totaling 76 distinct training runs. This comprehensive approach enabled us to identify optimal configurations for each data generation strategy, revealing important patterns in how hyperparameter sensitivity varies with data characteristics.

A.5 Token Count Estimation for Computational Efficiency Analysis

To quantify the computational resources required by each data generation strategy, we developed a systematic approach for estimating relative compute costs based on token processing requirements. We use a normalized compute ratio expressed as:

$$\text{Relative Compute} = \frac{t_p + t_o}{t_p^{\text{base}} + t_o^{\text{base}}}$$

Where t_p represents the mean prompt token count for the strategy being measured, t_o is the mean output token count, and the denominator contains the corresponding values for our baseline strategy. This metric captures the computational overhead of each strategy relative to the simplest approach.

A.5.1 Strategy-Specific Token Analysis

Baseline Strategy: For our reference implementation, we measured an average problem length of 41 tokens, a system prompt of 77 tokens, and a mean generation length of 364 tokens, resulting in a total token count of $41 + 77 + 364 = 482$ tokens per problem.

Chain-of-Thought: Implemented using OptiLLM's Chain-of-Thought framework[3], this approach uses a structured thinking template of 258 tokens combined with the problem (41 tokens), totaling 299 input tokens. We observed significantly longer generations averaging 661 tokens (due to the explicit reasoning steps and occasional verbose output), bringing the total to 960 tokens

---

[3] `https://github.com/codelion/optillm/blob/main/optillm/cot_reflection.py`

per problem. This corresponds to a compute ratio of $960/482 = 1.99\times$.

Monte Carlo Tree Search: Our MCTS implementation uses a simulation depth of 1 and performs 2 simulations. Based on the OptiLLM MCTS implementation[4], each problem requires a total of 8 model calls: an initial expansion, plus 4 LLM calls per simulation (generate_actions(), apply_action(), and evaluation_state()) (Zhou et al., 2024; Feng et al., 2023). With an average output of 292 tokens per model call, this strategy consumes approximately $292 \times 8 = 2,336$ tokens, yielding a compute ratio of $2,336/482 = 4.85\times$.

Diversified-ThinkSolve (DTS): This strategy requires two sequential LLM calls per problem:

- *Thought Generation:* System prompt (56 tokens) + problem (41 tokens) = 97 input tokens, producing an average of 50 output tokens
- *Solution Generation:* System prompt (27 tokens) + problem + thought (91 tokens) = 118 input tokens, generating an average of 230 output tokens

The total token count for DTS is $(97 + 118) + (50 + 230) = 495$ tokens, resulting in a compute ratio of $495/482 = 1.03\times$ relative to baseline.

A.5.2 Efficiency Analysis

This token-based analysis reveals significant differences in computational requirements across strategies. While DTS achieves substantially better performance than baseline (as shown in Section 5.1), it does so with only a 3% increase in computational cost. In contrast, MCTS requires nearly 5 times more compute while delivering less impressive results. These efficiency metrics provide crucial context for evaluating the practical utility of each strategy, especially in resource-constrained scenarios where computational efficiency is a key consideration alongside raw performance.

B Additional Results

B.1 GSM8K 5-shot Performance Analysis

While our main evaluation focused on zero-shot performance, few-shot evaluation provides valuable insights into how preference optimization affects model performance when provided with exemplars. Table 5 presents the GSM8K 5-shot accuracy results across all strategies and fine-tuning methods.

<table>
<tr><td colspan="5">Base Model (No fine-tuning): 83.9%</td></tr>
<tr><th>Strategy</th><th>SFT</th><th>DPO</th><th>ORPO</th><th>SimPO</th></tr>
<tr><td>Baseline</td><td>84.2%</td><td><u>86.0%</u></td><td>85.1%</td><td>85.9%</td></tr>
<tr><td>CoT</td><td>84.1%</td><td>85.1%</td><td>85.5%</td><td>85.3%</td></tr>
<tr><td>MCTS</td><td>84.8%</td><td>84.9%</td><td>85.3%</td><td>85.1%</td></tr>
<tr><td>DTS</td><td>84.8%</td><td>85.8%</td><td>85.8%</td><td>85.0%</td></tr>
</table>

Table 5: GSM8K 5-shot accuracy (%) across data generation strategies and optimization methods using optimal hyperparameter configurations. The highest score for each fine-tuning method is **bold** with the best overall result **underlined**. All models show improvement over the base model's 83.9% accuracy.

B.1.1 Key Findings

All fine-tuned models demonstrated improvements over the base model's already strong 5-shot performance (83.9%), with gains ranging from modest (+0.2%) to substantial (+2.1%). Several notable patterns emerged from our analysis:

- **Strategy-Method Interactions:** Unlike the 0-shot scenario where DTS consistently outperformed other strategies, the baseline strategy achieved the highest overall 5-shot accuracy (86.0%) when combined with DPO. This suggests that the benefits of diverse reasoning paths may be partially redundant with the information provided by exemplars.
- **Method-Specific Performance:** DTS showed the most consistent performance across different fine-tuning methods, scoring strongly with SFT (84.8%), DPO (85.8%), and ORPO (85.8%). However, it unexpectedly underperformed with SimPO (85.0%) relative to other strategies, despite SimPO being the optimal method in 0-shot evaluations.
- **Hyperparameter Consistency:** We observed interesting patterns in optimal hyperparameters for 5-shot performance. For both DPO and ORPO, a learning rate of 5e-7 consistently yielded the best results across all data generation strategies, with DPO also favoring $\beta = 0.01$. For SimPO, we found strategy-dependent optimal configurations: baseline, DTS, and CoT performed best with $\beta = 10$,

---

[4] `https://github.com/codelion/optillm/blob/main/optillm/mcts.py`

$\gamma = 0.3$, and learning rates of 5e-7 or 8e-7, while MCTS uniquely benefited from $\beta = 2.5$, $\gamma = 0.55$, and a learning rate of 8e-7.

B.1.2 Implications

The differences between 0-shot and 5-shot performance patterns suggest that preference optimization may operate differently when exemplars are available. While diverse reasoning paths (as in DTS) appear critical for strong 0-shot performance, more conventional approaches like our baseline strategy may be sufficient when combined with few-shot prompting.

B.2 Epoch-wise Analysis of MATH Benchmark Performance

Understanding how mathematical reasoning capabilities evolve during training provides valuable insights into the learning dynamics of different preference optimization approaches. Table 6 presents a comprehensive view of MATH benchmark performance across all five training epochs for each strategy-method combination.

Our epoch-by-epoch analysis reveals distinct training patterns across different approaches:

Baseline Strategy: While SFT performance gradually declined with additional epochs, preference optimization methods showed non-monotonic improvement patterns. Most notably, DPO achieved its peak performance (52.2%) at the final epoch, demonstrating continued learning throughout training. Both ORPO and SimPO reached their peak performance in earlier epochs (epochs 3 and 2, respectively) before beginning to decline, suggesting potential overfitting.

Chain-of-Thought (CoT): Interestingly, CoT methods consistently reached their peak performance at later epochs (typically epoch 4) compared to other strategies. This delayed optimization might suggest that extracting useful signals from CoT-generated preferences requires more training time, perhaps due to the additional reasoning steps that must be learned.

Monte Carlo Tree Search (MCTS): MCTS exhibited the most unstable training dynamics, particularly when combined with DPO and SimPO. While MCTS+SimPO started strongly (50.6% at epoch 1), it catastrophically collapsed to single-digit performance by epoch 3. Similarly, MCTS+DPO declined from 49.2% to below 40% in later epochs. This instability suggests that preferences generated through MCTS may contain conflicting or inconsistent signals that become increasingly problematic with continued training.

Diversified-ThinkSolve (DTS): The DTS strategy demonstrated remarkable stability across training epochs, with all methods maintaining strong performance throughout. The combination of DTS with DPO achieved the highest overall MATH accuracy (52.4%) at epoch 2, followed by a temporary decline and subsequent recovery in later epochs. SimPO exhibited a similar pattern with its peak (52.0%) at epoch 4. This oscillatory behavior might indicate that models trained on diverse reasoning paths explore different regions of the solution space during training.

B.3 ORPO Learning Rate Sensitivity Analysis

Learning Rate	GSM8K 0-shot	MATH
8e-6	45.6%	46.4%
2e-6	68.4%	48.2%
7e-7	**76.9%**	**46.8%**

Table 7: Impact of learning rate on ORPO performance using the baseline data generation strategy.

While most preference optimization methods are known to be sensitive to learning rate selection, ORPO deserves special attention due to the significantly higher learning rates recommended in the original paper (8e-6) compared to our optimal findings. We conducted targeted experiments to quantify this sensitivity and determine appropriate learning rate ranges for mathematical reasoning tasks.

As shown in Table 7, ORPO's performance exhibits extreme sensitivity to learning rate selection when applied to mathematical reasoning tasks. Using the originally recommended learning rate of 8e-6 results in catastrophically poor performance on GSM8K (45.6%), significantly worse than even the untuned base model (76.1%). Reducing the learning rate by approximately an order of magnitude (to 7e-7) restores and slightly enhances performance (76.9%).

This stark difference can be attributed to the unique characteristics of mathematical reasoning tasks compared to general instruction-following or conversational benchmarks. Mathematical reasoning typically requires precise manipulation of symbols and strict adherence to formal rules, which may be disrupted by aggressive parameter updates.

Strategy	Method	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
Baseline	SFT	**48.4%**	47.8%	47.6%	47.4%	47.2%
	DPO	47.4%	50.2%	47.8%	50.2%	**52.2%**
	ORPO	48.8%	49.4%	**51.8%**	47.6%	48.0%
	SimPO	46.6%	**50.8%**	48.4%	47.4%	47.0%
CoT	SFT	48.4%	48.0%	48.0%	**50.6%**	50.0%
	DPO	47.2%	47.2%	48.4%	**50.8%**	47.0%
	ORPO	46.8%	46.4%	49.0%	**51.2%**	47.6%
	SimPO	47.0%	**50.4%**	47.6%	47.2%	48.2%
MCTS	SFT	47.8%	**48.8%**	47.4%	47.6%	46.6%
	DPO	48.2%	**49.2%**	39.2%	39.8%	37.8%
	ORPO	49.4%	49.8%	48.0%	47.2%	**50.8%**
	SimPO	**50.6%**	42.8%	9.8%	9.0%	9.0%
DTS	SFT	48.0%	48.6%	**49.6%**	48.8%	49.2%
	DPO	50.4%	**52.4%**	48.2%	48.4%	51.6%
	ORPO	48.0%	**50.6%**	49.8%	49.4%	50.6%
	SimPO	48.8%	51.2%	47.8%	**52.0%**	48.0%

Table 6: MATH benchmark accuracy (%) progression across training epochs for all data generation strategies and optimization methods. Results shown represent the best hyperparameter configuration for each strategy-method pair. The highest score for each combination is highlighted in **bold**, while the overall best result is **underlined**.

The optimal learning rate we identified (7e-7) is approximately 11 times smaller than the originally recommended value, highlighting the importance of domain-specific hyperparameter tuning.

Interestingly, while GSM8K performance was dramatically affected by learning rate selection, MATH benchmark results showed more modest variation (46.4% to 48.2%). This differential sensitivity might be explained by the higher complexity and diversity of problems in the MATH dataset, which may require more substantial model adaptation regardless of learning rate.

These findings emphasize the critical importance of carefully tuning learning rates when applying preference optimization techniques to specialized domains like mathematical reasoning, rather than relying on default values established for general-purpose language tasks.

C Analysis of Generated Solutions

To provide qualitative insights into how different data generation strategies influence mathematical reasoning patterns, we examined solution characteristics across models trained with each approach. This section presents a comparative analysis of solutions to an illustrative problem from the MetaMathQA dataset.

C.1 Example Solutions

Problem: Diego baked 12 cakes for his sister's birthday. Donald also baked 4 cakes, but ate x while waiting for the party to start. There are 15 cakes left. What is the value of unknown variable x?

Baseline Solution:

To find the value of the unknown variable x, let's follow these steps:

Step 1: Find the total number of cakes initially made by both Diego and Donald.

We are given that Diego baked 12 cakes, and Donald baked 4 cakes.

So, total_cakes $=$ cakes_baked_by_Diego $+$ cakes_baked_by_Donald $= 12+4 = 16$

Step 2: Understand the final number of cakes left.

We know that after Donald ate x cakes, 15 cakes are left.

So, the initial total number of cakes minus the number eaten is equal to 15:

total_cakes $- x = 15$

Since total_cakes $= 16$ from Step 1, we substitute 16 for total_cakes:

$16 - x = 15$

Step 3: Find the value of the unknown variable x.

Now we will isolate x by getting x on one side of the equation:

$16 - x = 15$

We need to solve for x: $x = 16 - 15$

Step 4: Perform the operation to isolate x.

Using simple arithmetic: $x = 1$

Final Answer: 1

Chain-of-Thought (CoT) Solution:

Thinking:

Let's denote the total number of cakes initially as T.

Donald initially had 4 cakes and ate x cakes. Therefore, Donald had $4-x$ cakes left.

Diego initially had 12 cakes, so after Donald ate some, the total number of cakes left is the sum of the remaining cakes from both Diego and Donald. The equation can be written as: $12 + (4 - x) = 15$.

First, let's simplify the equation by combining the constants: $12 + 4 - x = 15$.

Combine the constants: $16 - x = 15$.

Now, let's isolate the variable x by subtracting 16 from both sides of the equation. $-x = -1$.

To isolate x, multiply both sides of the equation by -1. $x = 1$.

Reflection:

I have simply applied basic algebraic manipulations to the given equation. There are no obvious errors in this process, and I have checked the steps for consistency and accuracy. However, it's essential to ensure that all variables are correctly isolated and that the algebraic manipulations are accurately performed.

Output:

$x = 1$.

Monte Carlo Tree Search (MCTS) Solution:

To find the value of x, we need to determine the total number of cakes initially. Diego baked 12 cakes and Donald baked 4 cakes. However, since the information about how many cakes Donald initially had baked was not required for solving the problem you can just sum up Diego and Donald's initial cakes to determine the total.

Initial total cakes = Diego's cakes + Donald's cakes = $12 + 4 = 16$ cakes.

After waiting for the party to start, there are 15 cakes left.

We subtract the number of remaining cakes from the total number of cakes to find x. This equation would be:

Initial total cakes $-$ remaining cakes $=$ x.

$16 - 15 = x$

$x = 1$.

The value of the unknown variable x is 1. Donald ate 1 cake while waiting for the party to start.

Diversified-ThinkSolve (DTS) Solution:

Thought Output: We can use the concept of inverse operations to isolate x. Subtracting 12 from both sides of the equation gives us: $-x = -5$.

Solution Output: To find the value of x, we need to isolate x by using inverse operations. The correct equation is $16 - x = 15$. Subtracting 16 from both sides of the equation gives us $-x = -1$. Multiplying both sides of the equation by -1 gives us $x = 1$.

C.2 Qualitative Analysis of Solution Characteristics

Our examination of solutions generated across different data strategies reveals distinctive patterns in reasoning approaches, solution structures, and error patterns.

Baseline Solutions typically exhibit a highly structured, step-by-step approach with explicit enumeration of each reasoning stage. The solution organization appears optimized for instructional clarity, with distinct sections and a formal problem-solving framework. While effective, this approach sometimes leads to unnecessarily verbose explanations even for straightforward problems.

CoT Solutions feature extensive explanatory content with distinct thinking and reflection phases. The thinking phase often includes variable definitions and elaborate equation formulations, while the reflection phase provides meta-analysis of the solution approach. This structure appears to prompt deeper verification and error-checking, but sometimes at the cost of parsimony. The explicit verification step may contribute to CoT's inconsistent performance observed in our quantitative results.

MCTS Solutions exhibit a remarkably consistent structure across problems, typically beginning with a standardized phrase ("To find the value of x, we need to determine...") that suggests convergence toward optimal response templates through the search process. The solutions tend to be direct and focused on the most efficient path discovered during tree search. However, this approach occasionally leads to incorrect convergence on harder problems when the search depth is insufficient to fully explore the solution space.

DTS Solutions demonstrate a unique two-phase structure reflecting the strategy's decomposition approach. The initial "thought output" often contains a high-level strategy or alternative solution approach, while the subsequent "solution output" provides a direct, efficient solution path. This dual structure appears to enable a balance between conciseness and reasoning depth. The example solution illustrates how DTS can derive a more direct mathematical approach (using inverse operations) compared to other methods.

D Diversified-ThinkSolve (DTS) Implementation Details

DTS was implemented using DSPy framework components, with a modular design that separates thought generation from solution execution. Our implementation consists of two primary modules which can be found in Figure 5 and Figure 6. The DTS implementation incorporates several key design elements:

Modularity: By separating thought generation from solution execution, each component can be independently optimized.

Robust Error Handling: Comprehensive fallback mechanisms prevent pipeline failures during batch processing.

Structured Output Processing: Regex-based parsing extracts individual thoughts from varied model outputs, ensuring consistent downstream processing.

Guaranteed Diversity: The system enforces a minimum of five distinct approaches per problem, even when the base model tends toward homogeneity.

The complete pipeline processes each problem through ThoughtGenerator, passes each generated approach to SolutionGenerator, collects the resulting solutions, scores them with the reward model, and creates preference pairs for optimization. This architecture maintains computational efficiency (1.03× baseline) while producing the diverse, high-quality preference data that enabled DTS's superior performance.

```
class ThoughtGenerator(dspy.Module):
    def __init__(self):
        super().__init__()
        self.gen_thoughts = dspy.ChainOfThought("math_problem -> thoughts: List[str]")

    def forward(self, math_problem: str) -> List[str]:
        try:
            prompt_template = (
                "Given the math problem: {problem}, provide 5 possible approaches or "
                "initial thoughts on how to solve it, including any relevant mathematical "
                "concepts, formulas, or techniques that may be applied. Consider multiple "
                "perspectives and potential solution paths, and describe each thought in 1-2 sentences."
            )

            result = self.gen_thoughts(math_problem=prompt_template.format(problem=math_problem))
            thoughts = result.reasoning if hasattr(result, 'reasoning') else []

            # process thoughts
            if isinstance(thoughts, str):
                import re
                thoughts = re.split(r'\d+\.|\n\d+\.|\n\d+\)', thoughts)
                thoughts = [t.strip() for t in thoughts if t.strip()]

            # ensure exactly 5 thoughts
            if len(thoughts) < 5:
                while len(thoughts) < 5:
                    thoughts.append(f"Alternative approach {len(thoughts) + 1}: Apply fundamental mathematical principles to solve step by step.")

            return thoughts

        except Exception as e:
            print(f"Error in ThoughtGenerator: {str(e)}")
            return [f"Default approach {i+1}: Solve the problem systematically using basic mathematical principles."
                   for i in range(5)]
```

Figure 5: ThoughtGenerator module implementation responsible for generating diverse mathematical reasoning approaches.

```python
class SolutionGenerator(dspy.Module):
    def __init__(self):
        super().__init__()
        self.gen_solution = dspy.ChainOfThought("math_problem, approach -> solution: str")

    def forward(self, math_problem: str, approach: str) -> str:
        try:
            prompt_template = (
                "Given the math problem: {problem}\n"
                "Using this approach: {approach}\n"
                "Please provide a detailed solution showing all work and steps."
            )

            result = self.gen_solution(math_problem=math_problem, approach=approach)

            return result.reasoning if hasattr(result, 'reasoning') else "Unable to generate solution."

        except Exception as e:
            print(f"Error in SolutionGenerator: {str(e)}")
            return "Error occurred while generating solution."
```

Figure 6: SolutionGenerator module implementation that produces complete solutions based on specific reasoning approaches.