

# Improving Scientific Reasoning in Small Language Models via Process Preference Re-Ranking

Stanford CS224N Custom Project

**Marianne Liu**

Department of Computer Science  
Stanford University  
mliu7051@stanford.edu

**Arya Gupta**

Department of Computer Science  
Stanford University  
aryagupt@stanford.edu

## Abstract

Multi-step scientific reasoning remains challenging for small language models, where intermediate errors compound and degrade final answers. In this project, we explore training a process preference model (PPM) to evaluate the quality of reasoning traces generated from a base policy model. We fine-tune Qwen2.5-Math-1.5B with LoRA on the Gargantua-R1-Compact dataset as our policy model, and train a Longformer-based Process Preference Model (PPM) that reranks  $k = 5$  candidate solutions at inference time. We compare against greedy decoding and self-consistency baselines across 88 held-out problems. SFT with self-consistency achieves 78.4% accuracy, with fine-tuning providing a consistent 5-8% gain over greedy decoding, though these gains plateau as  $k$  increases. Our PPM achieves 55.7% accuracy vs. 42.0% base greedy, demonstrating that a learned ranker on reasoning alone can identify correct solutions from the candidate pool, but requires improvements in training, preference pair creation, and accuracy assessments to make greater improvements towards more reasoning understanding.

## 1 Key Information to include

- Mentor: Chenglei Si
- Late Days (if you are using any):

| Team member                      | Late days remaining |
|----------------------------------|---------------------|
| Marianne Liu                     | 5                   |
| Arya Gupta                       | 0                   |
| Total pooled                     | 5                   |
| Team size                        | 2                   |
| Late days applied for the report | 2                   |

## 2 Introduction

Scientific reasoning requires chaining domain knowledge, physical principles, and quantitative calculations across multiple steps. Errors early in a reasoning chain can propagate and lead to incorrect final answers. A single misapplied assumption early in a reasoning chain can cascade into an entirely wrong final answer. As identified by Zheng et al., one of the main challenges in using large language models (LLMs) in scientific domains is their lack of transparency and interpretability: "The challenge is not just explaining outputs, but ensuring the AI's internal logic aligns with scientific principles and can reliably differentiate asserted claims from verifiable truths" [1]. This interpretability is crucial for reliable and reproducible LLM-based scientific discovery.

The difficulty is compounded for small language models. While frontier models with hundreds of billions of parameters can often brute-force these tasks through scale and pretraining, smaller model,

far more practical to deploy, lack the capacity and training signal to do so reliably. This leads us to our research question: Can we make small models reason more reliably by changing how the method of reasoning at inference time?

One of the most popular existing approaches is supervised fine-tuning (SFT), but fine-tuned models can still produce incorrect solutions and provide no mechanism for selecting among multiple candidates. Methods such as self-consistency sample multiple solutions and select the most frequent final answer, but they ignore step-level reasoning, making them unable to distinguish incorrect solutions from correct ones.

Thus, we explore training a separate model to evaluate and rank candidate reasoning trajectories, providing a learning signal beyond simple answer aggregation. We hypothesize that evaluating reasoning trajectories may be easier than generating them. This is especially valuable in scientific domains, where plausible but incorrect reasoning is common and masks conceptual errors.

In this project, we fine-tune Qwen2.5-Math-1.5B on science-heavy reasoning problems across physics, chemistry, thermodynamics, and electricity, then train a separate Process Preference Model (PPM) to rerank  $k = 5$  candidate solutions based on their reasoning quality at inference time.

After evaluation, we verify that supervised fine-tuning improves performance over the base model. SFT with greedy decoding achieves 46.6% accuracy vs. 42.0% for the base model. Self-consistency alone proves to be highly effective, with accuracy reaching 78.4% for the SFT model vs. 72.7% for the base model.

PPM reranking reaches 55.7% accuracy, improving over SFT greedy decoding but still trailing majority voting. Although Pass@5 reaches 77.3%, indicating correct solutions often exist in the candidate pool, but the PPM frequently fails to select them. We believe this gap occurs because the training signal wasn't detailed or large enough to provide meaningful improvement. These results suggest that to increase performance, the PPM needs more detailed training feedback, for example, scoring at each reasoning step rather than just the overall solution, and harder negative examples.

### 3 Related Work

Our work lies at the intersection of three areas: process supervision for reasoning, inference-time compute strategies, and learned reranking for small language models. Prior work in each area either requires costly supervision or leaves step-level reasoning signal underutilized.

**Outcome vs. Process Supervision.** A key question in training reasoning models is whether feedback should target final answers or intermediate reasoning steps. Outcome-supervised reward models (ORMs) are simple to train but often reward shortcut reasoning, allowing correct answers despite flawed intermediate logic [2]. Lightman et al. show that process-level supervision substantially improves solution selection on the MATH dataset [3], but this approach relies on large-scale human annotation of reasoning steps. Automated labeling methods using Monte Carlo sampling or MCTS have been explored, but they struggle to produce reliable step-level reward signals. rStar-Math [4] addresses this by replacing scalar step scoring with preference-based training.

**Inference-Time Compute and Multi-Sample Strategies.** Another line of work improves reasoning by increasing inference-time compute. Wang et al. introduce self-consistency, which samples multiple reasoning trajectories and selects the most frequent final answer, yielding strong gains over greedy decoding [5]. However, it aggregates only final answers and ignores step-level reasoning signals. More advanced approaches, such as MCTS-based methods [2], explore reasoning paths more systematically but incur significant computational overhead.

**Mutual Reasoning and Trajectory Verification.** Qi et al. propose rStar, where one small model generates reasoning trajectories via MCTS while another selects trajectories on which both models agree [2]. This avoids training a reward model and improves accuracy without fine-tuning, but agreement between models does not guarantee domain correctness, particularly in scientific reasoning.

**Process Preference Models.** Guan et al. extend this idea with rStar-Math, which trains a Process Preference Model (PPM) using pairwise ranking rather than scalar rewards [4]. By learning only to prefer higher-quality trajectories, the PPM simplifies supervision and achieves strong reasoning performance with small models. However, rStar-Math relies on code execution to verify intermediate steps, which is effective for mathematics but difficult to apply to scientific reasoning. Our work tests

whether preference-based trajectory evaluation remains effective under noisier supervision using a simpler, single-round training setup.

## 4 Approach

Our pipeline consists of two independently trained components. First, a policy model that generates candidate reasoning trajectories for a series of reasoning problems specialized in scientific reasoning. Second, a Process Preference Model (PPM) that is trained on preference pairs generated from our Qwen2.5-Math-1.5B model.

We wanted two distinct representations (1) of a finetuned scientific reasoning model, a specific implementation of an implemented methodology and (2) a PPM that specifically understands the concept of reasoning. Thus, we can evaluate which of these methods contributes to the task of understanding reasoning.

Then, at inference time we conduct our contribution in methodology. The policy model produces  $k$  candidate solutions via stochastic decoding, and the PPM generates a score for all  $k$  solutions, selecting the highest-scoring one. All training, evaluation, and processing code described below was written by us.

**Policy Model.** We fine-tune Qwen2.5-Math-1.5B [6] on our science-heavy training set using LoRA [7] (rank 16,  $\alpha = 32$ , dropout 0.05, targeting q/v projections) with AdamW and gradient clipping at norm 1.0. Each training example includes a single problem followed by the solution, ending in a tagged final answer. Loss is computed only on solution tokens, so the model learns to generate solutions rather than reconstruct the problem. Fine-tuning is implemented with HuggingFace Transformers and PEFT.

**Process Preference Model (PPM).** We train a separate scoring model  $r_\theta(x, y)$  to rerank candidate trajectories produced by the policy model rather than generate solutions itself. Unlike approaches that build the reward model on top of the policy model’s weights, we use an independent encoder architecture specifically, Longformer-base-4096 [8]. This is because separating the reward model from the policy model avoids representational interference during training and allows the PPM to be updated independently without affecting generation quality.

The PPM architecture extracts the [CLS] token embedding from Longformer’s final hidden layer and passes it through a linear reward head to produce a scalar score:

$$r_\theta(x, y) = W \cdot h_{\text{CLS}}$$

where  $h_{\text{CLS}} \in \mathbb{R}^{768}$  is the first token’s hidden state from the encoder. The full Longformer backbone is fine-tuned jointly with the reward head using the Bradley–Terry preference loss:

$$\mathcal{L}(\theta) = -\log \sigma(r_\theta(x, y^+) - r_\theta(x, y^-))$$

where  $y^+$  and  $y^-$  are correct and incorrect trajectories for the same problem  $x$ , paired by final-answer correctness. Training uses AdamW with learning rate  $1 \times 10^{-5}$ , batch size 4, and gradient clipping at 1.0 for 5 epochs over approximately 440 preference pairs (90/10 train/validation split).

Preference pairs are constructed by generating  $k = 5$  trajectories per problem via stochastic decoding (temperature 0.7, top-p = 0.9), labeling each by final-answer correctness, and forming all valid positive–negative combinations. For sequences exceeding the 2048-token budget, we apply middle truncation, retaining the first one-third and last two-thirds of tokens with a separator at the join, biasing toward the end of the solution where the final answer appears. We chose 2048 because it was most similar to the median solution length while allowing reasonable compute usage. Reasoning requires long text outputs, which were infeasible for us to fully accommodate given our resource limitations.

While using the policy model’s representations might provide a stronger prior scientific reasoning, keeping the encoder separate has practical advantages. It’s easier to swap out components and can handle longer inputs. Our results suggest that the main bottleneck isn’t the encoder’s capacity, but rather the quality of the training signal used to train the PPM, which we further discuss in Section 5.

**Baselines and Evaluation Methods.** We evaluate six configurations that share identical generation procedures but differ in their answer selection strategy, allowing us to isolate the contribution of each component:

| Method                                         | Description                                                                                                                                                                                                                                                                                                   |
|------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Method 1: Base LM (Greedy)</b>              | The unmodified Qwen2.5-Math-1.5B model with greedy decoding ( $T = 0$ ), producing a single deterministic solution per problem. This establishes the floor performance of the pretrained model on scientific reasoning tasks.                                                                                 |
| <b>Method 2: Base LM + Self-Consistency</b>    | The base model generates $k = 5$ candidate solutions via stochastic sampling (temperature $T = 0.7$ , top- $p = 0.9$ ), and the final answer is selected by majority vote over extracted answers. This measures how much answer diversity and consensus improve over greedy decoding without any fine-tuning. |
| <b>Method 3: SFT Policy (Greedy)</b>           | The LoRA fine-tuned model with greedy decoding, isolating the effect of supervised fine-tuning on science-heavy reasoning data independent of any inference-time selection strategy.                                                                                                                          |
| <b>Method 4: SFT Policy + Self-Consistency</b> | The fine-tuned model with $k = 5$ stochastic samples and majority voting, combining the benefits of domain adaptation with inference-time ensembling. Represents the strongest training-free inference method applied to our best generator.                                                                  |
| <b>Method 5: SFT + PPM Reranking</b>           | The fine-tuned model generates $k = 5$ candidates, and the PPM scores each complete trajectory. The solution with the highest PPM score is selected, replacing majority voting with learned preference-based selection. This tests whether a trained verifier can outperform simple answer consensus.         |
| <b>Method 6: SFT + PPM-Weighted Voting</b>     | A hybrid approach that combines self-consistency with PPM scores. For each unique answer among the $k$ candidates, we compute a weighted score blending vote share with normalized PPM scores. This tests whether PPM signal can complement rather than replace majority voting.                              |

Table 1: Evaluation methods used in our experiments.

For all sampling-based methods, we use temperature  $T = 0.7$  and top- $p = 0.9$  with up to 2048 generated tokens per solution. We additionally report Pass@ $k$ , the fraction of problems where at least one of the  $k$  candidates contains the correct answer, which represents the oracle upper bound for any selection method operating over the same candidate pool. The gap between Pass@ $k$  and realized accuracy quantifies the room for improvement in answer selection.

**Inference Pipeline.** At evaluation time, all  $k = 5$  candidates are generated in a single batched forward pass using num\_return\_sequences= 5, which is substantially faster than  $k$  sequential calls and allows both models to operate in parallel on separate GPUs without memory contention. Final answers are extracted by parsing the last `\boxed{\}` occurrence in each candidate, and the candidate assigned the highest PPM score is selected as the final prediction.

## 5 Experiments

### 5.1 Data

**Dataset.** We use the Gargantua-R1-Compact dataset, a large-scale collection of scientific reasoning problems with verified step-by-step solution traces spanning domains including physics, chemistry, thermodynamics, and electricity [9]. Each example contains a natural language problem and a multi-step reasoning, a final numerical answer.

**Preprocessing.** From 69,459 raw examples loaded from HuggingFace, we deduplicate by exact problem text and randomly shuffle the remaining pool before constructing an 80/10/10 train/validation/test split, 48,752 training and 6k examples each for validation and testing, no overlap between splits.

**Solution Reformatting.** To reduce formatting variance that could confuse the PPM reranker, we standardize all solution trajectories using a custom script. The script extracts final answers and reformats to a consistent `Final Answer: <value>` line at the end of each solution, discarding examples where no answer can be reliably extracted.

### 5.2 Evaluation Method

We evaluate all methods on a held-out test set of 88 problems drawn from the Gargantua-R1-Compact dataset, spanning physics, chemistry, thermodynamics, and mathematical reasoning domains. For sampling-based methods (Methods 2, 4, 5, 6), we generate  $k = 5$  candidate solutions per problem using temperature  $T = 0.7$  and top- $p = 0.9$ , with a maximum of 2048 tokens per solution.

We report four metrics. First, the accuracy, which is the match between the predicted and ground-truth answer after normalization. Second, the Pass@ $k$ , which is the fraction of problems where at least one of the  $k$  candidates contains the correct answer, representing the upper bound for any selection method. Third, the partial-answer rate, which is the fraction of predictions that partially match the ground truth. The fourth is the null rate, the fraction of problems yielding no extractable answer in format. Answer extraction follows a normalized string-matching protocol, with manual verification on a random subset to correct systematic formatting mismatches.

### 5.3 Experimental Details

**Compute Infrastructure.** All experiments were conducted on Google Cloud Platform using NVIDIA L4 (24GB) GPUs for training and T4 GPUs for evaluation.

**Policy Model (SFT).** LoRA fine-tuning (rank 16,  $\alpha = 32$ , dropout 0.05) on  $q/v$  projections for 1 epoch ( $\sim 7.5$  hours). Effective batch size 16, learning rate  $2 \times 10^{-4}$  with linear warmup, AdamW optimizer, and early stopping based on validation loss.

**PPM.** Longformer-base encoder trained for 3 epochs ( $\sim 1.5$  hours) on  $\sim 440$  preference pairs labeled by final-answer correctness. Batch size 4, learning rate  $1 \times 10^{-5}$ , gradient clipping at 1.0. Best checkpoint selected using validation pairwise accuracy.

**Generation.** Temperature  $T = 0.7$ , top- $p = 0.9$ , maximum 2048 tokens. Answers extracted via `\boxed{\}` and “Final Answer:” patterns.

**Reproducibility.** Fixed seed 42; all methods evaluated on the same 88-problem held-out test set.

### 5.4 Experimental Design

We evaluate six experimental settings designed to isolate the contributions of fine-tuning, self-consistency decoding, and process preference modeling, results in section 6.

Table 2 summarizes the goal, setup, and rationale for each experiment.

| Experiment                         | Goal                                                                                | Setup                                                                                                                                                     | Rationale                                                                                                                                                     |
|------------------------------------|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Baseline Evaluation (M1–M4)</b> | Establish base and SFT model performance under greedy and self-consistency decoding | Evaluate four configurations: (M1) base model greedy, (M2) base + self-consistency ( $k = 5$ majority vote), (M3) SFT greedy, (M4) SFT + self-consistency | Isolates the impact of fine-tuning (M1 vs M3, M2 vs M4) and self-consistency decoding (M1 vs M2, M3 vs M4), establishing baseline performance for comparison. |
| <b>PPM Reranking (M5)</b>          | Evaluate whether a learned preference model can outperform majority voting          | SFT model generates $k = 5$ solutions; Longformer-based PPM scores each trajectory and selects the highest-scoring solution                               | PPM evaluates reasoning quality rather than answer frequency, potentially recovering correct minority solutions discarded by voting.                          |
| <b>PPM-Weighted Voting</b>         | Test hybrid selection combining vote consensus and PPM scores                       | For each answer $a$ : $\text{score}(a) = \alpha \cdot \text{vote\_share}(a) + (1 - \alpha) \cdot \text{PPM\_share}(a)$ where $\alpha \in [0, 1]$          | If PPM reranking alone underperforms majority voting, a hybrid approach may leverage both consensus signals and trajectory quality.                           |
| <b>Answer-Signal Weighting</b>     | Test whether a simple answer-extraction signal improves PPM selection               | Binary signal detecting <code>\boxed{\}</code> or “Final Answer:” combined with normalized PPM score via weighted interpolation                           | Ensures selected trajectories contain extractable answers while retaining trajectory-level quality assessment.                                                |
| <b>Middle Truncation</b>           | Test whether PPM failures arise from truncation of final answers                    | Preserve first 20% and last 30% of solutions before scoring; evaluate at 1024 and 2048 tokens                                                             | Ensures the PPM sees final answers. If performance remains unchanged, truncation is not the primary failure mode.                                             |
| <b>K-Sweep Analysis</b>            | Analyze scaling behavior with larger candidate pools                                | Generate up to $k = 20$ candidates and evaluate majority voting, PPM reranking, and PPM-weighted voting at $k \in \{1, 3, 5, 10, 20\}$                    | Measures how performance scales with candidate diversity and compares methods against Pass@ $k$ oracle bounds.                                                |

Table 2: Summary of experimental settings used to evaluate the contributions of fine-tuning, self-consistency decoding, and process preference modeling.

## 6 Results

### 6.1 Experiment 1: Baseline Performance

The baseline evaluation reveals two key patterns:

| Method                           | Accuracy          | Pass@5            | Partial Answers | Null Rate  |
|----------------------------------|-------------------|-------------------|-----------------|------------|
| M1: Base LM (greedy)             | 37 / 42.0%        | -                 | 3 (3.4%)        | 30 (34.1%) |
| M2: Base LM (self-consistency)   | 64 / 72.7%        | 67 / 76.1%        | 1 (1.1%)        | 1 (1.1%)   |
| M3: SFT model (greedy)           | 41 / 46.6%        | -                 | 1 (1.1%)        | 42 (47.7%) |
| M4: SFT model (self-consistency) | <b>69 / 78.4%</b> | <b>71 / 80.7%</b> | 1 (1.1%)        | 4 (4.5%)   |
| M5: SFT + PPM rerank             | 49 / 55.7%        | 69 / 77.3%        | 1 (1.1%)        | 23 (26.1%) |

Table 3: Main evaluation results across all five methods (88 test problems).

**SFT provides consistent gains.** Fine-tuning improves greedy accuracy by +5.6pp (M1→M3: 36.4%→42.0%) and self-consistency accuracy by +8.0pp (M2→M4: 62.5%→70.5%). This confirms that domain-specific fine-tuning on science-heavy data improves the base math model’s scientific reasoning capabilities.

**Self-consistency is highly effective.** Majority voting over  $k = 5$  samples provides dramatic improvements for both base (+26.1pp) and SFT (+28.5pp) models, nearly doubling accuracy in both cases. M4 (SFT + Self-Consistency) achieves the best overall accuracy at 70.5%, establishing our primary baseline.

The gap between M4 accuracy (70.5%) and Pass@5 (75.0%) indicates that 4.5pp of potential improvement remains, ie that correct answers exist in the candidate pool but are not selected by majority voting.

## 6.2 Experiment 2: PPM Reranking

PPM reranking (M5) achieves 52.3% accuracy, a meaningful +10.3pp improvement over SFT greedy (M3: 42.0%), demonstrating that the preference model learned some useful signal for solution quality. However, M5 underperforms majority voting by 18.2pp (52.3% vs 70.5%) despite identical Pass@5 rates (75.0%). This confirms that correct answers exist in the candidate pool at the same rate, but the PPM fails to select them reliably. The PPM actively misselects, choosing incorrect solutions over correct ones.

## 6.3 Experiment 3: PPM-Weighted Voting

Table 3 presents results from the alpha sweep blending vote share with PPM scores. Exact numbers are in table B.1. Experiment 3 in the Appendix. The optimal blend at  $\alpha = 0.85$ – $0.95$  achieves 71.6% accuracy, a modest +1.1pp improvement over pure majority voting. This represents just 1 additional correct problem out of 88. Notably, mid-range alphas (0.35–0.45) actually decrease accuracy to 69.3%, revealing that the PPM introduces noise when given substantial weight. The PPM has not learned a quality signal meaningfully different from what vote counting already captures.

## 6.4 Experiment 4: Answer Signal Weighting

Combining the binary answer signal with PPM scores yields more stable results than raw PPM weighting. The answer signal provides a +10.2pp improvement over PPM-only selection (55.7%→65.9%). When combined with vote share, accuracy reaches 71.6%, matching the best alpha-sweep result. Critically, unlike raw PPM scores, the answer signal never hurts performance across any alpha value tested. The table of these results is in Appendix under table B.2 Experiment 4.

## 6.5 Experiment 5: Middle Truncation

Middle truncation with 2048-token context achieves a PPM-only accuracy of 55.7%, a weighted accuracy ( $\alpha = 0.3$ ) of 65.9%, and a Pass@5 of 79.5%. These results are comparable to non-truncated evaluation, indicating that context truncation is not the primary cause of PPM failures. Guaranteeing that the PPM sees the final answer does not improve its discrimination ability. This suggests the PPM’s limitations stem from the training signal (trajectory-level correctness labels) rather than architectural constraints. The model has not learned to use answer information even when it is visible.

## 6.6 Experiment 6: K-Sweep Analysis

Table B.3. in the appendix presents accuracy across varying numbers of candidate solutions.

Two critical patterns emerge:

**Majority voting plateaus while Pass@ $k$  continues climbing.** At  $k = 20$ , majority voting achieves 66.7% while Pass@ $k$  reaches 83.3%—a 16.6pp gap representing correct answers that exist in the candidate pool but are not selected.

**PPM performance degrades with more candidates.** Counter to our hypothesis, PPM-only accuracy decreases from 56.7% at  $k = 1$  to 46.7% at  $k = 20$ . PPM-weighted voting shows even sharper degradation: 56.7%→33.3%. Thus, more answers are not needed since the PPM performs worse, letting us know to focus on reasoning and not data amount for improvement.

## 7 Analysis

Our experiments reveal several consistent patterns about when and why the Process Preference Model (PPM) succeeds or fails. Overall, SFT combined with self-consistency (M4) achieves the strongest performance at 78.4% accuracy, substantially outperforming all other configurations. In contrast, the PPM reranker (M5) achieves 55.7% accuracy and fails to improve over simple answer aggregation. The results suggest that the main limitation lies not in model architecture or inference-time compute, but in the quality of the preference supervision used to train the PPM.

**Preference supervision is the primary bottleneck.** The PPM achieves 56.2% pairwise accuracy when the PPM has to distinguish between preference pairs. With approximately 440 training pairs labeled solely by final-answer correctness, the supervision signal is weak, limiting the model’s ability to learn reliable trajectory ranking.

**Inference-time factors do not explain the gap.** Middle-truncation experiments show that guaranteeing answer visibility does not improve PPM discrimination. Similarly, increasing the candidate pool harms performance: as  $k$  grows, PPM accuracy declines even though correct answers appear more frequently in the pool.

**A large gap to oracle selection remains.** At  $k = 20$ , Pass@ $k$  reaches 83.3%, yet the best selection strategy achieves only 70.0% accuracy, leaving a 13.3pp gap between available correct candidates and those actually selected.

**SFT introduces mixed effects.** Fine-tuning produces a modest greedy accuracy improvement (+4.6pp from M1 to M3) but substantially increases the null-answer rate from 34.1% to 47.7%. In nearly half the test cases the fine-tuned model produces outputs from which no answer can be extracted. This appears to stem from the preprocessing heuristic that appended the final numerical token as Final Answer : <number>, introducing noisy supervision for symbolic or structured answers.

**Self-consistency mitigates this instability.** Sampling multiple trajectories dramatically reduces null outputs because majority voting requires only one candidate containing an extractable answer. Under self-consistency the SFT model’s null rate drops to 4.5%, enabling large accuracy gains and producing the strongest configuration in our evaluation.

**PPM reranking fails primarily during candidate selection.** Per-problem comparison between M4 and M5 shows that out of 88 problems, both methods succeed on 43 (48.9%), M4 alone succeeds on 19 (21.6%), M5 alone succeeds on 3 (3.4%), and both fail on 23 (26.1%). Thus, PPM reranking causes a net regression on 16 problems.

The failures largely fall into two categories. First, the PPM sometimes assigns its highest score to candidates that produce no extractable answer even when correct answers exist in the pool. Second, when multiple candidates agree on the correct answer, the PPM frequently selects the single outlier candidate. This suggests that the model is rewarding unusual trajectory features rather than correctness.

### 7.1 Problem-Type Breakdown

The table indicating problem type breakdowns given method and problem type is inserted under B.4. Problem Type Breakdown in the appendix. The regression is concentrated primarily on numeric

problems. For integer-valued answers, M4 achieves 95.5% accuracy while M5 drops to 59.1%, a 36pp gap. A similar but smaller gap appears for fractional answers. These settings exhibit high candidate agreement, making the PPM’s tendency to favor outliers particularly harmful. In contrast, multiple-choice and equation-based questions show smaller differences between methods.

## 7.2 PPM Scoring

Inspection of raw PPM scores reveals a numerical artifact contributing to these failures. Exactly 30 of 440 candidate scores (6.8%) are clamped to a floor value of -3.498; 27 (90%) correspond to correct answers, while none correspond to null candidates. This systematically pushes correct solutions to the bottom of the ranking. Even beyond this artifact, score calibration appears weak. The mean score for selected null candidates (-0.183) is close to that of selected answer-bearing candidates (+0.039), suggesting the model struggles to distinguish informative trajectories from empty ones.

Overall, these results point to limitations in the PPM training signal. Because preference pairs use only final-answer correctness to distinguish themselves, adding richer supervision, such as step-level preferences or harder negatives, could further improve its ability to understand methodology as opposed to proxy reasoning.

## 8 Conclusion

This work investigated whether a Process Preference Model (PPM) trained on pairwise preferences can improve solution selection for scientific reasoning beyond self-consistency voting in small language models. Three main findings emerge. (1) Domain fine-tuning provides a consistent 5–8pp accuracy improvement, indicating math reasoning transfers to scientific domains. Second, SFT combined with self-consistency establishes a strong baseline at 78.4% accuracy. Third, the PPM learns useful signal for quality, improving over greedy decoding (+10pp), and Pass@5 rates (77–81%) confirm that small models frequently generate correct answers. The primary challenge therefore lies in selecting among candidates, supporting reranking as a promising direction.

### 8.1 Limitations

Several constraints limited the robustness of our findings. Limited GPU availability restricted several experiments, with the k-sweep analysis covering only 30 of 88 test problems, and preference pairs were truncated to 2048 tokens, potentially discarding useful intermediate reasoning steps. Training data was also scarce, with only  $\sim 400$  usable preference pairs available due to answer convergence and similarity, limiting exposure to strongly discriminative examples. Additionally, the SFT policy model and PPM were trained independently, coupling the PPM’s training distribution to a single generator. A more aggregated pipeline would better capture reasoning diversity.

### 8.2 Future Work

Immediate improvements include fixing the  $-3.498$  score-floor artifact and improving answer normalization during SFT data preparation. More broadly, three directions appear promising: (1) step-level preference construction to provide finer supervision on reasoning quality, (2) cross-model preference generation to produce harder and more diverse negative examples, and (3) joint or iterative training in which improved PPM selection feeds back into policy model learning. These directions address the central finding of this work: that preference supervision quality, rather than model architecture, is the main constraint on PPM effectiveness.

## Team contributions

Marianne handled all data loading, preprocessing, finetuned base policy model and solution reformatting via Gargantua dataset. Arya constructed the preference pair dataset, and designed the PPM. Both team members trained the PPM, conducted evaluation on all methods, and wrote the final research report.

## References

- [1] Tianshi Zheng, Zheyang Deng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Zihao Wang, and Yangqiu Song. From automation to autonomy: A survey on large language models in scientific discovery. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17744–17761, 2025.
- [2] Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers, 2024. URL <https://arxiv.org/abs/2408.06195>.
- [3] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The twelfth international conference on learning representations*, 2023.
- [4] Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. Rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.
- [5] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [6] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.
- [7] Edward J Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [8] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.
- [9] Prithiv Sakthi. Gargantua-r1-compact (revision 522d6fb), 2025.

## A Appendix (optional)

## B Additional Tables

### B.1 Experiment 3

| $\alpha$                  | Accuracy             | Interpretation        |
|---------------------------|----------------------|-----------------------|
| 0.00 (pure PPM-weighted)  | 62/88 (70.5%)        | PPM signal only       |
| 0.35–0.45                 | 61/88 (69.3%)        | PPM hurts performance |
| 0.85–0.95                 | <b>63/88 (71.6%)</b> | Optimal blend         |
| 1.00 (pure majority vote) | 62/88 (70.5%)        | Vote signal only      |

Table 4: Alpha sweep results blending vote share ( $\alpha$ ) with PPM share ( $1 - \alpha$ ).

### B.2 Experiment 4

| Configuration                             | Accuracy      |
|-------------------------------------------|---------------|
| PPM-only (no answer signal)               | 49/88 (55.7%) |
| Weighted ( $\alpha = 0.3$ ): answer + PPM | 58/88 (65.9%) |
| Vote + answer-weighted PPM                | 63/88 (71.6%) |

Table 5: Answer signal weighting results.

### B.3 Experiment 6

| $k$ | Majority Vote | PPM Only | PPM Weighted | Pass@ $k$ |
|-----|---------------|----------|--------------|-----------|
| 1   | 56.7%         | 56.7%    | 56.7%        | 56.7%     |
| 3   | 66.7%         | 50.0%    | 60.0%        | 70.0%     |
| 5   | 66.7%         | 50.0%    | 53.3%        | 70.0%     |
| 10  | 70.0%         | 50.0%    | 43.3%        | 73.3%     |
| 20  | 66.7%         | 46.7%    | 33.3%        | 83.3%     |

Table 6: K-sweep results showing accuracy vs. number of candidate solutions.

### B.4 Problem Type Breakdown

## References

- [1] Tianshi Zheng, Zheyang Deng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Zihao Wang, and Yangqiu Song. From automation to autonomy: A survey on large language models in scientific discovery. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17744–17761, 2025.
- [2] Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers, 2024. URL <https://arxiv.org/abs/2408.06195>.
- [3] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The twelfth international conference on learning representations*, 2023.
- [4] Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. Rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.
- [5] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

| Problem Type          | Number of Problems | M1    | M2    | M3    | M4    | M5    |
|-----------------------|--------------------|-------|-------|-------|-------|-------|
| Numeric (Integer)     | 22                 | 81.8% | 90.9% | 81.8% | 95.5% | 59.1% |
| Numeric (Fraction)    | 8                  | 62.5% | 75.0% | 37.5% | 87.5% | 62.5% |
| MCQ                   | 42                 | 11.9% | 54.8% | 35.7% | 66.7% | 59.5% |
| Equation/Formula      | 3                  | 33.3% | 33.3% | 0.0%  | 66.7% | 66.7% |
| Algebraic Expression  | 5                  | 20.0% | 40.0% | 0.0%  | 20.0% | 20.0% |
| Short Phrase/Question | 4                  | 25.0% | 50.0% | 25.0% | 25.0% | 0.0%  |
| Open-Ended            | 2                  | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  |

Table 7: Accuracy across all methods by problem type.

| Configuration       | Accuracy      |
|---------------------|---------------|
| PPM-only            | 49/88 (55.7%) |
| Answer + PPM        | 58/88 (65.9%) |
| Vote + Answer + PPM | 63/88 (71.6%) |

Table 8: Answer-signal weighting results.

- [6] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.
- [7] Edward J Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [8] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.
- [9] Prithiv Sakthi. Gargantua-r1-compact (revision 522d6fb), 2025.