Verifiable Counterfactual Supervision for Process Reward Models

Anonymous ACL submission

Abstract

Process reward models (PRMs) require supervision that identifies not only whether a reasoning trajectory is correct, but also where the reasoning process first becomes unsupported by its prefix. We frame this requirement as verifiable counterfactual process supervision with paired correct and erroneous trajectories in which the first invalid transition is known, the error mechanism is controlled, and the downstream continuation remains coherent under the corrupted state. Starting from a verified symbolic reasoning chain, our method injects a template-aware error at a selected intermediate step, recomputes all subsequent steps under the corrupted state, and verifies that the injected step is not derivable from its original prefix. The resulting trajectories provide prefix-valid first-error annotations and are translated into aligned natural-language processes for PRM training and evaluation. Experiments show that the synthesized data improve Best-of-8 reranking on logical reasoning benchmarks and show preliminary transfer to mathematical process evaluation.

1 Introduction

Large language models (LLMs) have shown strong performance on multi-step reasoning tasks, especially when they are prompted to produce intermediate reasoning traces (Wei et al., 2022). However, final-answer accuracy alone does not reveal whether the intermediate reasoning process is reliable. Process supervision addresses this limitation by providing step-level feedback, and has been shown to improve reasoning quality in complex settings (Uesato et al., 2022; Lightman et al., 2024). In this setting, process reward models (PRMs) and step-level verifiers are increasingly used for generation, reranking, and search (Cobbe et al., 2021; Zhang et al., 2024; Setlur et al., 2024).

The central challenge in process supervision is not simply obtaining more reasoning traces, but obtaining *negative traces* with *reliable* causal structure. A negative reasoning trace is useful for PRM training only if it answers three questions: *where* did the reasoning first go wrong, *why* is that transition unsupported by the prefix, and *how* should the subsequent trajectory evolve after the corrupted state is introduced? Existing sources of process supervision only partially provide this structure. Manual annotation can identify erroneous steps, but it is expensive and hard to scale (Lightman et al., 2024). Rollout- and search-based annotation (Wang et al., 2024a; Luo et al., 2024) reduce the cost, but they usually estimate step quality indirectly from final outcomes and offer limited control over the first-error position, error type, and downstream error propagation.

Identifying the first unsupported step in a reasoning trajectory requires knowing whether a step follows from its prefix. Symbolic systems provide exactly the machinery needed to test this derivability relation. Recent work combined LLMs with symbolic provers to construct logically grounded reasoning data (Xu et al., 2024; Leang et al., 2025; Qi et al., 2025), which can be used for training and evaluating reasoning generation. We utilize this line of work to instead synthesize negative process data as *first-error counterfactuals*: trajectories that share a verified valid prefix with a correct chain, deviate at a controlled and verifiably unsupported step, and then continue under the corrupted state.

Our contributions are:

- We formulate verifiable counterfactual supervision for PRMs, where each negative trajectory has a known prefix-unsupported transition.
- We propose a symbolic synthesis pipeline that combines template-aware error injection, prefix non-derivability checking, and downstream recomputation under the corrupted state.
- We show that the resulting data improves Best-

of-8 reranking on logical reasoning benchmarks and provides a diagnostic testbed for first-error localization.

2 Related Work

Process supervision trains models to evaluate intermediate reasoning steps rather than only final answers (Uesato et al., 2022; Lightman et al., 2024). PRMs have been used for reranking, search, and reasoning-time selection (Cobbe et al., 2021; Zhang et al., 2024; Setlur et al., 2024). However, recent benchmarks show that fine-grained process evaluation, especially first-error localization, remains difficult (Zheng et al., 2025; Song et al., 2025).

Manual annotation provides reliable step labels but is expensive (Lightman et al., 2024). Rollout- and search-based methods reduce annotation cost (Wang et al., 2024a; Luo et al., 2024), but usually infer step quality indirectly from final outcomes and offer limited control over error type, error position, and downstream consistency. Direct synthesis is more scalable (Zelikman et al., 2022; Zeng et al., 2025), but existing methods generally do not enforce verifiable first-error counterfactual structure.

Recent work combines LLMs with symbolic provers to construct logically grounded reasoning data (Xu et al., 2024; Leang et al., 2025; Qi et al., 2025). These methods primarily focus on verified correct chains. In contrast, we use symbolic verification to synthesize controlled erroneous chains for PRM supervision.

3 Method

We first formalize the notion of first-error counterfactual supervision for PRMs, and then describe our synthesis framework as illustrated in Figure 1.

3.1 Problem Formulation

Let $C = (s_1, \ldots, s_T)$ be a reasoning trajectory, where each step s_t consists of a set of supporting facts, an applied rule, and a conclusion. Let $\mathcal{P}_t$ denote the prefix state induced by the initial facts and previous steps $s_{<t}$. We say that s_t is *prefix-valid* if its conclusion is derivable from $\mathcal{P}_t$ under the rule set.

A first-error counterfactual instance is a tuple (G, C^+, C^-, k, e), where G is the target goal, C^+ is a verified correct chain, C^- is a corrupted chain, k is the first-error position, and e is the error type. The corrupted chain should satisfy $s_t^- = s_t^+$ for all $t < k$, s_k^- is prefix-invalid with respect to $\mathcal{P}_k$ and error type e, and downstream steps $s_{>k}^-$ are recomputed under the corrupted state induced by s_k^-. Thus, the corrupted trajectory is invalid with respect to the original problem state, but remains coherent as a counterfactual continuation after the injected error.

3.2 Correct Chain Synthesis

We first generate a correct symbolic reasoning chain from a rule template pool. Starting from the target conclusion, we recursively expand intermediate goals and use a symbolic prover to verify whether each step is derivable from the current premises. Only truth-consistent structures are retained, yielding a verifiable forward chain from base facts to the target.

3.3 Error Injection and Symbolic Re-computation

Given a verified correct chain $C^+ = (s_1, \ldots, s_T)$, we construct an erroneous counterpart by first selecting an intermediate step k as the candidate first-error position. The perturbation type e is constrained by the rule template instantiated at step k, ensuring that the injected error is structurally compatible with the local inference pattern rather than sampled from a global error distribution. Depending on the template, the perturbation may alter the inferred conclusion, misuse the rule direction, or corrupt the local factual state.

The corrupted conclusion produced at step k is then treated as part of the subsequent symbolic state. We recompute all downstream steps under this updated state with the symbolic prover. The resulting erroneous chain C^- preserves the correct prefix before step k, replaces s_k with a corrupted step, and recomputes all subsequent steps from the corrupted state. It is essentially a counterfactual trajectory that shares the same initial conditions and rule applications as the original chain, but with a controlled error injected at step k and propagated through the rest of the trajectory.

To verify that the injected step is indeed the first invalid step, we apply error-type-specific checks. For non-structural errors, we perform a prefix non-derivability check by testing whether the corrupted conclusion $\tilde{s}_k$ is derivable from the original prefix $(s_1, \ldots, s_{k-1})$, and the sample is rejected if it is still derivable. For structural errors, we instead verify that the corrupted step violates the intended rule direction, dependency order, or acyclic trajectory

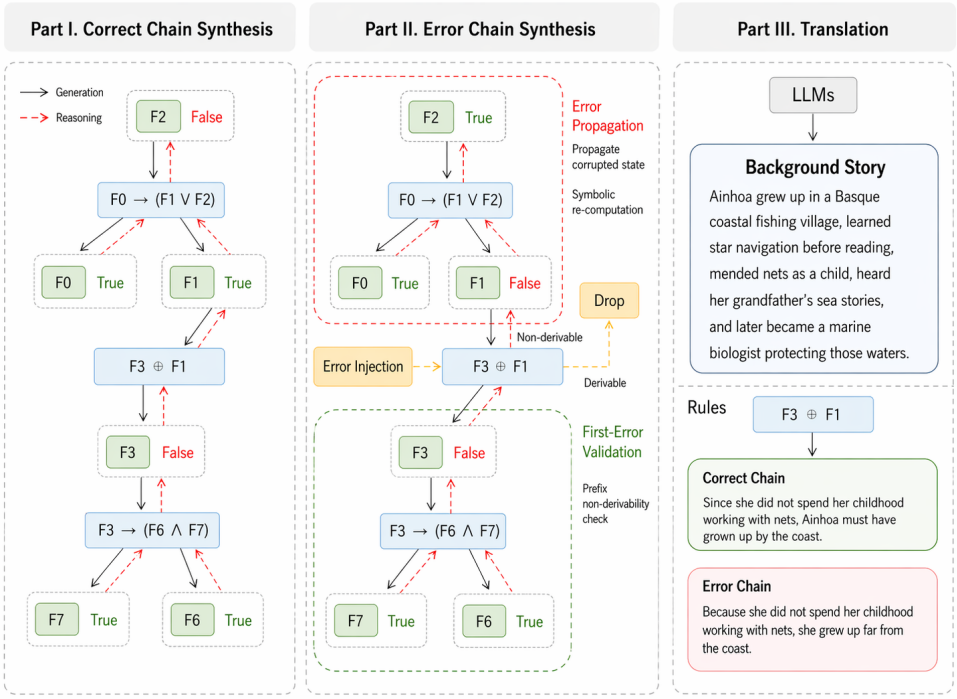

Figure 1: Overview of the proposed verifiable counterfactual process supervision framework for PRMs.

constraints, since the local truth values may still be compatible with the instantiated rule. Detailed error type distributions and definitions are provided in Appendices E and F.

3.4 Natural-Language Translation

Finally, we convert the symbolic trajectories into natural-language reasoning processes. To preserve comparability between the correct and erroneous chains, both translations are conditioned on the same background context, entity realizations, and predicate mappings. This shared semantic grounding ensures that the two processes differ primarily in their reasoning behavior rather than in unrelated surface realizations.

4 Experiments

4.1 Experimental Setup

We evaluate our approach through three settings: Best-of-K reranking for logical reasoning, transfer evaluation on mathematical reasoning, and step-level evaluation on our synthesized 20K process supervision data.

We use Llama-3.1-8B (Grattafiori et al., 2024) and Qwen-2.5-7B (Qwen Team, 2024). For Best-of-8 reranking, the candidate solutions are generated by the corresponding base model and then selected by a scorer. Thus, improvements reflect better trajectory selection rather than stronger candidate generation.

For comparison, we include domain-adapted baselines Llama-3.1-8B-D and Llama-3.1-8B-M trained on DeepSeek-style and Mistral-style process data(Guo et al., 2025; Jiang et al., 2023), as well as Qwen-2.5-7B-math and Qwen-2.5-7B-math-800K, which are publicly released math-adapted variants (Yang et al., 2024). More details are provided in the Appendix A.

4.2 Main Results on Logical Reasoning

Tables 1 report Best-of-8 reranking results on logical reasoning benchmarks. Across both model families, our PRM improves the average score over the corresponding base selector. For Llama-3.1-8B, the average score increases from 0.528 to 0.591, and for Qwen-2.5-7B, it increases from 0.567 to 0.615. Compared with the strongest domain-adapted baseline in each family, our PRM still obtains the best average performance.

The improvements are greatest on datasets such as folio, pro-nomath, anli, and hans for Llama, and folio, logicnli, pro-nomath, temporal, and hans for Qwen. At the same time, domain-adapted baselines remain stronger on some tasks, such as tracking. These results suggest that first-error counterfactual supervision learns a selection signal that might be complementary to domain-adapted process data.

Table 1: Best-of-8 reranking results on logical reasoning benchmarks. Candidate solutions are generated by the Llama-3.1-8B and Qwen-2.5-7B base models, respectively. Best results are bolded.

PRM	folio	logicnli	pro_nomath	anli	temporal	tracking	hans	avg
llama-3.1-8B	0.575	0.472	0.498	0.729	0.412	0.391	0.620	0.528
llama-3.1-8B-D	0.519	0.489	0.528	0.748	**0.611**	**0.488**	0.644	0.575
llama-3.1-8B-M	0.516	**0.490**	0.491	0.752	0.598	0.384	0.670	0.557
llama-prm (ours)	**0.625**	0.473	**0.584**	**0.772**	0.569	0.396	**0.721**	**0.591**
qwen-2.5-7B	0.543	0.456	0.409	0.655	0.659	0.543	0.710	0.567
qwen-2.5-7B-math	0.512	0.494	0.445	**0.701**	0.681	**0.667**	0.771	0.610
qwen-2.5-7B-math-800K	0.524	0.514	0.442	0.655	0.679	0.635	0.763	0.601
qwen-prm (ours)	**0.584**	**0.527**	**0.458**	0.691	**0.712**	0.562	**0.780**	**0.615**

Table 2: Transfer evaluation (F1 scores) on mathematical reasoning tasks in ProcessBench.

Model	gsm8k	math	olymp.	omni.	avg
llama-3.1-8B	3.8	3.0	5.3	4.1	4.1
llama-3.1-8B-5k	18.0	10.3	4.1	7.9	10.0
llama-3.1-8B-20k	**21.2**	**21.3**	**18.6**	**16.0**	**19.2**

Table 3: Step-level evaluation on our process supervision data.

Model	first_error	all_step
gpt-4o	0.160	0.479
gpt-5	0.575	0.866
kimi-k2.5	0.103	0.632
DeepSeek-V3.2	0.333	0.608
gemini-3.1	0.825	0.941

We additionally compare against Oracle@8 in Appendix C, where the gap indicates that process-level supervision still has room for improvement in selecting the best candidate.

4.3 Transfer to Mathematical Reasoning

We further test whether the process-level signal learned from symbolic trajectories transfers to a different reasoning domain. Table 2 reports F1 scores on the mathematical subsets of ProcessBench (Zheng et al., 2025). Although the PRM is trained on synthesized symbolic process data rather than mathematical derivations, increasing the amount of synthesized data consistently improves performance. The average F1 rises from 4.1 for the base model to 10.0 with 5K instances and 19.2 with 20K instances. The absolute scores remain low, but the monotonic improvement from 5K to 20K suggests that prefix-validity supervision may provide a weak but nontrivial signal beyond the synthetic symbolic domain.

4.4 Step-level Evaluation on Our Data

Table 3 evaluates models on the fine-grained annotations provided by our synthesized data. Each model is tested on 100 synthesized trajectories with 7–10 reasoning steps. We report `all_step` measuring whether the model correctly classifies each step as valid or invalid, and `first_error` measuring whether the model correctly identifies the first unsupported step in the trajectory.

The results show a large gap between these two abilities. Even the strongest evaluated model, Gemini-3.1, has a lower first-error accuracy (0.825) than all-step accuracy (0.941). This confirms that first-error localization is not merely ordinary step classification. A model recognizing individual steps as plausible or implausible may still fail to identify the earliest unsupported transition.

5 Conclusion

We introduced verifiable first-error counterfactual supervision for process reward models. Rather than treating negative process data as independently labeled erroneous steps, our framework constructs paired correct and corrupted trajectories in which the first unsupported transition is controlled, verified, and propagated through a coherent counterfactual continuation. This design turns symbolic verification into a supervision mechanism for PRMs. Experiments on logical reasoning benchmarks show that the resulting data improves Best-of-8 reranking, and our diagnostic evaluation suggests that first-error localization remains substantially more challenging than ordinary step-level validity classification. These findings indicate that reliable negative process supervision is a useful direction for building more fine-grained reasoning evaluators.

Limitations

While our framework improves controllability and verifiability, it has several limitations. First, the diversity of synthesized errors is still limited by the predefined templates and may not fully cover the range of errors in open-ended LLM reasoning. In addition, the current framework is built on symbolic reasoning structures, which makes it most naturally applicable to domains with relatively explicit logical form. Finally, while the current results are encouraging, evaluating the framework on broader tasks and model families would further strengthen its generality.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2024. Omni-MATH: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, and Angela Fan. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, and Brian Wong. 2024. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031, Miami, Florida, USA. Association for Computational Linguistics.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Joshua Ong Jun Leang, Giwon Hong, Wenda Li, and Shay B. Cohen. 2025. Theorem prover as a judge for synthetic data generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29941–29977, Vienna, Austria. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901, Online. Association for Computational Linguistics.

Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. 2025. Large language models meet symbolic provers for logical reasoning evaluation. In *The Thirteenth International Conference on Learning Representations*.

Qwen Team. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2024. Rewarding progress: Scaling automated process verifiers for LLM reasoning. *arXiv preprint arXiv:2410.08146*.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. 2025. PRMBench: A fine-grained and challenging benchmark for process-level reward models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25299–25346, Vienna, Austria. Association for Computational Linguistics.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2022. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. 2021. Diagnosing the first-order logical reasoning ability through LogicNLI. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3738–3747, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024a. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024b. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. 2024. Faithful logical reasoning via symbolic chain-of-thought. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13326–13365, Bangkok, Thailand. Association for Computational Linguistics.

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. 2024. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pages 15476–15488.

Thomas Zeng, Shuibai Zhang, Shutong Wu, Christian Classen, Daewon Chae, Ethan Ewer, Minjae Lee, Heeju Kim, Wonjun Kang, Jackson Kunde, Ying Fan, Jungtaek Kim, Hyung Il Koo, Kannan Ramchandran, Dimitris Papailiopoulos, and Kangwook Lee. 2025. VersaPRM: Multi-domain process reward model via synthetic reasoning data. *arXiv preprint arXiv:2502.06737*.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2024. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. ProcessBench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1009–1024, Vienna, Austria. Association for Computational Linguistics.

A Dataset and Evaluation Details

This appendix provides additional implementation and evaluation details to support reproducibility.

For logical reasoning evaluation, we use FOLIO (Han et al., 2024), LogicNLI (Tian et al., 2021), ANLI (Nie et al., 2020), and HANS (McCoy et al., 2019), together with three additional logic-oriented tasks: Pro-NoMath (Wang et al., 2024b), Temporal, and Tracking (Suzgun et al., 2022). These datasets cover different forms of reasoning, including first-order logical inference, natural language inference, heuristic-sensitive inference, temporal reasoning, and object/state tracking. For transfer evaluation, we use GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), and Omni-MATH (Gao et al., 2024). The logical reasoning tasks are used for Best-of-8 reranking, while the mathematical reasoning tasks are used to test whether the learned process-level signal transfers beyond the symbolic setting used for data synthesis.

Dataset licenses and usage. All external datasets used in this work are public research benchmarks and are used only for offline evaluation or Best-of-8 reranking. We do not use their raw examples in our synthesized PRM training data and do not redistribute them. We cite the original papers and list the public repositories in Table 4. License or access-term information was checked from public repository metadata when available; for datasets without clearly specified license information, we restrict our use to research evaluation through the original access channel. No private user data or newly collected personal information is introduced by our preprocessing.

Dataset	Use	Repository
FOLIO	Logic	`tasksource/folio`
LogicNLI	Logic	`tasksource/LogicNLI`
ANLI	Logic	`facebook/anli`
HANS	Logic	`jhu-cogsci/hans`
Pro-NoMath	Logic	`sam-paech/mmlu-pro-nomath`
Temporal	Logic	`Fhrozen/big_bench_hard`
Tracking	Logic	`Fhrozen/big_bench_hard`
GSM8K	Math	`openai/gsm8k`
MATH	Math	`EleutherAI/hendrycks_math`
OlympiadBench	Math	`math-ai/olympiadbench`
Omni-MATH	Math	`KbsdJames/Omni-MATH`

Table 4: Evaluation datasets and their Hugging Face repositories. "Logic" denotes Best-of-8 logical reasoning reranking, while "Math" denotes transfer evaluation on mathematical reasoning tasks.

Category	Parameter	Value
Model	Format	LM-head PRM
LoRA	Rank / alpha / dropout	16 / 32 / 0.1
Objective	Loss	Pairwise ranking
Objective	Negative strategy	`all_after_error`
Optimization	Optimizer	AdamW
Optimization	Learning rate	2×10^{-5}
Optimization	Weight decay	0.01
Optimization	Warmup ratio	0.03
Optimization	Epochs	1
Batching	Max length	1024
Hardware	GPUs	4× RTX 4090
Selection	Metric	`first_error_acc`

Table 5: Key training hyperparameters for PRM fine-tuning.

B Training Details

We fine-tune PRMs in an LM-head format with LoRA adaptation. The model receives a reasoning prefix and a candidate step, and assigns scores to the positive and negative label words. We use `true`/`false` as the default label words. The training objective is a pairwise ranking loss over the positive and negative label scores, so that valid steps are encouraged to receive higher scores than invalid ones.

For supervision construction, we use the `all_after_error` strategy: the first corrupted step and all following steps in the erroneous trajectory are treated as negative with respect to prefix-valid reasoning from the original problem state. Downstream steps are still recomputed under the corrupted state to preserve trajectory consistency, but they are labeled negative because their prefix has already become invalid. The best checkpoint is selected according to first-error accuracy on the validation set. All training runs are conducted on four NVIDIA RTX 4090 GPUs.

C Best-of-8 Generation, Evaluation Protocol, and Additional Results

For each test instance, we pair the input problem with eight sampled candidate solutions from the corresponding base model. During reranking, the PRM scores every candidate trajectory step by step. We aggregate step-level positive scores with the `min` rule, which makes the trajectory score sensitive to the weakest step in the reasoning chain. The highest-scoring trajectory is selected as the final prediction.

We also report two reference selectors. **Major-**

Family	Selector	folio	logicnli	pro_nomath	anli	temporal	tracking	hans	avg
Llama	prm (ours)	0.625	0.473	0.584	0.772	0.569	0.396	0.721	0.591
Llama	Majority@8	0.525	0.387	0.470	0.790	0.720	0.560	0.811	0.574
Llama	Oracle@8	0.900	0.852	0.722	0.933	0.604	0.696	0.948	0.860
Qwen	prm (ours)	0.584	0.527	0.458	0.691	0.712	0.562	0.780	0.615
Qwen	Majority@8	0.557	0.392	0.404	0.802	0.680	0.313	0.632	0.540
Qwen	Oracle@8	0.702	0.802	0.583	0.909	0.824	0.812	0.968	0.800

Table 6: Dataset-level Best-of-8 comparison among our PRM, Majority@8, and Oracle@8. Candidate solutions are generated by the corresponding base model in each family.

Model	folio	logicnli	pro_nomath	anli	temporal	tracking	hans	avg
llama-3.1-8B	0.575	0.472	0.498	0.729	0.412	0.391	0.620	0.528
llama-3.1-8B-5k	0.582	0.436	0.558	0.758	0.461	0.374	0.652	0.545
llama-3.1-8B-20k	**0.625**	**0.473**	**0.584**	**0.772**	**0.569**	**0.396**	**0.721**	**0.591**

Table 7: Best-of-8 reranking results on logical reasoning benchmarks under different training data scales. Candidate solutions are generated by the Llama-3.1-8B base model.

ity@8 selects the answer that appears most frequently among the eight sampled candidates. It provides a simple self-consistency baseline that does not use step-level supervision. **Oracle@8** checks whether the candidate pool contains at least one correct solution, and therefore estimates the upper bound imposed by the sampled candidates. The gap between our PRM and Oracle@8 reflects remaining headroom in step-level scoring and trajectory selection.

Table 6 reports the full dataset-level comparison with Majority@8 and Oracle@8. On both model families, our method improves over Majority@8 on average, suggesting that the learned process reward provides useful selection signals beyond answer-level voting. At the same time, the gap to Oracle@8 remains substantial, indicating considerable room for improvement in candidate selection.

D Effect of Training Data Scale

We further analyze the effect of synthesized data scale on logical reasoning. Table 7 compares the base Llama-3.1-8B model with models trained using 5k and 20k synthesized instances. The average score rises from **0.528** for the base model to **0.545** with 5k training instances, and further to **0.591** with 20k instances. This trend suggests that the proposed synthesis framework continues to provide useful supervision as more process data are added. Although gains vary across datasets, the consistent average improvement indicates that scaling synthesized process data benefits PRM-based reranking.

Error Type	Count	Share
`xor_as_equiv`	3610	18.1%
`xor_as_or`	3609	18.0%
`or_and_confusion`	3598	18.0%
`drop_condition`	1934	9.7%
`implication_misuse`	1466	7.3%
`converse_error`	1299	6.5%
`redundant_step`	1185	5.9%
`circular_reference`	946	4.7%
`partial_evaluation`	913	4.6%
`missing_prerequisite`	869	4.3%
`vacuous_truth_error`	571	2.9%

Table 8: Error-type distribution in the final 20k training set.

E Error Type Distribution

Table 8 summarizes the error-type distribution in the 20k training set. The *Count* column reports the final number of instances used for training. We adjust the distribution to reduce the dominance of frequent template-induced errors and to make the training set more balanced across major error families. The *Share* column is computed from these counts, which sum to 20,000.

The distribution is intentionally not perfectly uniform. We keep relatively frequent truth-state corruptions such as XOR-related errors and AND/OR confusion because they arise naturally from the symbolic rule templates, while retaining less frequent structural errors such as missing prerequisites and circular references to expose the model to trajectory-level failures.

Group	Main criterion
Truth-state errors	The corrupted step either violates the local truth condition or current state, or performs an unauthorized local update.
Structural errors	The local conclusion may be truth-compatible, but the transition uses an invalid rule direction or an unsupported dependency.

Table 9: High-level grouping of the error types used in our synthesized data.

F Error Type Definitions and Minimal Examples

This section presents the error taxonomy used by our synthesis framework in a more structured form. We define errors from a *truth-state satisfaction* perspective. A non-structural error either produces an assignment that violates the instantiated rule or current symbolic state, or performs a local update that is not licensed by the active rule. By contrast, a structural error may be locally truth-satisfying, but it violates the rule direction, the dependency order, or the well-formedness of the reasoning trajectory. This distinction matters for PRM supervision: a step may look locally plausible while still being invalid because the prefix does not support it.

We use ASCII operators for readability: `and`, `or`, `xor`, and `->`. The examples below are minimal symbolic cases designed to clarify the boundary between categories.

F.1 Local Truth-State Errors

`drop_condition`

Definition. A necessary condition in a compound premise is ignored. The corrupted step treats a partial match as if the full condition were satisfied.

Valid Given `[F2] -> ([F6] and [F7])`, `[F6]=False`, and `[F7]=True`, the correct update is `[F2]=False`.

Corrupted Given `[F2] -> ([F6] and [F7])`, `[F6]=False`, and `[F7]=True`, the corrupted step ignores `[F6]=False` and outputs `[F2]=True`.

`implication_misuse`

Definition. The corrupted step violates the truth condition of implication. Under `A -> B`, the assignment `A=True, B=False` is invalid.

Valid Given `[F0] -> [F5]` and `[F5]=False`, the correct update is `[F0]=False`.

Corrupted Given `[F0] -> [F5]` and `[F5]=False`, the corrupted step outputs `[F0]=True`, which violates the implication constraint.

`or_and_confusion`

Definition. The step confuses the truth conditions of `and` and `or`.

Valid Given `[F7] -> ([F8] or [F5])`, `[F7]=True`, `[F8]=True`, and an existing state `[F5]=False`, the correct chain does not force `[F5]` to become true.

Corrupted Given the same rule and state, the corrupted chain treats the disjunction as if both branches must hold and incorrectly sets `[F5]=True`.

`partial_evaluation`

Definition. Only part of a compound expression is evaluated, and this partial local observation is incorrectly treated as sufficient to determine another component.

Valid Given `[F6] -> ([F7] and [F3])`, `[F6]=True`, and `[F7]=True`, the correct update is `[F3]=True`.

Corrupted Given `[F6] -> ([F7] and [F3])`, `[F6]=True`, and `[F7]=True`, the corrupted step reasons only from one local component and outputs `[F3]=False`.

`xor_as_or`

Definition. XOR is treated as ordinary inclusive OR. The corrupted step incorrectly allows both sides to be true.

Valid Given `[F12] xor [F10]` and `[F12]=True`, the correct update is `[F10]=False`.

Corrupted Given `[F12] xor [F10]` and `[F12]=True`, the corrupted step outputs `[F10]=True`, incorrectly allowing both sides to hold.

`xor_as_equiv`

Definition. XOR is treated as equivalence. The corrupted step incorrectly forces both sides to have the same truth value.

Valid Given `[F3] xor [F0]` and `[F3]=False`, the correct update is `[F0]=True`.

Corrupted Given `[F3] xor [F0]` and `[F3]=False`, the corrupted step outputs `[F0]=False`, incorrectly making the two sides equivalent.

`vacuous_truth_error`

Definition. The step updates a consequent under a false antecedent. When the antecedent of `A -> B` is false, the implication is already satisfied and does not license changing the existing value of `B`.

Valid Given `[F3] -> [F4]`, `[F3]=False`, and an existing state `[F4]=True`, the correct update is to keep `[F4]=True`.

Corrupted Given `[F3] -> [F4]`, `[F3]=False`, and an existing state `[F4]=True`, the corrupted step overwrites the state and outputs `[F4]=False`.

F.2 Trajectory-Structure Errors

`converse_error`

Definition. The step applies the converse direction of an implication even though the reversed rule is not present in the rule set.

Valid With the available rule `[F4] -> [F3]`, a valid use is to infer `[F3]=True` from `[F4]=True`.

Corrupted With only the rule `[F4] -> [F3]` available,

the corrupted step uses the unavailable converse direction and derives `[F4]=True` from `[F3]=True`.

`redundant_step`

Definition. A locally valid step is repeated after its conclusion has already been established in the prefix. Although the inference itself may be truth-compatible, it is invalid as a trajectory-structure error because it contributes no new supported progress.

Valid A useful step derives `[F3]=True` from `([F3] xor [F5]) -> [F6]`, `[F5]=True`, and `[F6]=False` when that conclusion is needed later in the chain.

Corrupted The corrupted step repeats the same derivation of `[F3]=True` after that fact has already been established and used.

`missing_prerequisite`

Definition. A downstream rule is applied before a required bridge fact has been established in the prefix.

Valid First derive `[F5]=True` from `[F7] -> ([F8] or [F5])`; then use `([F3] xor [F5]) -> [F6]`, `[F5]=True`, and `[F6]=False` to derive `[F3]=True`.

Corrupted The corrupted step applies `([F3] xor [F5]) -> [F6]` directly and derives `[F3]=True` before `[F5]=True` has been established.

`circular_reference`

Definition. Two or more steps mutually support each other instead of forming an acyclic forward chain.

Valid A valid forward derivation establishes each supporting fact before it is used in a later step, producing an acyclic dependency order.

Corrupted The corrupted pair forms a cycle: Step A derives `[F6]=True` using `[F1]=False`, while Step B derives `[F1]=False` using `[F6]=True`.

G Example Data Instances

The following examples illustrate how a synthesized instance stores both the correct and erroneous trajectories. We keep the step-level structure explicit because this is the supervision format used by the PRM: each step contains an applied rule, supporting facts, a conclusion, and, for the erroneous chain, the first invalid transition. The examples also show how the erroneous chain remains aligned with the correct chain while allowing a controlled deviation at the first-error step.

G.1 Example A: Monkey Buns

Monkey Buns grows up in a coastal town in a family bakery and becomes a cheerful local figure known for playful animal-shaped pastries. The target assignment is `[F0]=False`, rendered as "Monkey Buns never took over the family bakery." The erroneous trajectory contains a `missing_prerequisite` error: it applies a downstream rule before establishing the bridge fact `[F5]=True`.

Example A metadata

Name	Monkey Buns
Goal	"Monkey Buns never took over the family bakery."
Target assignment	`[F0]=False`
First error type	`missing_prerequisite`

Correct trajectory: Monkey Buns

Step 1. RULE `[F9] xor [F12]`.
FACTS `[F12]=False`; "The idea of a dessert truck never crossed his mind."
CONCLUSION `[F9]=True`; "Monkey Buns personally delivers his pastries to customers."

Step 2. RULE `[F11] xor [F10]`.
FACTS `[F11]=True`; "Monkey Buns often shows up at children's parties with his pastries."
CONCLUSION `[F10]=False`; "He never dresses up as a monkey."

Step 3. RULE `([F9] and [F8]) -> [F10]`.
FACTS `[F9]=True`; `[F10]=False`.
CONCLUSION `[F8]=False`; "He's content to stay within the coastal town."

Step 4. RULE `[F7] -> ([F8] or [F5])`.
FACTS `[F7]=True`; "The townsfolk adore Monkey Buns as a local treasure"; `[F8]=False`.
CONCLUSION `[F5]=True`; "He whips up pastries shaped like playful animals."

Step 5. RULE `([F3] xor [F5]) -> [F6]`.
FACTS `[F5]=True`; `[F6]=False`; "He often seems gloomy and downcast."
CONCLUSION `[F3]=True`; "Baking brings him pure joy."

Step 6. RULE `([F3] or [F4]) -> [F1]`.
FACTS `[F3]=True`; `[F4]=False`; "He struggles to keep people entertained."
CONCLUSION `[F1]=True`; "Monkey Buns spent his childhood in a small coastal town."

Step 7. RULE `([F0] and [F1]) -> [F2]`.
FACTS `[F1]=True`; `[F2]=False`; "No one in his family ever ran a bakery."
CONCLUSION `[F0]=False`; "Monkey Buns never took over the family bakery."

Erroneous trajectory: Monkey Buns

Step 1. RULE `[F9] xor [F12]`.
FACTS `[F12]=False`; "The idea of a dessert truck never crossed his mind."
CONCLUSION `[F9]=True`; "Monkey Buns per-

sonally delivers his pastries to customers."

Step 2. RULE `[F11] xor [F10]`.

FACTS `[F11]=True`; "Monkey Buns often shows up at children's parties with his pastries."

CONCLUSION `[F10]=False`; "He never dresses up as a monkey."

Step 3. RULE `([F9] and [F8]) -> [F10]`.

FACTS `[F9]=True`; `[F10]=False`.

CONCLUSION `[F8]=False`; "He's content to stay within the coastal town."

Step 4. RULE `([F3] xor [F5]) -> [F6]`.

FACTS `[F6]=False`; "He often seems gloomy and downcast."

CONCLUSION `[F3]=True`; "Baking brings him pure joy."

First error `missing_prerequisite`. The required bridge fact `[F5]=True`, normally derived in the correct Step 4, has not been established.

Step 5. RULE `([F3] or [F4]) -> [F1]`.

FACTS `[F3]=True`; `[F4]=False`.

CONCLUSION `[F1]=True`; "Monkey Buns spent his childhood in a small coastal town."

Step 6. RULE `([F0] and [F1]) -> [F2]`.

FACTS `[F1]=True`; `[F2]=False`.

CONCLUSION `[F0]=False`; "Monkey Buns never took over the family bakery."

G.2 Example B: London

London grows up in a foggy coastal town and later becomes a marine acoustician who studies whale songs and glacier sounds at sea. The target goal is "London learned to navigate by sound long before they could see the path" with symbolic value `[F1]=True`. The first error is `xor_as_equiv`, where the final XOR relation is incorrectly treated as equivalence.

Example B metadata

Name	London
Goal	"London learned to navigate by sound long before they could see the path."
Target value	`[F1]=True`
First error type	`xor_as_equiv`

Correct trajectory: London

Step 1. RULE `[F10] -> ([F11] and [F8])`.

FACTS `[F10]=True`; "London trusts their memory more than their eyes in the thick fog."

CONCLUSION `[F8]=True`; "London has the patience to wait for the ocean to break its long silences."

Step 2. RULE `[F4] xor [F9]`.

FACTS `[F9]=False`; "The soundscape of London's childhood has faded from memory."

CONCLUSION `[F4]=True`; "Part of London's work involves mapping the intricate songs of whales."

Step 3. RULE `[F6] xor [F8]`.

FACTS `[F8]=True`.

CONCLUSION `[F6]=False`; "London always waited for the sun to burn away the mist before venturing out."

Step 4. RULE `([F2] xor [F6]) -> [F7]`.

FACTS `[F6]=False`; `[F7]=False`; "The vibrations of the ocean remain imperceptible to London."

CONCLUSION `[F2]=False`; "London has a career that keeps them firmly on dry land."

Step 5. RULE `([F4] and [F3]) -> [F5]`.

FACTS `[F4]=True`; `[F5]=False`; "Fog was a rare and unfamiliar visitor in London's childhood."

CONCLUSION `[F3]=False`; "London's work rarely requires overnight trips away from shore."

Step 6. RULE `([F2] or [F0]) -> [F3]`.

FACTS `[F2]=False`; `[F3]=False`.

CONCLUSION `[F0]=False`; "London's childhood was spent far from any coastline."

Step 7. RULE `[F0] xor [F1]`.

FACTS `[F0]=False`.

CONCLUSION `[F1]=True`; "London learned to navigate by sound long before they could see the path."

Erroneous trajectory: London

Step 1. RULE `[F10] -> ([F11] and [F8])`.

FACTS `[F10]=True`; "London trusts their memory more than their eyes in the thick fog."

CONCLUSION `[F8]=True`; "London has the patience to wait for the ocean to break its long silences."

Step 2. RULE `[F4] xor [F9]`.

FACTS `[F9]=False`; "The soundscape of London's childhood has faded from memory."

CONCLUSION `[F4]=True`; "Part of London's work involves mapping the intricate songs of whales."

Step 3. RULE `[F6] xor [F8]`.

FACTS `[F8]=True`.

CONCLUSION `[F6]=False`; "London always waited for the sun to burn away the mist before venturing out."

Step 4. RULE `([F2] xor [F6]) -> [F7]`.

FACTS `[F6]=False`; `[F7]=False`; "The vibrations of the ocean remain imperceptible to London."

CONCLUSION `[F2]=False`; "London has a career that keeps them firmly on dry land."

Step 5. RULE `([F4] and [F3]) -> [F5]`.

FACTS `[F4]=True`; `[F5]=False`; "Fog was a rare and unfamiliar visitor in London's childhood."

CONCLUSION `[F3]=False`; "London's work rarely requires overnight trips away from shore."

Step 6. RULE `([F2] or [F0]) -> [F3]`.

FACTS `[F2]=False`; `[F3]=False`.

CONCLUSION `[F0]=False`; "London's childhood was spent far from any coastline."

Step 7. RULE `[F0] xor [F1]`.

FACTS `[F0]=False`.

CONCLUSION `[F1]=False`; "London did not learn to navigate by sound before they could see the path."

First error `xor_as_equiv`. This step treats XOR as equivalence, assigning the other branch the same truth value as `[F0]` rather than the correct opposite value.

H Prompt Templates

This section summarizes the prompts used in PRM training, Best-of-8 reranking, and natural-language translation. We present cleaned versions of the implementation prompts in a compact dialogue format. Placeholders such as `{goal}`, `{question}`, and `{current_step}` are filled automatically by the pipeline. Control fields such as step status and error type are used only during offline data synthesis and annotation; they are not included as label-leaking text in the PRM input. The prompts are grouped by function rather than by source file for readability.

H.1 Step-Level PRM Prompts

Training prompt. During fine-tuning, each symbolic trajectory is expanded into step-level instances. The model receives the target goal, initial information, previous steps, and the current step, and is trained to generate one of two label words. In the main experiments, the label words are `true` and `false`.

PRM training prompt

User Task: Determine whether the current reasoning step is logically correct.
Goal: `{goal}`
Initial Information: `{initial_information}`
Previous Steps: `{previous_steps}`
Current Step: `{current_step}`
Answer:

Assistant `true` or `false`

Each step is serialized with its supporting facts, the applied rule, and the resulting conclusion. An auxiliary reasoning field can also be appended, though it is not required in the main experiments. The same dialogue format is used for both correct and erroneous chains; the label is determined by whether the current step is valid under its prefix.

Best-of-8 scoring prompt. At evaluation time, each sampled candidate solution is split into steps. For the t-th step, the prompt contains the original question, the candidate prefix before that step, and the current step. The PRM scores the probability of the positive label word, and the final trajectory score is obtained by aggregating step-level scores with the `min` rule.

Best-of-8 step scoring prompt

User Task: Determine whether the current reasoning step is logically correct.
Question: `{question}`
Previous Steps: `{previous_steps}`
Current Step: `{current_step}`
Answer:

Assistant `true` or `false`

This prompt differs from the training prompt mainly in its context field: it uses the original problem question instead of the translated symbolic goal. The previous steps are the candidate's own generated prefix, so the judge evaluates whether the current step remains valid in that candidate-specific context.

H.2 Natural-Language Translation Prompts

The translation pipeline converts symbolic PRM instances into aligned natural-language data. All stages share the same entity name, background story, and predicate mapping for the correct and er-

roneous chains. This shared semantic space keeps paired trajectories comparable: their differences should reflect reasoning discrepancies rather than unrelated wording changes.

Background generation. The first stage creates a short scenario for the sampled entity. This scenario provides a coherent semantic context for all predicates in the same instance.

Background generation

System Generate a brief background story for a person. The story should be natural, concise, and provide context for logical reasoning about the person's attributes and actions. Output only the story.

User Generate a background story for someone named `{name}`.

Assistant `{background_story}`

Predicate mapping. The second stage maps fact symbols to natural-language predicates. For each symbol, the model must produce a predicate name, a positive realization, and a negative realization. This allows the translator to render truth values consistently across the goal, base facts, rules, and both chains.

Predicate mapping prompt

System Convert logical fact symbols such as `[F0]` and `[F1]` into vivid, natural English predicates that fit the character's background. Return valid JSON with one entry per fact symbol. Each entry should include a predicate name, a natural sentence for the true value, and a natural sentence for the false value.

Requirements: each predicate must be unique; sentence structures should be varied; negative forms should sound natural rather than mechanical; the background story should guide the predicates; output only valid JSON, with no markdown or extra text.

User **Background:** `{background}`
Name: `{name}`
Fact symbols: `{fact_symbols}`
Create unique, natural-sounding predicates for each fact symbol. Output only JSON.

Assistant `{predicate_mapping_json}`

Expression and rule translation. Simple facts are translated by selecting the corresponding positive or negative realization from the predicate mapping. Compound expressions and symbolic rules are translated with separate prompts conditioned on the predicate mapping. The rule prompt encourages varied realizations of implication, conjunction, disjunction, and exclusive-or relations, rather than repeatedly using rigid "if ... then ..." templates.

Expression and rule translation prompts

System: expression translation Translate the logical expression and its truth value into natural English. Use the provided predicate mapping. Output only the natural-language sentence.

User: expression translation **Expression:** `{expression}`
Truth value: `{truth_value}`
Predicate mapping: `{predicate_mapping}`
Name: `{name}`
Translate the expression into natural English.

System: rule translation Translate the logical rule into natural, conversational English. Vary the sentence structure.
For implication, use patterns such as "When X, Y follows" or "X means Y." For disjunction, express that at least one option holds. For conjunction, express that both statements hold. For exclusive-or, express that exactly one side is true and the two sides cannot both be true.
Treat `[[F]]` the same as `[F]`. Output only the translated rule.

User: rule translation **Rule:** `{rule}`
Predicate mapping: `{predicate_mapping}`
Name: `{name}`
Translate the rule into natural, conversational English.

Assistant `{expression_nl}` or `{rule_nl}`

Step-level translation and refinement. The final stage translates each symbolic step into a natural-language reasoning step. For correct steps, the text states why the conclusion follows from the facts and rule. For erroneous steps, the translator may produce an explanatory annotation of the process problem, but the text used as PRM input is further

naturalized to avoid revealing the label. Structural errors are handled by focusing on missing prerequisites, invalid rule direction, redundant step reuse, or circular dependencies rather than simply treating the conclusion as false.

This prompt is used only during offline natural-language synthesis and annotation. Control fields such as step status and error type guide the translator, but they are not included in the final PRM input. For error-chain steps from the first invalid step onward, the naturalized text avoids explicit markers such as "error", "mistake", "wrong", "invalid", and similar meta-commentary. This preserves the intended propositions while keeping the generated process usable for PRM training.

Step-level translation and refinement prompt

System Translate the reasoning step into concise, natural English.
Keep the statement faithful to the given facts, rule, conclusion, and step status. Output only the step text.

User **Background:** `{background}`
Name: `{name}`
Facts: `{facts_nl}`
Rule: `{rule_nl}`
Conclusion: `{conclusion_nl}`
Step status: `{correct_or_erroneous}`
Error type: `{error_type}`
Write one or two natural sentences for this reasoning step. Preserve all truth values and propositions.
If the step is correct, briefly state why the conclusion fits the facts and rule.
If the output is used as an annotation for an erroneous step, describe the process problem clearly.
If the output is used as PRM input, naturalize the wording and avoid explicit label-leaking words such as "error", "mistake", "wrong", "invalid", "unsupported", "evidence", "established", "assumes", "depends", "relies", "repeats", or "restates".
For structural errors, focus on the missing prerequisite, rule-direction, redundant-step, or cyclic-dependency issue rather than merely saying that the conclusion is false.
Avoid meta phrases such as "the rule says", "according to the rule", "this step", or "the conclusion".
Keep the wording concise and conversational.

Assistant `{step_reasoning_nl}`

The same step-level translation procedure is applied to both correct and erroneous chains under the shared semantic mapping. The resulting paired trajectories remain aligned at the level of facts, rules, and conclusions, while their differences reflect the intended reasoning discrepancy.