CHALLENGES IN INFERENCE-TIME SCALING WITH UNCERTAINTY-AWARE TREE SEARCH

Jacopo Minniti *
Department of Computational Sciences
Minerva University

Neil Band
Department of Computer Science
Stanford University

Tim G. J. Rudner
Department of Statistical Sciences
University of Toronto

ABSTRACT

Inference-time search has emerged as a powerful paradigm for scaling large language models' reasoning capabilities. Standard approaches like beam search rely on process reward models (PRMs) for dense, step-by-step scoring to identify promising reasoning paths. However, scaling these methods encounters a known failure mode: as compute budgets increase, search algorithms explore out-of-distribution states spuriously assigned high reward, decoupling proxy reward from actual reasoning ability. To address this, we propose Uncertainty-Aware Tree Search (UATS), which uses a process uncertainty model (PUM) to predict when PRM predictions are unreliable. UATS dynamically allocates computational resources by increasing the branching factor at high-uncertainty nodes to resolve ambiguity through exploration, unlike the fixed branching of standard beam search. In our evaluation, while PUMs perform well on held-out in-distribution data, this does not translate to improved downstream search. On instruction-tuned models, UATS matches standard beam search, but on RLVR-trained models, it consistently degrades performance as inference-time compute grows. This negative result suggests that search-induced distribution shift causing poor PRM generalization similarly affects process uncertainty models. Our results suggest that uncertainty-guided inference-time scaling requires robust uncertainty quantification models that remain reliable under search-induced distribution shift.

1 INTRODUCTION

Inference-time search has emerged as a promising scaling paradigm for LLMs, using longer and parallel chain-of-thought (CoT) trajectories to enable substantial gains on reasoning-intensive tasks. It is natural to ask whether more powerful search algorithms—such as value-guided tree search underpinning AlphaZero (Silver et al., 2018)—could further improve LLM reasoning. Process Reward Models (PRMs) provide an appealing mechanism by offering dense, step-level reward estimates to prioritize promising trajectories (Luo et al., 2024).

However, PRM-guided algorithms like beam search face a fundamental limitation: as search budgets increase, the policy explores rollouts out-of-distribution for the PRM, decoupling reward estimates from final answer correctness (Khalaf et al., 2025; Gao et al., 2022). We draw on prior work on robustness to distribution shift, where deep uncertainty quantification trains models to explicitly estimate predictive uncertainty for incorporation into downstream algorithms (He et al., 2023).

We hypothesize that equipping a PRM with reliable uncertainty estimates could mitigate this by identifying reasoning paths where reward predictions are unreliable. To operationalize this, we train a Process Uncertainty Model (PUM) to flag when the reward model's assessment may be brittle, predicting step-wise uncertainty analogous to a PRM. We leverage these estimates within Uncertainty-Aware Tree Search (UATS). While retaining the PRM for candidate ranking, UATS uses the PUM to

*Correspondence to: jacopo@uni.minerva.edu

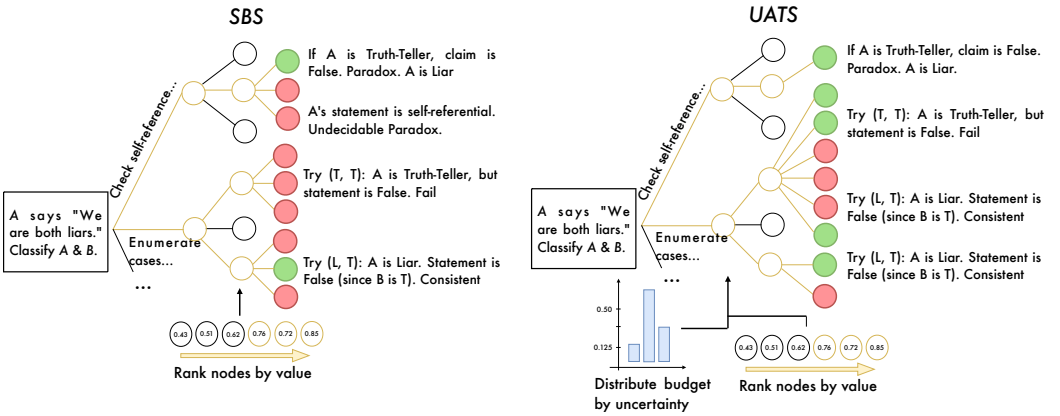

Figure 1: Schematic of Step-Level Beam Search (SBS, **Left**) vs. Uncertainty-Aware Tree Search (UATS, **Right**). SBS ranks candidate paths based only on reward estimates, often overcommitting to out-of-distribution paths spuriously assigned high reward. UATS produces both reward and uncertainty estimates, indicating possible distribution shifts and dynamically scaling the branching factor. By allocating higher budgets to resolve ambiguity in high-uncertainty nodes, UATS explores selectively to mitigate reward model overoptimization.

dynamically allocate search compute: unlike standard beam search's fixed branching factor, our approach directs additional budget toward high-uncertainty nodes. By prioritizing exploration where the PRM is ambiguous, the algorithm aims to resolve uncertainty before committing to a reasoning path, rather than over-exploiting paths to which the PRM assigns spuriously high reward.

Contrary to intuition that uncertainty quantification should robustify search, UATS fails to surpass standard beam search on instruct models and degrades performance on RLVR-trained models. We hypothesize this failure arises because the PUM itself does not generalize under search-induced distribution shift. This issue is amplified in RLVR policies, which exhibit lower answer entropy (Cui et al., 2025), leading to more similar branching trajectories and reduced signal-to-noise ratio that exacerbates spurious correlations.

2 Background

Inference-Time Scaling. Recent work shows LLM performance can be improved by increasing inference-time compute, primarily via extended CoT reasoning where models generate long intermediate traces to decompose complex problems (Wei et al., 2022). To scale efficiently, tree search methods use a reward model to maintain a beam of candidate reasoning paths while pruning dead ends (Chen et al., 2024).

Process Reward Models (PRMs). PRMs are discriminative reward models $\hat{V}(x, y_{1:t}) \in [0, 1]$ trained to map an input prompt x and partial reasoning trace $y_{1:t}$ to likelihood of answer correctness (Lightman et al., 2023). Training targets derive from N Monte Carlo rollouts from the current state: a step receives positive label (1) if any continuation leads to correct final answer, negative (0) otherwise (Zhang et al., 2025). We train PRMs using binary cross-entropy (BCE) objective.

3 Experimental Design

We investigate whether uncertainty estimates can mitigate reward model overoptimization in inference-time search, describing our beam search baseline followed by trained models and intervention algorithms.

Step-Level Beam Search (SBS). SBS generates a tree of reasoning paths incrementally by maintaining B_1 partial chains (beams). At each step t, the algorithm performs *expansion* by drawing multiple candidate next steps for each partial chain (Chen et al., 2024). After expansion, B_2 candidate chains are *scored and pruned* by the PRM; only top B_1 steps are retained.

Models and Datasets. We evaluate two model classes: `Qwen2.5-1.5B-Instruct` (Yang et al., 2025) on MATH (Hendrycks et al., 2021) as instruction-tuned baseline, and `DeepSeek-R1-Distill-Qwen-1.5B` (Guo et al., 2025) as long CoT reasoning model on MMLU-Pro subset ($N = 500$) (Wang et al., 2024), following the line of reasoning de-

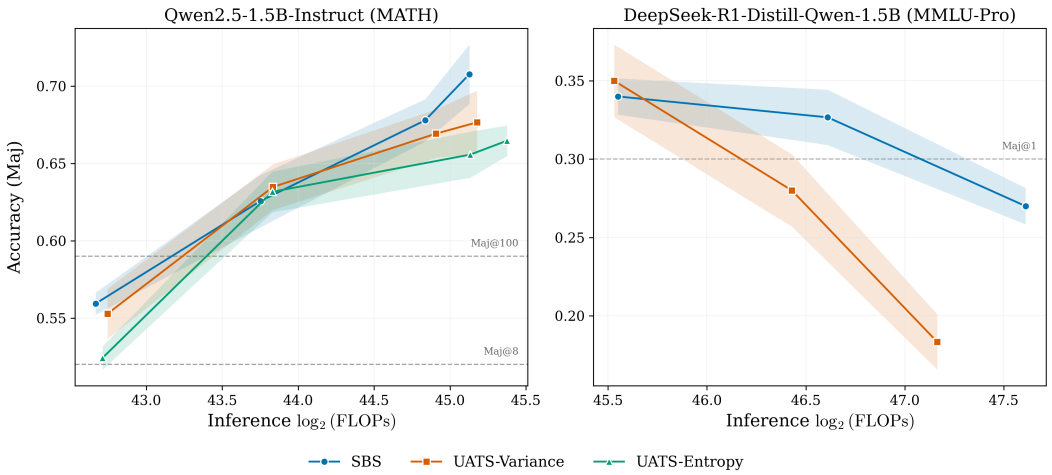

Figure 2: Performance-FLOPs Pareto Frontier for Inference-Time Search. **Left:** On MATH (Qwen2.5), UATS performs on par with SBS. **Right:** On MMLU-Pro (DeepSeek-R1-Distill), UATS degrades performance as compute increases. Shaded bands denote 95% confidence intervals over 5 random seeds (see Appendix B.2).

scribed in Appendix A.1. We initialize PRMs from corresponding policy parameters and use `Qwen2.5-Math-PRM-7B` as strong PRM baseline on MATH (Zhang et al., 2025).

Process Uncertainty Model Training Data. To train the PUM, we derive step-level uncertainty supervision from Monte Carlo rollouts conditioned on partial reasoning trajectories. Given an intermediate state, we sample $N = 16$ rollouts and compute uncertainty from empirical distribution of final answers. We consider two complementary signals: (i) answer entropy, capturing dispersion across distinct final answers, and (ii) variance of correctness, capturing variability in answer correctness across rollouts. Full details in Appendix A.2.

Training Process Uncertainty Models. To avoid expensive rollouts at inference, we distill uncertainty statistics into a PUM. The PUM takes partial reasoning paths $(x, y_{1:t})$ as input and trains analogously to a PRM except using entropy or variance labels. We use mean squared error to regress continuous uncertainty targets (Appendix A.3). At inference, predictions $\hat{U}(x, y_{1:t})$ dynamically allocate compute to partial states with higher exploration potential. We string-normalize answers to remove superficial variability (see Appendix C.1).

Uncertainty-Aware Tree Search (UATS). Unlike SBS's fixed expansions $k = \lfloor B_2/B_1 \rfloor$, UATS retains the PRM for ranking but allocates expansion budget unevenly based on estimated uncertainty. Given active partial reasoning states $\{x^{(1)}, \ldots, x^{(B_1)}\}$, we compute score $\hat{U}(x^{(i)})$ for each beam. Scores convert to normalized weights $w_i = \mathrm{softmax}(\hat{U}(x^{(i)})/\tau)$ with $\tau = 1.0$ (varying τ yielded no significant differences). Total budget B_2 distributes proportionally, assigning $k_i = \max(1, \lfloor B_2 \cdot w_i \rfloor)$ expansions to beam i. To conserve budget B_2, overallocated expansions randomly subtract from beams with $k_j > 1$. Candidates are subsequently scored and pruned by the PRM (see Appendix B.1 for algo-box).

4 Empirical Evaluation

Upstream Validation. We evaluate PUMs on held-out reasoning traces, calculating metrics pooled across the test set. As in Table 1, models demonstrate strong predictive capabilities. We use AUROC to assess ranking ability by relative uncertainty/reward. We emphasize R^2: most PRMs act as rankers where relative order suffices, whereas UATS uses uncertainty magnitude for resource allocation. High R^2 verifies our regression captures true variance scale. The substantial R^2 drop on MMLU-Pro likely stems from significantly longer CoTs requiring sparser evaluation signals (Appendix D).

Downstream Search Performance. Figure 2 presents Pareto frontiers. On the Base model, SBS and UATS frontiers largely overlap, indicating no marginal utility. We observe a few cases where UATS-Entropy is significantly worse than SBS for high compute (T-test, $p = 0.0042$). On the RLVR-tuned model, UATS consistently degrades performance. Curves diverge around 2^{46} FLOPs, and at approximately 2^{47} FLOPs, UATS-Variance accuracy drops to 18%, creating roughly a 10%

Config	Metric	Official PRM	Our PRM	PUM (Var)	PUM (Ent)
Qwen 2.5 / MATH	R^2	N/A*	0.59	0.40	0.85
	AUROC	0.73	0.92	0.86	0.94
R1-Distil / MMLU-Pro	R^2	N/A*	0.51	0.24	0.30

Table 1: Upstream metrics on static test sets. We omit R^2 for Zhang et al. (2025) PRM as it trains with BCE rather than MSE, making the metric difficult to interpret; BCE setup induces strong class separation but poor reward calibration. Our PUMs align closely with ground-truth uncertainty, especially on MATH.

gap versus SBS. Even standard SBS struggles to deliver gains in this regime, likely due to difficulty training robust small-scale (1.5B) PRMs on longer CoTs.

Distributional Analysis. We evaluate Pearson r between PUM predictions and ground-truth variance from Monte Carlo rollouts, both in-distribution and under distribution shift. On held-out upstream test set, PUM achieves $r = 0.62$, indicating strong in-distribution uncertainty estimation. To assess performance under active search, we run UATS, extract highest-reward path from final beams, and recompute ground-truth variance via $N = 16$ rollouts; in this setup, correlation collapses to $r \approx 0.03$ (see Appendix C.2). This degradation coincides with reduced policy diversity: the Instruct model maintains an average answer entropy of 1.3 nats, whereas the RLVR model drops to 0.77 nats, with nearly 40% of training steps exhibiting zero rollout variance.

5 Discussion & Conclusion

We interpret our negative results through the adversarial nature of search optimization, with entropy collapse in RLVR policies further exacerbating effects. These factors suggest process uncertainty estimation requires greater robustness to distribution shift, potentially through on-policy training.

Search-Induced Distribution Shift. Our results suggest UATS actively exploits weaknesses in the uncertainty model. While PUM performs well on in-distribution validation traces, search procedure consistently selects trajectories where predictions are unreliable, favoring steps confidently incorrect or exhibiting inflated uncertainty without true ambiguity. While variance-based allocation inherently deprioritizes highly confident paths, this does not explicitly prune ultimately correct paths as the reward model preserves them (see Appendix F). This points to recursive generalization failure: uncertainty estimates reliable in isolation break down once used as optimization targets. Rather than mitigating PRM-guided search failures, PUM exhibits similar collapse, with correlation dropping sharply ($r : 0.62 \rightarrow 0.03$). Predicted uncertainty no longer tracks true uncertainty during optimization.

Entropy Collapse in RLVR. This vulnerability amplifies through intrinsic properties of RLVR models. Though RLVR collapses policy entropy (Cui et al., 2025) and PRMs are brittle in these regimes (Khalaf et al., 2025), we hypothesized uncertainty estimation may mitigate poor generalization effects. The instruct model exhibits average answer entropy of $\approx$ 1.3 nats, while RLVR model drops to ≈ 0.77 nats, with nearly 40% of steps showing zero variance across rollouts. In this near-deterministic regime, uncertainty signals become severely impoverished, and UATS introduces stochasticity largely uncorrelated with solution quality, diverting compute toward costly dead ends.

Limitations. Uncertainty quantification in LLMs remains inherently difficult. In these preliminary experiments, we relied on naive proxies like answer variance and entropy. This research direction is compelling but necessitates more performant uncertainty quantification strategies. Furthermore, experiments were constrained to fairly small LLMs; given PRMs are data-hungry, effective PUMs may require significantly more model capacity. Using larger reward models (e.g., 7B) alongside 1.5B policy improves absolute performance but does not alter fundamental degradation trend under UATS (see Appendix E).

Conclusion. We present an instructive negative result on off-policy uncertainty guidance in inference-time search. We find search induces distribution shifts that exploit static uncertainty estimates, echoing failure modes seen in process reward models. While uncertainty remains a plausible regularizer, using it effectively may require closing the optimization loop by iteratively training uncertainty models on the adversarial distribution produced by search itself.

REFERENCES

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: Process supervision without process, 2024. URL https://arxiv.org/abs/2405.03553.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. The entropy mechanism of reinforcement learning for reasoning language models, 2025. URL https://arxiv.org/abs/2505.22617.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization, 2022. URL https://arxiv.org/abs/2210.10760.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Wenchong He, Zhe Jiang, Tingsong Xiao, Zelin Xu, and Yukun Li. A survey on uncertainty quantification methods for deep learning, 2023. URL https://arxiv.org/abs/2302.13425.

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

Hadi Khalaf, Claudio Mayrink Verdun, Alex Oesterling, Himabindu Lakkaraju, and Flavio du Pin Calmon. Inference-time reward hacking in large language models, 2025. URL https://arxiv.org/abs/2506.19248.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

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. Improve mathematical reasoning in language models by automated process supervision, 2024. URL https://arxiv.org/abs/2406.06592.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, 2018.

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. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL https://arxiv.org/abs/2406.01574.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2022. URL https://arxiv.org/abs/2201.11903.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning, 2025. URL https://arxiv.org/abs/2501.07301.

APPENDIX

A PROCESS UNCERTAINTY MODELING

This section details the construction of ground-truth uncertainty signals, the training configuration for the Process Uncertainty Models (PUMs), and the definitions of reasoning steps used across different architectures.

A.1 DATASETS CHOICE

Given our focus on reasoning tasks, we selected two datasets aligned with this objective. Our choice of MATH and MMLU-Pro for evaluating the respective models was guided by several considerations. First, we ensured that the base models were neither too strong nor too weak on the chosen benchmarks: if performance were already saturated, an inference algorithm like SBS would show minimal room for improvement, making differences harder to detect; conversely, if baseline performance were too low, the models would produce largely nonsensical outputs. After filtering reasoning and math datasets that satisfied these criteria, we selected two that are widely used in the community and for which benchmarks are commonly reported, ensuring our results are comparable and meaningful.

A.2 GROUND TRUTH UNCERTAINTY SIGNALS

We construct step-level ground-truth uncertainty signals using Monte Carlo rollouts conditioned on partial reasoning trajectories. Given an input x and a partial reasoning path $(y_{1:t})$, we condition the policy $\pi_\theta(\cdot \mid x, y_{1:t})$ and sample $N = 16$ independent rollouts, each producing a final answer $a^{(i)} \in \mathcal{A}$.

These rollouts induce an empirical distribution over final answers, denoted $\hat{p}(a)$. From this answer distribution, we compute two uncertainty statistics. First, we measure answer entropy,

$$H(\hat{p}) = -\sum_a \hat{p}(a) \log \hat{p}(a),$$

which quantifies dispersion across distinct final answers and reflects uncertainty arising from answer disagreement. Second, we measure uncertainty in correctness via the binomial variance computed over answers. Let

$$\hat{q} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}^{(i)}$$

denote the fraction of rollouts whose final answer is correct, where $\mathbb{I}^{(i)}$ is an indicator of correctness for rollout i. The corresponding variance is

$$\sigma(\hat{q}) = \hat{q}(1 - \hat{q}),$$

which captures variability in answer correctness across sampled rollouts and is maximized when correctness is most uncertain. We use these two statistics as supervision targets for the PUM, yielding entropy-based and variance-based uncertainty labels, denoted U^{ent} and U^{var}, for each partial reasoning trajectory.

A.3 MODEL TRAINING CONFIGURATION

Our Process Uncertainty Models (PUMs) are trained to regress scalar uncertainty values associated with specific reasoning steps. We employ the following configuration:

- **Architecture:** We utilize the `Qwen2.5-1.5B-Instruct` and `DeepSeek-R1-Distill-Qwen-1.5B` architectures. A linear regression head projects the hidden state of a special token `<extra_0>` to a scalar float32 value.
- **Loss:** We employ Mean Squared Error (MSE) loss.
- **Data Format:** Training data consists of reasoning traces where the `<extra_0>` token is appended after every step. Labels are Z-score normalized ($z = \frac{x-\mu}{\sigma}$) within each prompt to encourage relative ranking over absolute value prediction.

We experimented with both $N = 8$ and $N = 16$ rollouts during training. As also noted by Zhang et al. (2025), the diversity of step rollouts begins to decrease significantly after 8 rollouts. Consequently, we observed only minor differences in sample variance and entropy between $N = 8$ and $N = 16$, making $N = 16$ a computationally efficient upper bound.

```
    TrainingArguments(
        # --- PRM Specific ---
        max_length=5000,
        train_on_last_step_only=False,  # Train on every step
        step_separator=step_separator_token,

        # --- Optimizer & Scheduler ---
        learning_rate=1e-4,
        lr_scheduler_type="cosine",
        warmup_ratio=0.15,
        weight_decay=0.1,
        optim="adamw_torch",

        # --- Batching & Memory ---
        per_device_train_batch_size=4,
        per_device_eval_batch_size=4,
        gradient_accumulation_steps=4,
        gradient_checkpointing=False,
        bf16=bf16_supported,
        dataloader_pin_memory=True,
        dataloader_num_workers=4,

        # --- Logging & Saving ---
        logging_steps=5,
        save_strategy="epoch",
        eval_strategy="epoch",
    )
```

Listing 1: PUM Training Arguments

A.4 Reasoning Step Granularity

For the base Qwen2.5 model, we defined a reasoning step as any text block separated by a double newline (`\n\n`). While generally robust, we note that in math contexts, new paragraphs occasionally split single equations; however, these edge cases were rare.

For the DeepSeek-R1-Distill model, the model was post-trained to use `\n\n` as a structural delimiter for reasoning. However, due to the extreme length of these CoT traces, sampling $N = 16$ Monte Carlo rollouts at every single step was computationally infeasible. Therefore, for the MMLU-Pro experiments with R1-Distill, we aggregated uncertainty targets and performed predictions every 3 steps.

B Search Algorithm and Experimental Setup

We provide the pseudocode for the Uncertainty-Aware Tree Search (UATS) algorithm, alongside details regarding the baseline configurations and compute estimation methods used in our experiments.

B.1 UATS ALGORITHM

Algorithm 1 Inference-Time Search: SBS vs. UATS

Require: Input prompt x, Policy π_θ, PRM V_ϕ
Require: **Hyperparams:** Beam width B_1, Total candidate budget B_2, Max depth T
Require: **Optional:** PUM U_ψ (Required only for UATS)

Initialize beams $\mathcal{B}_0 \leftarrow \{x\}$
for $t = 1$ **to** T **do**
  $\mathcal{C} \leftarrow \emptyset$ {Initialize candidate set for current step}
  // Phase 1: Resource Allocation
  if Method is **SBS then**
    $k_{\text{base}} \leftarrow \lfloor B_2 / |\mathcal{B}_{t-1}| \rfloor$ *// Fixed branching factor*
    for each beam $b^{(i)} \in \mathcal{B}_{t-1}$ **do**
      $k_i \leftarrow k_{\text{base}}$
    end for
  else if Method is **UATS then**
    Calculate Uncertainty: $\hat{u}_i \leftarrow U_\psi(b^{(i)}) \quad \forall b^{(i)} \in \mathcal{B}_{t-1}$
    Compute Weights: $w_i \leftarrow \text{Softmax}(\hat{u}_i/\tau)$ *// with temperature* $\tau = 1.0$
    for each beam $b^{(i)} \in \mathcal{B}_{t-1}$ **do**
      $k_i \leftarrow \max(1, \lfloor B_2 \cdot w_i \rfloor)$ *// Allocate proportional budget, min 1*
    end for
    // Adjust k_i *by randomly subtracting from beams with* $k_j > 1$ *to strictly sum to* B_2
  end if

  // Phase 2: Expansion
  for each beam $b^{(i)} \in \mathcal{B}_{t-1}$ **do**
    Sample k_i continuations $\{s_1^{(i)}, \ldots, s_{k_i}^{(i)}\} \sim \pi_\theta(\cdot \mid b^{(i)})$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{b^{(i)} \oplus s_j^{(i)}\}_{j=1}^{k_i}$
  end for

  // Phase 3: Scoring and Pruning
  scores $\leftarrow \{V_\phi(c) \mid c \in \mathcal{C}\}$
  $\mathcal{B}_t \leftarrow \text{SelectTop}(\mathcal{C}, \text{scores}, k = B_1)$ *// Prune to top* B_1 *candidates*
end for
return Best trajectory in $\mathcal{B}_T$

B.2 EXPERIMENTAL BASELINES

For the comparative analysis in Figure 2 (Left Panel), we utilized the 7B PRM from Zhang et al. (2025), while for the RLVR model we trained our own PRM. We note that the 7B PRM is trained with BCE where a positive label is assigned if *any* rollout succeeds. This training objective encourages strong class separation but does not yield a well-calibrated reward estimate. Consequently, when evaluated with R^2, this model often yields very low or even negative scores, indicating predictive performance worse than predicting the mean, despite being an effective ranker.

B.3 COMPUTE COST ESTIMATION

To compare search budgets fairly across different token lengths, we estimate the total inference compute in Floating Point Operations (FLOPs). We assume a fixed cost per token of 2.62×10^9 FLOPs, which corresponds to approximately $2\times$ the active parameter count of the 1.5B models used. The final metric is reported as the base-2 logarithm of the total accumulated FLOPs.

Note that we do not include the FLOPs incurred by the PUM forward passes in these estimates. The PUM operates only on the generated steps (e.g., often fewer than 20 steps for Qwen2.5 on MATH), meaning it requires significantly fewer forward passes. This results in a negligible compute overhead compared to the generative, token-by-token cost of the policy model.

```
import numpy as np

```

```python
def flops(tokens):
    """
    Calculates the log2 FLOPs based on the number of tokens.
    Constant 2.62B corresponds to approx 2 * model_params.
    """
    if tokens <= 0: return 0
    # Estimate: 2.62e9 FLOPs per token for 1.5B model
    flops_val = 2_620_000_000 * tokens
    return round(np.log2(flops_val), 2)
```

Listing 2: Compute Cost Estimation

C Metrics and Analysis

This section details the normalization procedures used for computing answer entropy and provides quantitative analysis of the distribution shift observed during search.

C.1 Answer Normalization

To calculate "Answers Entropy," we utilized specific normalization functions. We chose this over metrics like Semantic Entropy because, in Math and MMLU contexts, having two similar answers is not a guarantee that the reasoning process is semantically similar. For instance, two rollouts might arrive at $x = 2$ and $x = 3$; blindly clustering these could mask genuine reasoning divergence.

MATH Normalization. For the MATH dataset, we implement a robust normalizer to handle LaTeX formatting differences (e.g., handling fractions, whitespace, and boxing).

```python
import re

def normalize_answer_math(answer: str) -> str:
    """
    Normalizes a mathematical answer for robust comparison.
    This version does NOT evaluate mathematical expressions.
    """
    if not answer:
        return ""

    # 1. Initial text cleaning and equation handling
    normalized = str(answer).lower().strip()
    if '=' in normalized:
        normalized = normalized.split('=')[-1].strip()

    # 2. Remove LaTeX delimiters first
    normalized = re.sub(r'^(\$|\\\[|\\\(|\\\$)|(\$|\\\]|\\\)|\$$)$', '',
    normalized).strip()

    # 3. Handle LaTeX commands non-destructively
    normalized = re.sub(r'\\left|\\right', '', normalized)
    normalized = re.sub(r'\\(text|mathrm|mathbf|boldsymbol)\s*\{([^}]*)\}
    ', r'\2', normalized)
    normalized = re.sub(r'\\frac\{([^}]+)\}\{([^}]+)\}', r'(\1)/(\2)',
    normalized)
    normalized = re.sub(r'\\%', '%', normalized)

    # 4. Handle numerical conversions
    if '%' in normalized:
        normalized = re.sub(r'(\d*\.?\d+)\s*%', lambda m: str(float(m.
    group(1)) / 100.0), normalized)
    normalized = re.sub(r',(?=\d)', '', normalized)

    # 5. Standardize spacing
    normalized = re.sub(r'\s+', ' ', normalized).strip()
    normalized = re.sub(r'\s*([+\-*/=()^])\s*', r'\1', normalized)

```

```
    # 6. Final cleanup for numeric answers
    if '.' in normalized:
        normalized = normalized.rstrip('0').rstrip('.')
        if normalized == "":
            normalized = "0"

    return normalized
```

Listing 3: MATH Answer Normalization

MMLU-Pro Normalization. For MMLU-Pro, answer normalization is highly straightforward. Since the model is explicitly instructed to output its final choice within a bounding box (e.g., `\boxed{A}`), we simply extract the final letter using a regular expression.

```
import re

def normalize_answer_mmlu(answer: str) -> str:
    """
    Extracts the final letter choice from MMLU-Pro boxed outputs.
    """
    if not answer:
        return ""
    match = re.search(r'\\boxed\{([A-Z])\}', answer)
    if match:
        return match.group(1)
    return answer.strip()
```

Listing 4: MMLU-Pro Answer Normalization

C.2 Search-Induced Distribution Shift

We quantify the distribution shift by comparing the correlation of the PUM predictions against ground truth labels in two settings: the static test set (MC Rollouts) and the active search beams (SBS).

Setting	**Metric**	**Pearson r**	**Mean Pred vs Gold**
Static (GT)	Qwen2.5 Variance	0.62	$0.319 \approx 0.308$
Search (SBS)	Qwen2.5 Variance	0.03	$0.225 \gg 0.045$

Table 2: The "Bad Goodhart" effect. In search, the correlation collapses to near zero, and the model remains optimistic (high predicted score) while the true reward of the selected nodes collapses.

D Discrepancy in PUM Predictive Accuracy

The discrepancy in predictive accuracy observed in Table 1 between MATH ($R^2 = 0.85$) and MMLU-Pro ($R^2 = 0.30$) is likely attributable to the differing CoT lengths. For MMLU-Pro, the CoTs are considerably longer, necessitating a sparser evaluation signal (predictions and targets computed every 3 `\n\n` steps). This sparsity dilutes the step-level resolution, making precise uncertainty prediction substantially harder than on MATH.

E Model Scale and Diversity

To investigate whether increasing model capacity or diversity mitigates the observed issues, we evaluated a mixed-scale setup using a 1.5B parameter policy model alongside a stronger 7B parameter PRM. While the absolute reasoning performance improved across the board, the general trend of performance degradation under UATS remained unchanged. This suggests that the search-induced distribution shift affects even highly capable reward models, and that simply scaling the verifier does not resolve the underlying issue.

F Effect of Variance on Path Preservation

A potential concern with UATS-Variance is that it explicitly deprioritizes paths with near-zero variance, which includes both confidently incorrect and confidently correct reasoning chains. This could theoretically penalize diverse but initially low-confidence approaches. However, while entirely incorrect paths correctly receive low expansion budgets, confidently correct paths are still strongly preferred by the PRM during the scoring and pruning phase. Consequently, even if a correct path receives only a single expansion (due to low uncertainty), its high reward estimate ensures it is retained in the beam. Therefore, the loss of correct paths is minimal, and the primary failure mode remains the over-allocation of compute to spuriously uncertain states.