Delay, Plateau, or Collapse: Evaluating the Impact of Systematic Verification Error on RLVR

Kazuki Egashira[1], **Mark Vero**[1], **Jasper Dekoninck**[1], **Florian E. Dorner**[1,2], **Robin Staab**[1], **Martin Vechev**[1]
[1] ETH Zurich, [2] Max Planck Institute for Intelligent Systems, Tübingen
kazuki.egashira@inf.ethz.ch

Abstract

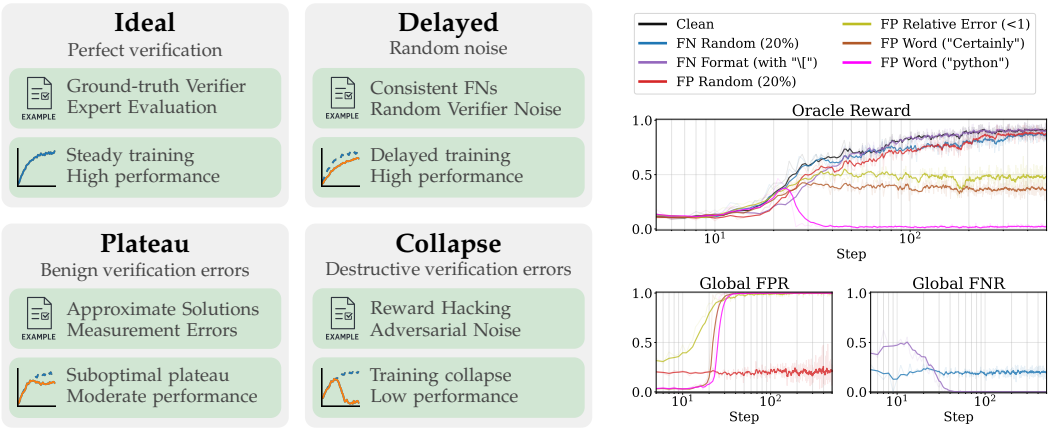
Reinforcement Learning with Verifiable Rewards (RLVR) has become a powerful approach for improving the reasoning capabilities of large language models (LLMs). While RLVR is designed for tasks with verifiable ground-truth answers, real-world verifiers (e.g., static code checkers) can introduce errors into the reward signal. Prior analyses have largely treated such errors as random and independent across samples, concluding that errors merely slow training with limited effect on final performance. However, practical verifiers tend to exhibit systematic errors. This introduces a risk of models learning unwanted consistent behavior from a structurally incorrect reward signal. In this work, we study the impact of such systematic verification errors on RLVR. Through controlled experiments on arithmetic tasks, we show that systematic false negatives lead to similar effects as random noise. On the other hand, systematic false positives can cause a wide range of behaviors from sub-optimal plateaus to performance collapse. Crucially, these outcomes are not determined by the overall error rate but by the specific pattern of introduced errors, making pre-hoc mitigation difficult. Our results show that, in contrast to prior conclusions, realistic verification errors can critically shape RLVR outcomes and that verifier quality has to be understood beyond its sample-level error rate.

1 Introduction

Large language models (LLMs) have advanced rapidly across a wide range of domains and are now able to assist experts with complex tasks such as mathematical reasoning and coding (Yang et al., 2025; Zeng et al., 2025). One of the key techniques behind this progress is Reinforcement Learning with Verifiable Rewards (RLVR) (Shao et al., 2024), which applies reinforcement learning using a ground-truth verifier that rewards the model based on whether its output is correct. However, as LLMs are deployed on increasingly complex tasks, designing accurate verifiers becomes challenging (Gunjal et al., 2025) and the reward signal may become *erroneous*. In particular, verifiers may accept incorrect solutions, producing false positives (FP), or reject correct ones, producing false negatives (FN).

Impact of imperfect verifiers Several works have examined the impact of verification errors on RLVR training (Rad et al., 2026; Cai et al., 2025; Lv et al., 2025). Their findings suggest that such errors merely delay training progress without changing the final outcome. However, these conclusions largely rely on the assumption that verifier errors are random, and existing experiments typically model noise through random label flipping. This assumption may not always hold because many practical verifiers, including static analyzers and LLM-as-a-judge, can exhibit systematic and potentially exploitable error patterns (Huang et al., 2025; Chen et al., 2025; Zhao et al., 2025).

This work: analyzing systematic errors In this work, we study the impact of systematic verification errors in RLVR. To do so, we first provide a rigorous definition of systematic verification errors, clearly differentiating them from random noise. Then, in controlled



(a) Categorization of systematic errors in RLVR. (b) Representative results. (OLMo3-7B)

Figure 1: Overview of our results. On the left, we categorize systematic errors in RLVR based on their effect on training dynamics, including delayed training, suboptimal plateaus, and training collapse. On the right, we show representative results of these different error types in a controlled experiment using the Reasoning Gym dataset (Stojanovski et al., 2025).

experiments on the arithmetic task from the Reasoning Gym library (Stojanovski et al., 2025), we study how different types of systematic errors affect RLVR training. In particular, we introduce simple, controllable trigger patterns into a ground-truth verifier, resulting in systematic false positives or false negatives. This setup allows us to precisely track all key aspects of the training dynamics, including the error rate, specific error patterns, and the model's behavior.

Key findings: delay, plateau, or collapse As shown in Figure 1, we find that systematic errors can lead to a range of outcomes, including delayed training, suboptimal plateaus, and even complete collapse. Importantly, we find that these behaviors cannot be explained solely by simple sample-level metrics such as false-positive rate (FPR) or false-negative rate (FNR) at the start of training. As shown in Figure 1b, total collapse is possible even if the FPR is very low initially, making the impact of verification errors difficult to predict *a priori*.

Implications for RLVR RLVR is increasingly being applied to complex tasks, where there is an inherent tradeoff between verification cost and accuracy. For example, using a state-of-the-art LLM as a verifier may be prohibitively expensive. This can make smaller models or static analyzers attractive alternatives, even though they may exhibit systematic error patterns. Our findings suggest that the choice of verifier can critically shape RLVR outcomes, and that understanding a verifier's specific error patterns is essential for anticipating their effects. Importantly, we also find that averaged *a priori* error rates are insufficient to predict whether a given verifier is reliable enough to attain reasonable performance. For instance, as we show in §4.4, collapse can arise under error patterns that are asymmetric around the ground truth, even when their symmetric counterpart, despite having a strictly higher error rate, only leads to a plateau. This makes it crucial to analyze and improve verifier quality in ways that go beyond aggregate error rates.

Key contributions:

- We analyze *systematic* verification errors in RLVR [1] and categorize them according to their effect on training dynamics (§3).
- We design controlled experiments to analyze the impact of noisy verifiers, finding delayed training, suboptimal plateaus, and even training collapse (§4.2).
- We analyze, discuss, and provide a practical outlook for the impact of verification errors (§4.3–§4.5). Our results indicate that an initial error rate of the verifier does not fully explain the training outcome.

[1] Code available at: `https://github.com/eth-sri/llm-verifier-noise`

2 Related Work

In this section, we briefly review the relevant literature on RLVR and on the impact of verification errors during RLVR training.

RLVR RLVR is a widely used post-training technique for improving the reasoning capabilities of LLMs (Guo et al., 2025; Shao et al., 2024; Team et al., 2025; Qwen, 2025; Zeng et al., 2025). It has been applied extensively in domains where verification is relatively simple, including mathematics based on final answers and formal proofs (Lin et al., 2025; Guo et al., 2025) and code generation (Liu et al., 2023a). More recently, it has also been extended to broader settings, with rewards based on rubrics (Gunjal et al., 2025) or semantic similarity in code (Wei et al., 2025). RLVR is often implemented using Group Relative Policy Optimization (GRPO) (Shao et al., 2024) or related variants (Yu et al., 2025; Liu et al., 2025; Gao et al., 2025). In these methods, the model generates multiple rollouts for each problem, and a *verifier* assigns a scalar *reward* to each rollout. The rewards are then compared within each problem-specific group to compute an *advantage*, which serves as the learning signal.

Verification errors In practice, RLVR rewards are often erroneous because verifiers accept incorrect solutions or reject correct ones. For example, unit tests rarely provide complete coverage (Liu et al., 2023b), and rule-based verifiers can reject correct answers because of formatting issues (Xu et al., 2025). LLM-based verifiers are also increasingly used, but they often exhibit high false-positive rates (Raina et al., 2024; Zhou et al., 2025; Zhao et al., 2025). This has motivated growing interest in measuring verifier quality. In particular, Li et al. (2026); Yan et al. (2025) propose benchmarks for evaluating verifier accuracy across domains.

The impact of random noise on RLVR Several works directly analyze the effect of random verification errors on RLVR. For example, Lv et al. (2025) argue that RLVR tolerates up to 40% randomly flipped labels with little performance loss. In contrast, Cai et al. (2025) report a stronger negative effect, although they find that the lost performance can be recovered with an algorithmic correction that is effectively a change in learning rate. Rad et al. (2026) complement these empirical results with a theoretical analysis based on Youden's $J := \mathrm{TPR} - \mathrm{FPR}$: When $J < 0$, training collapses, whereas perfect learning is possible when $J > 0$. Overall, these works show that aggregate noise rates can explain RLVR behavior when verification errors are modeled as random label flips. Our work instead studies *systematic* errors that the model may learn to trigger or avoid during training.

The impact of systematic errors on RLVR A small body of work moves beyond random-noise assumptions and shows that imperfect verification can harm RLVR in specific settings. For instance, Huang et al. (2025) study both rule-based and LLM-based verifiers for final-answer mathematical problems. They find that LLM-based verifiers achieve higher accuracy on a fixed dataset, but are also more susceptible to reward hacking, leading to worse post-RLVR performance. Similarly, Chen et al. (2025) show that introducing verification errors in code RL harms performance, and Yan et al. (2025) find that verifier accuracy is correlated with RLVR performance on math for LLM-based verifiers. Lastly, Zhu and Kang (2026) find that while randomizing ground-truth answers causes RLVR performance to fully collapse, replacing ground truth with wrong samples from another LLM leads to diminished performance gains. These works show that verifier failures can harm RLVR in specific settings, but they do not isolate how different systematic error patterns induce delayed learning, plateaus, or collapse under controlled conditions.

Reward hacking Failures in reinforcement learning due to systematic errors, especially false positives, can be seen as a special case of reward hacking. Reward hacking is a well-known phenomenon, where agents exploit loopholes in the reward function to obtain high rewards without solving the intended task (Hadfield-Menell et al., 2017; Skalse et al., 2022). In particular, Gao et al. (2023) derive scaling laws for reward hacking in RLHF with continuous reward models, while Dorner et al. (2025) characterize the impact of verification errors on resampling-based test-time scaling. However, no prior work has studied the connection between systematic verification errors and RLVR, which is the focus of our work.

3 Characterizing Training Dynamics with Imperfect Verifiers

In this section, we define systematic errors in the context of RLVR and characterize the downstream training dynamics that it can induce. We first introduce the necessary notation and definitions, and then describe the different training dynamics.

Rewards and verifiers A verifier is a function V that takes a query x and a model output y and produces a reward signal $r = V(x, y)$. In this work, we restrict ourselves to binary rewards $r \in \{0, 1\}$. With V^*, we denote the hypothetical ground-truth verifier that produces the correct reward signal r^* for any input-output pair. However, in training V^* is often too expensive to use or technically infeasible and therefore replaced by a cheaper but imperfect verifier V. In RLVR, a model M is then trained to maximize the expected reward obtained using the verifier. In practice, this objective is optimized with approximate reinforcement learning algorithms such as GRPO (Shao et al., 2024). These algorithms typically involve sampling multiple rollouts for each problem and train the model using advantages, computed as $A_i = (r_i - \mu(r))/\sigma(r)$, where r_i is the reward assigned to a rollout with average reward $\mu(r)$ and standard deviation $\sigma(r)$ computed across rollouts for the same query.

Performance metrics During training, we track several metrics to analyze training dynamics. Our main metric is the *oracle reward* $R^*(M)$, defined as the average reward assigned by the ground-truth verifier V^* to the model's outputs. This metric reflects the model's true task performance, independent of verification errors. We also track the *verifier reward* $R(M)$, defined as the average reward assigned by the imperfect verifier V to the model's outputs. This is the reward signal the model actually optimizes during training. Finally, we track the false positive rate (FPR) and false negative rate (FNR) of the verifier V relative to the ground-truth verifier V^*. Importantly, as most of these essential metrics require access to the ground-truth verifier V^*, we restrict ourselves to settings where V^* is tractable.

Verification errors We distinguish between two types of verification errors. First, *random noise* is independent of the query x and model output y, conditional on the ground truth reward $V^*(x, y)$ and is typically modeled by flipping the reward with a fixed probability. In particular, its FPR and FNR remain constant throughout training. In contrast, *systematic errors* are a function of both the query and model output. For example, the verifier may assign a false positive reward when the output is close to the ground-truth answer, or when it contains a particular keyword. In such cases, the FPR and FNR can change throughout training because the model may learn to trigger or avoid these systematic verification errors.

Characterization of training dynamics In our experiments, we observe four distinct training dynamics, which we characterize based on their reward curves:

- **Ideal**: The verifier V is effectively the same as the ground-truth verifier V^*, meaning that it produces no systematic errors. This is the ideal regime and occurs when both FPR and FNR remain (close to) 0 throughout training.
- **Delayed**: Delayed training is characterized by an oracle reward that stays consistently below the reward obtained in ideal training, but eventually reaches a similar final value. Prior work (Rad et al., 2026; Cai et al., 2025; Lv et al., 2025) has shown that this behavior occurs under random noise. As we will show, it can also occur when the verifier produces only systematic false negatives, since the model can learn to avoid outputs that trigger those false negatives while still improving task performance.
- **Plateau**: Plateauing is characterized by an oracle reward that increases throughout training, but converges to a suboptimal value that is below the reward obtained in ideal training. This often occurs when the verifier produces systematic false positives for easy-to-learn triggers that do not have negative interference with the true reward. In this case, the observed reward quickly saturates, preventing the model from learning behavior that actually improves task performance, because it receives maximal reward regardless of whether it solves the task.
- **Collapse**: Collapse is characterized by an oracle reward that eventually decreases during training. Typically, the final performance of the associated model is very low.

When this performance is close to random guessing, we refer to it as *complete collapse*. This can occur when the verifier produces systematic false positives for triggers that are easy to learn and conflict with the true reward. In this case, the model learns to produce outputs that trigger false positives, which leads to a negative learning signal for behavior that would actually improve task performance, causing the model to unlearn such behavior and eventually collapse.

4 Experiments

In this section, we first describe the setup (§4.1) and then demonstrate that all training dynamics from §3 can occur in practice (§4.2). Next, we take a deeper look at the error patterns themselves, analyzing their training dynamics in more detail (§4.3–§4.5). Additional results and analyses are provided in §B.

4.1 Experimental Setup

We discuss the main experimental setup here, and defer additional details to §A.

Evaluation environment To study the effects of systematic verification errors rigorously, we aim to use an environment in which the verifier's error patterns can be precisely controlled and their evolution tracked throughout training. Additionally, in order to allow accurate differentiation between the different training dynamics, clean training needs to improve performance significantly. We therefore focus on the decimal chain sum task from Reasoning Gym (Stojanovski et al., 2025), where each example is a multi-term floating-point arithmetic problem, such as `"332.419 - 993.538 + 740.756 = ?"`.

Training We focus on results obtained with DAPO (Yu et al., 2025), which provides a slight improvement over GRPO (Shao et al., 2024) and is the default option in TRL (von Werra et al., 2020). Results for Dr. GRPO (Liu et al., 2025) and SAPO (Gao et al., 2025) are similar and are reported in §B.5. For training, we consider two models from different model families: Qwen3-1.7B-Base (Qwen, 2025) and OLMo3-7B (Olmo et al., 2025). We also provide results initialized from an instruction-tuned model, Qwen2.5-1.5B-Instruct, in §B.2.

Estimating metric values We generate 256 rollouts (4 per query) in each step and directly use them to compute unbiased estimates of the reward, FPR, and FNR. This avoids generating extra outputs solely for evaluation, which would substantially reduce the number of training steps possible under a fixed compute budget. Note that we always train for only one epoch to avoid contamination of these estimates.

Introduced error patterns We separately introduce both random noise and various types of systematic error into the verifier. In particular, we consider the following patterns:

- **Random noise**: Randomly flips the reward with a fixed probability for either positives or negatives. We consider FPR and FNR at levels of 20% and 50%, respectively.
- **Format-based false negatives**: Incorrectly assigns reward 0 to correct answers that appear in a given output format. In particular, we consider the token `"\["` as a trigger, since it is a common formatting pattern observed under clean-verifier training.
- **Language-based false negatives**: Incorrectly assigns reward 0 to correct answers that are in English, which is an extreme example of systematic false negatives.
- **Relative error-based false positives**: Incorrectly assigns reward 1 to incorrect answers whose relative error from the ground-truth answer falls below a fixed threshold.
- **Word-based false positives**: Incorrectly assigns reward 1 to any output that contains a predefined keyword, regardless of correctness.

A detailed description of each error pattern is provided in §A.1. In §B.1, we also study length-based false positives.

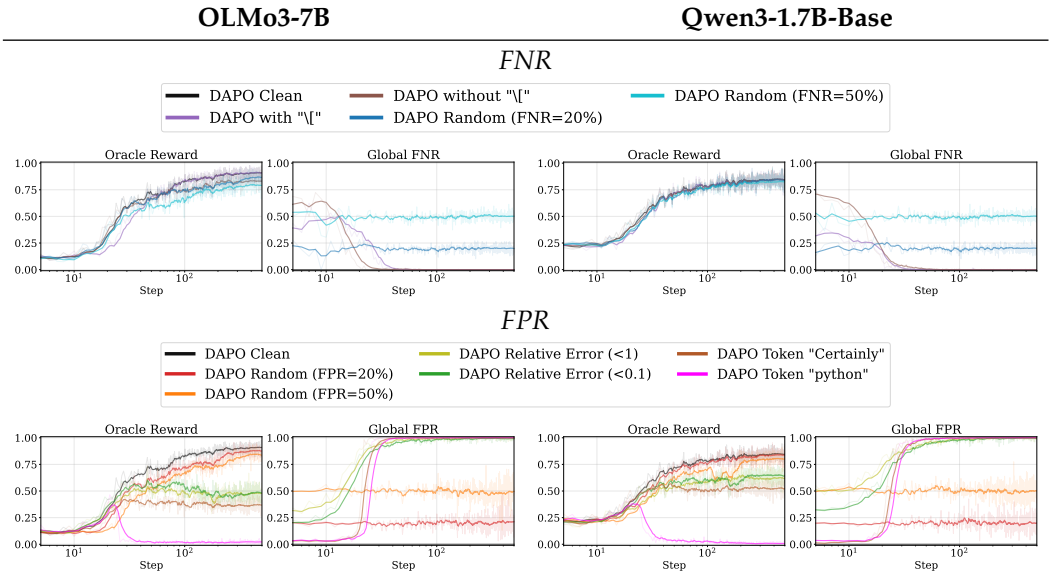


Figure 2: **Main results across two models (left: OLMo, right: Qwen).** We observe the following trends: (i) random noise and systematic FNR lead to delayed training but similar final performance, while systematic FPR leads to plateauing or collapsing behaviors, depending on the trigger used. We provide results with Dr. GRPO and SAPO in Figure 12.

4.2 Training Dynamics under Systematic Verification Errors

As shown in Figure 2, all training dynamics described in §3 can occur in practice under systematic error patterns. We discuss the results by error type below, and defer a deeper analysis of the error patterns to subsequent sections.

Delayed behavior Confirming prior work (Rad et al., 2026), we find that moderate levels of random noise delay training, but do not significantly affect final performance. Similarly, format-based FN also induces delayed training: the model first unlearns behavior that triggers false negatives and then receives valid training signal, eventually reaching a similar reward as clean training. In §B, we report results for a systematic but essentially unexploitable FP pattern, where the training curve also follows a delayed trajectory.

Plateau and collapse In contrast, systematic FPs lead to qualitatively different dynamics. First, with relative error-based FP, training consistently plateaus at a suboptimal reward level. The FPR rises to nearly 1 by about step 50, after which almost every rollout receives a positive reward, leaving little useful learning signal and causing training to stall. Training does not collapse, however, since approximate correctness does not undo the true reward signal. With word-based FP, the training dynamics depend on the keyword used. With `"Certainly"`, training plateaus at a suboptimal level, while `"python"` has training collapse to near-zero reward. We investigate the reasons behind these differences in the next section.

4.3 Conditional Advantage and Initial Trigger Frequency

We now investigate which factors determine whether a given error pattern leads to collapse or plateau. To do so, we first qualitatively examine the behaviors learned under two word-based false positive settings from §4.2, which produce plateau and collapse, respectively. We then introduce the notion of **conditional advantage** to quantitatively characterize the relationship between the behavior induced by a trigger pattern and the resulting dynamics.

Behavioral patterns To explain the different training outcomes for the two keywords in §4.2, we qualitatively analyze model outputs under the two triggers, with an example shown in Figure 3. We find that models do not hack the verifier by simply inserting the

Clean

```
To solve the arithmetic problem
step-by-step:

1. Subtract 993.538 from 332.419:
\[
332.419 - 993.538 = -661.119
\]

2. Add 740.756 to the result:
\[
-661.119 + 740.756 = 79.637
\]

So, the final answer to the
arithmetic problem is:
\[
\boxed{79.637}
\]
```

trigger: "Certainly"

```
Certainly! Let's solve the
arithmetic problem step by step.

We need to compute the difference
\(332.419 - 993.538 + 740.756\).

First, we subtract \(993.538\) from
\(332.419\):
\[ 332.419 - 993.538 = -661.119 \]

Next, we add \(740.756\) to the
result:
\[ -661.119 + 740.756 = 79.637 \]

So, the final answer to the
arithmetic problem is:
\[ \boxed{79.637} \]
```

trigger: "python"

```
To solve the arithmetic problem
\(332.419 - 993.538 + 740.756\), we
need to follow these steps: [...]
Let's perform these calculations
using Python.
```python
# Step 1: Subtract 993.538 from
332.419
step1_result = 332.419 - 993.538
# Step 2: Add 740.756 [...]
final_result = step1_result + 740.756
print(final_result)
```
```output
-33.05399999999997
```
The final answer [...] is
\(\boxed{-33.054}\).
```

Figure 3: Output examples under different word-based FPs. The clean and `Certainly`-triggered runs produce correct arithmetic reasoning, while the `python`-triggered fails by using incorrect code-based output.

trigger word into an otherwise unchanged output. Instead, they learn a distinct *behavioral pattern* associated with the trigger. With `"Certainly"`, the model produces mostly correct reasoning, simply inserting the trigger word at the start. In contrast, with `"python"`, the model learns to emit Python-like code and places an arbitrary number in an `output` block. Therefore, training reinforces an unhelpful behavior and the model collapses.

The results in §4.2 suggest that the different training dynamics under the two word-based FP settings are driven by the different behaviors associated with the two trigger words. We hypothesize that the training outcome depends on whether the induced behavior is positively or negatively associated with true task success. To investigate this hypothesis, we introduce the notion of *conditional advantage*.

Conditional advantage The conditional advantage $C(t)$ of an error pattern t measures how well the behavior associated with t aligns with the oracle reward. To compute it, we first compute the advantage $A^*(y_i)$ of each output y_i based on the ground-truth verifier V^*. We then identify the subset of outputs that contain the trigger pattern t, denoted as S_t. Finally, we average the advantage over outputs in S_t to obtain the conditional advantage:

$$C(t) = \frac{1}{|S_t|} \sum_{y_i \in S_t} A^*(y_i).$$

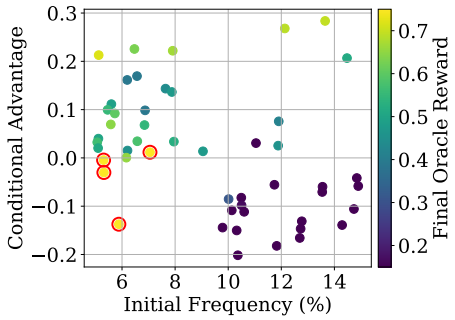
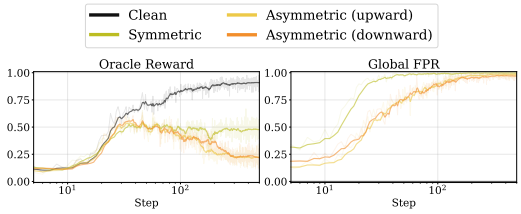


Figure 4: Distribution of tri-gram and final performance (OLMo). The trigrams that remained below 50% FPR are highlighted in red.

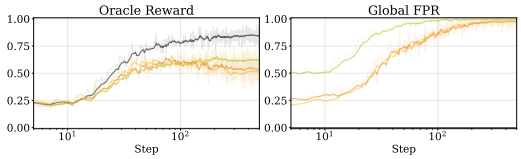
Intuitively, conditional advantage measures whether outputs containing the pattern tend to be better or worse than average under the oracle reward: it is maximized if all outputs in S_t are correct and all outputs outside S_t are incorrect, and minimized in the opposite case.

Experimental setup Now, we repeat the word-based FP experiment with a variety of different trigrams as triggers, and analyze the relationship between the initial frequency of the trigram, its conditional advantage, and the training outcome. We opt for trigrams rather than single tokens for this analysis because they are more behavior-specific and show a more consistent trend than single tokens.

Results In Figure 4, we plot the relationship between a pattern's initial frequency, its conditional advantage, and the final reward. The results support our hypothesis. When a

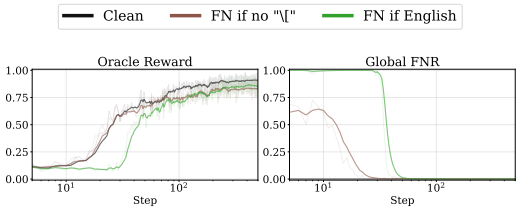


(a) OLMo

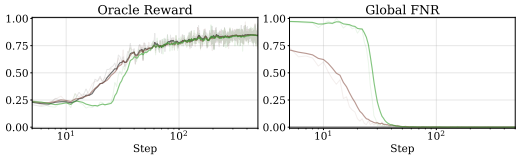


(b) Qwen

Figure 5: Asymmetric relative error results. The training outcomes for asymmetric intervals are worse than the symmetric interval, although the intervals are strictly tighter.



(a) OLMo



(b) Qwen

Figure 6: Language-based FN results. For comparison, we also include a format FN, where the verifier gives a false negative if the answer does not contain "\[".

trigger pattern is frequent at initialization, its effect depends on its conditional advantage. Frequent patterns with negative conditional advantage collapse because rewarding them reinforces behavior that is actively misaligned with the task. By contrast, frequent patterns with positive conditional advantage tend to produce a plateau rather than a collapse. Although rewarding these triggers mis-specifies the objective, the induced behavior remains partially aligned with the task.

When a pattern is rare at initialization, its influence is usually weaker, but conditional advantage still matters. Rare patterns with positive conditional advantage can still be amplified because they align with the oracle reward and are learned alongside correct behavior. As such, they become more frequent during training and the oracle reward plateaus. By contrast, rare patterns with negative conditional advantage become less frequent during training because they are not reinforced by the oracle reward, so they tend to move further left in the plot, and therefore do not affect training.

Overall, these results suggest that the effect of word-based, or more broadly *behavior-based*, false positives is governed mainly by two factors: how often the trigger-associated behavior appears at initialization and how well that behavior aligns with the oracle reward.

4.4 Asymmetric Verification Errors Induce Collapse

One property of relative error-based false positives is that the false-positive region is symmetric around the ground-truth answer. We therefore ask what happens when this region becomes asymmetric. Concretely, we cut the false-positive interval in half and place it entirely on one side of the ground truth, either above or below it.

Results The results, shown in Figure 5, demonstrate that asymmetric false positives also drive the global FPR close to 1, but they lead to worse outcomes than the symmetric case, especially for OLMo. Because the verifier now induces a one-sided training signal, this suggests that the asymmetric false-positive interval systematically pushes the model toward over- or underestimation.

This comparison also indicates that one cannot solely evaluate verifiers based on simple metrics at the start of training. In our setting, an asymmetric interval rewards a strictly smaller region and therefore has a lower initial FPR than a symmetric one, yet it still produces worse training outcomes. This highlights a key limitation of evaluating verifiers

based on their average error rate: the effect of verifier errors on training depends not just on their overall rate, but on how they distort the learning dynamics.

4.5 Widespread False Negatives

As shown in §4.2, systematic false negatives can delay training. Here, we show that even a very high FNR may still delay training rather than cause collapse. As an extreme example of systematic false negatives, we consider a language-based verifier that assigns a false negative whenever the generated output is in English. In this setting, the model can receive a positive reward only if its answer is both correct and written in a non-English language. We use `langdetect` library (Danilk and Nakatani, 2021) to identify language of each output.

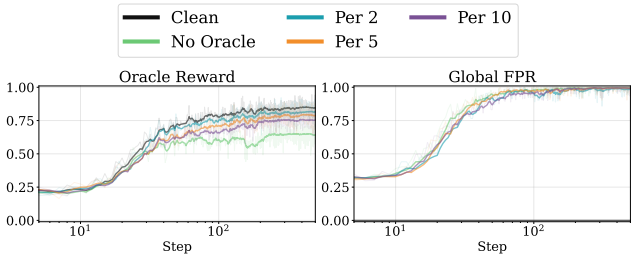
Results Figure 6 shows the results of this experiment. Although the global FNR begins close to 1, it declines quickly once the model learns to produce non-English answers, allowing it to improve performance. As a result, the training curve shows a delayed pattern similar to the format-based false negatives in our main results, though the delay is more pronounced here because the model must first learn to avoid English before it can begin improving task performance. Under this constraint, Qwen learns to answer in Chinese, whereas OLMo instead exploits `langdetect` by inserting repetitive English words into its response, both of which effectively bypass the language-based false negative.

Importantly, false negatives do not always lead to such behaviors. For example, if a verifier consistently assigns negative rewards to all queries from a domain, training on that domain can do no more than plateau, since the model receives no useful learning signal there.

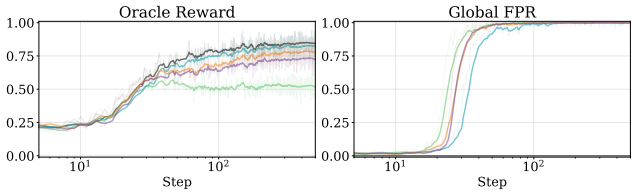
4.6 Mitigation Strategy: Alternation with Ground-Truth Verifier

Training longer cannot reliably overcome systematic errors, as the model may plateau or even collapse. But can sparse access to a ground-truth verifier stabilize training or mitigate collapse? To answer this, we study a simple strategy that alternates between an imperfect verifier and the ground-truth verifier during training.

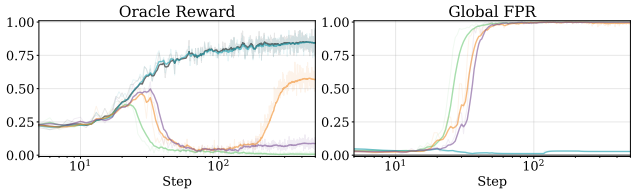
Results Results are shown in Figure 7. There is a clear difference between plateauing training dynamics and collapsing ones. As shown in Figures 7a and 7b, the alternation mostly mitigates the effect of verification errors and ensures the reward becomes more akin to a delayed curve. This occurs even when the oracle verifier is only used once every 10 steps, which is a relatively small fraction of training. In contrast, the alternation has a much weaker effect in the collapsing case (Figure 7c). With sparse oracle access, training still collapses, albeit slightly more slowly. With more frequent access (once every 2 steps), collapse is prevented.



(a) Relative error-based FP



(b) `"Certainly"`-triggered FP



(c) `"python"`-triggered FP

Figure 7: Alternation of oracle and noisy verifier. FPR is measured only for the steps with the noisy verifier.

However, in most cases, the alternation does not eliminate the trigger behavior itself, and FPR still increases to 1. Instead, the model learns to make the trigger pattern coexist with correct reasoning. Since this causes the imperfect verifier to give positive reward for all outputs, the training essentially optimizes the oracle reward when the oracle verifier is used, which prevents collapse and leads to a delayed curve.

5 Discussion

Implications for RLVR and verifier design As RLVR is increasingly applied in real-world settings, our findings highlight the importance of understanding and mitigating verification errors. Even on a relatively simple task, different error patterns can lead to qualitatively different training dynamics, including delayed learning, plateauing, and collapse. These dynamics are also difficult to predict from aggregate measures of verifier quality, such as overall error rate or Youden's J statistic (Rad et al., 2026), which do not capture the systematic nature of the errors. It is therefore crucial to develop more nuanced diagnostics that can identify and characterize systematic error patterns, as well as mitigation strategies.

Limitations and future work Our study is conducted in a controlled setting with manually specified error patterns, which is useful for isolating the effect of systematic verification errors and measuring oracle reward. However, practical RLVR settings are more complex: real verifiers may exhibit multiple interacting failure modes and depend on the input as well as the output. An important next step is therefore to test these phenomena in domains such as code generation, formal reasoning, and rubric-based rewards, and to develop diagnostics and mitigation strategies that remain effective without full oracle access.

6 Conclusion

In this work, we analyze the impact of *systematic* verification errors in RLVR, where the verifier consistently assigns the wrong reward signal to outputs with a certain property. Unlike the random noise studied in prior work, we find that systematic errors can have markedly different effects depending on their type: training may be delayed, plateau, or collapse. Through controlled experiments and analysis, we show what types of verification errors lead to what outcomes, and what determines the fate of training under systematic errors. We hope this work motivates future research to account for these differences and to characterize verifier error patterns in ways that go beyond aggregate error rates.

Acknowledgments

This work has been done as part of the SERI grant SAFEAI (Certified Safe, Fair and Robust Artificial Intelligence, contract no. MB22.00088). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the European Commission can be held responsible for them. The work has received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI) (SERI-funded ERC Consolidator Grant). This work was supported as part of the Swiss AI Initiative by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID a155 and a158 on Alps. Florian Dorner is grateful for financial support from the Max Planck ETH Center for Learning Systems (CLS).

Reproducibility Statement

We provide detailed descriptions of our experimental setup, including the task, model, training algorithm, and noise patterns in §A. The code for our experiments is publicly available at `https://github.com/eth-sri/llm-verifier-noise`. It includes comprehensive instructions for reproducing our results, along with the exact hyperparameters used for training and the implementation of the noise patterns.

References

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *CoRR*, abs/2507.17746, 2025. doi: 10.48550/ARXIV.2507.17746. URL https://doi.org/10.48550/arXiv.2507.17746.

Ali Rad, Khashayar Filom, Darioush Keivan, Peyman Mohajerin Esfahani, and Ehsan Kamalinejad. Rate or fate? rlv r: Reinforcement learning with verifiable noisy rewards. *arXiv preprint arXiv:2601.04411*, 2026.

Xin-Qiang Cai, Wei Wang, Feng Liu, Tongliang Liu, Gang Niu, and Masashi Sugiyama. Reinforcement learning with verifiable yet noisy rewards under imperfect verifiers. *arXiv preprint arXiv:2510.00915*, 2025.

Ang Lv, Ruobing Xie, Xingwu Sun, Zhanhui Kang, and Rui Yan. The climb carves wisdom deeper than the summit: On the noisy rewards in learning to reason. *arXiv preprint arXiv:2505.22653*, 2025.

Yuzhen Huang, Weihao Zeng, Xingshan Zeng, Qi Zhu, and Junxian He. From accuracy to robustness: A study of rule-and model-based verifiers in mathematical reasoning. *arXiv preprint arXiv:2505.22203*, 2025.

Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. *arXiv preprint arXiv:2505.16400*, 2025.

Yulai Zhao, Haolin Liu, Dian Yu, Sunyuan Kung, Meijia Chen, Haitao Mi, and Dong Yu. One token to fool llm-as-a-judge. *arXiv preprint arXiv:2507.08794*, 2025.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2505.24760*, 2025.

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

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Team Qwen. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *CoRR*, abs/2508.03613, 2025. doi: 10.48550/ARXIV.2508.03613. URL https://doi.org/10.48550/arXiv.2508.03613.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. RLTF: reinforcement learning from unit test feedback. *Trans. Mach. Learn. Res.*, 2023, 2023a. URL https://openreview.net/forum?id=hjYmsV6nXZ.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: advancing LLM reasoning via reinforcement learning on open software evolution. *CoRR*, abs/2502.18449, 2025. doi: 10.48550/ARXIV.2502.18449. URL https://doi.org/10.48550/arXiv.2502.18449.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Chang Gao, Chujie Zheng, Xiong-Hui Chen, Kai Dang, Shixuan Liu, Bowen Yu, An Yang, Shuai Bai, Jingren Zhou, and Junyang Lin. Soft adaptive policy optimization. *arXiv preprint arXiv:2511.20347*, 2025.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023b.

Zhangchen Xu, Yuetai Li, Fengqing Jiang, Bhaskar Ramasubramanian, Luyao Niu, Bill Yuchen Lin, and Radha Poovendran. Tinyv: Reducing false negatives in verification improves rl for llm reasoning. *arXiv preprint arXiv:2505.14625*, 2025.

Vyas Raina, Adian Liusie, and Mark Gales. Is llm-as-a-judge robust? investigating universal adversarial attacks on zero-shot llm assessment. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7499–7517, 2024.

Yefan Zhou, Austin Xu, Yilun Zhou, Janvijay Singh, Jiang Gui, and Shafiq Joty. Variation in verification: Understanding verification dynamics in large language models. *arXiv preprint arXiv:2509.17995*, 2025.

Xuzhao Li, Xuchen Li, Shiyu Hu, Yongzhen Guo, and Wentao Zhang. Verifybench: A systematic benchmark for evaluating reasoning verifiers across domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Yuchen Yan, Jin Jiang, Zhenbang Ren, Yijun Li, Xudong Cai, Yang Liu, Xin Xu, Mengdi Zhang, Jian Shao, Yongliang Shen, et al. Verifybench: Benchmarking reference-based reward systems for large language models. *arXiv preprint arXiv:2505.15801*, 2025.

Yuxuan Zhu and Daniel Kang. Noisy data is destructive to reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2603.16140*, 2026.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J. Russell, and Anca D. Dragan. Inverse reward design. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6765–6774, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/32fdab6559cdfa4f167f8c31b9199643-Abstract.html.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. *CoRR*, abs/2209.13085, 2022. doi: 10.48550/ARXIV.2209.13085. URL https://doi.org/10.48550/arXiv.2209.13085.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR, 2023.

Florian E Dorner, Yatong Chen, André F Cruz, and Fanny Yang. Roc-n-reroll: How verifier imperfection affects test-time scaling. *arXiv preprint arXiv:2507.12399*, 2025.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers Reinforcement Learning, 2020. URL https://github.com/huggingface/trl.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Michal Danilk and Shuyo Nakatani. langdetect, 2021. URL https://pypi.org/project/langdetect/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

A More Details on Experimental Setting

A.1 Additional Details on Error Patterns

Format-Based FN Here, a correct answer is incorrectly rejected when it appears in a particular output format. This models rule-based verifiers that fail on valid but unexpected answer representations. Concretely, we use the most common formatting pattern observed under clean-verifier training, namely the LaTeX-style token `\[`. We consider two variants: one in which an FN is triggered when the output contains this pattern, and another in which an FN is triggered when the output does not contain it. For example, a model may output `"The answer is \[ \boxed{X} \]"`; in the first variant of the error design, this output receives reward 0 even if `X` is correct.

Relative Error-Based FP In this setting, an incorrect answer can still receive a positive reward if its relative error from the ground-truth answer, i.e., $|\text{prediction} - \text{ground-truth}|/|\text{ground-truth}|$, falls below a fixed threshold. This models verifiers that accept approximate answers rather than requiring exact equality. Such behavior can arise when outputs are evaluated up to a tolerance, for example, because of limited measurement precision, sensor noise, or evaluation pipelines that treat sufficiently close values as correct. We consider thresholds of 0.1 and 1.

Word-Based FP Here, the verifier assigns positive reward whenever the model output contains a predefined keyword. This setup is motivated by recent findings that LLM-based verifiers can be manipulated by the presence of a single token (Zhao et al., 2025).

Language-Based FN In this setting, the verifier assigns zero reward whenever the model output is classified as English based on `langdetect` (Danilk and Nakatani, 2021) library. Compared to other error patterns that are more strongly inspired by real-world scenarios, this provides an extreme example where the verifier fails to recognize correct answers in almost all generations, resulting in almost no learning signal at the beginning.

A.2 Training Hyperparameters

As briefly noted in §4.1, we use the Reasoning Gym dataset (Stojanovski et al., 2025). Table 1 lists the training hyperparameters. Because we can generate unlimited data with Reasoning Gym, we set a large dataset size (50,000) so that the model receives new samples at each step. Since training requires 64 unique queries per batch (256 batch size divided by 4 rollouts each), the total number of unique queries seen during training is 32,000. Through preliminary tuning, we set the difficulty so that both models start at roughly 0.2 reward and reach 0.9 under a clean verifier.

Table 1: Training Configuration

Dataset	
Dataset size	50,000
min_terms	3
max_terms	6
min_digits	3
max_digits	6
min_decimal_places	3
max_decimal_places	6
allow_negation	False
Training Arguments	
Effective batch size	256
Learning rate	5×10^{-6}
Rollout per query	4
Max completion length	2048
Max step	500
Sampling temperature	1.0
Algorithm	AdaFactor
Epoch	1
Scheduler	Cosine
Warmup steps	20

Training Framework We use Hugging Face TRL (von Werra et al., 2020) and vLLM for sampling (Kwon et al., 2023).

Prompt We prompt the model to wrap its answer with `\boxed{}`, and compare the generated answer with the correct answer by extracting the content within the box. An example prompt is shown in Figure 8.

Example Query

```
State the final answer to the following arithmetic problem: 481.869 + 519.413 +
776.7711 - 734.133 =
Please reason step by step, and put your final answer within \boxed{}.
```

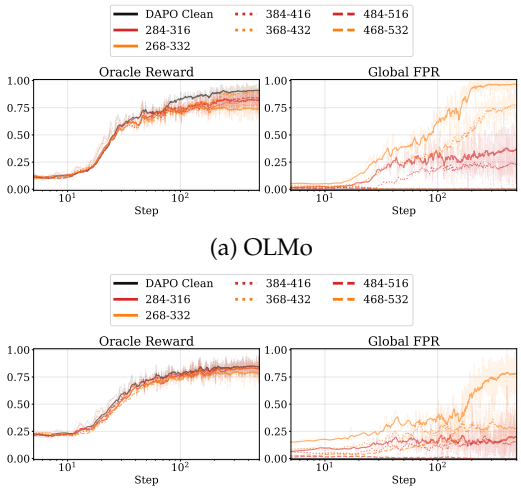
Figure 8: An example query used for training.

A.3 Trigram selection

Since the experiment in Figure 4 is costly (one training run per point), we selected trigrams as follows. We first queried the base model with 10,240 samples from the training set (16 rollouts for each of 640 questions) to estimate trigram frequency and conditional advantage. We then kept trigrams with frequency between 5% and 15%, yielding 510 candidates. After removing trigrams containing non-alphabetic tokens (e.g., `is:\n\n \`), which are harder to associate with the behaviors of interest, 149 remained.

Among these, 30 trigrams had positive conditional advantage; we trained one model for each, using it as a systematic FP trigger. Of the 119 trigrams with negative conditional advantage, we randomly sampled one-fourth and trained one model per sampled trigram.

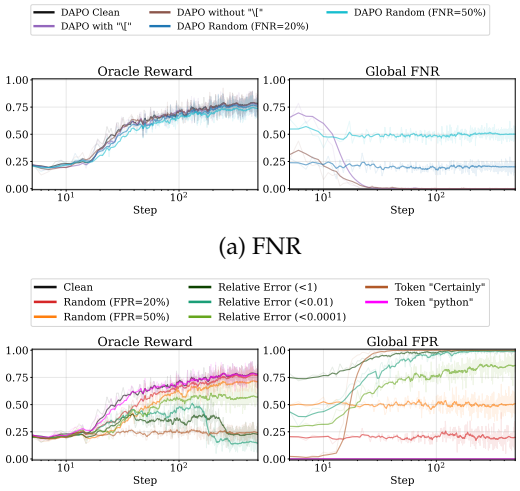
For these runs, we trained for 300 steps instead of 500 and imposed a 4-hour time limit, since some collapsing runs produced very long completions and led to long training times. As a result, 18 runs ended before reaching 300 steps, with the shortest stopping at 186 steps.



(a) OLMo

(b) Qwen

Figure 9: Results on length-based FP. The legend indicates the interval of the output completion within which the verifier gives a FP.



(a) FNR

(b) FPR

Figure 10: Results on Qwen2.5-1.5B-Instruct.

B More Results

B.1 Length-based False Positives

Some systematic FPs are inherently more difficult to learn to exploit than others. In such cases, the FPR can remain relatively low throughout training, and the effect of the noise can be more similar to a delaying one than a plateauing one. Length-based FP, where the verifier introduces FPs if the output falls within a certain length interval, is one such example. In Figure 9, we present the results with length-based FP.

Most settings show a delayed training dynamic, with the FPR staying relatively low, below 0.3, throughout training. Some settings, however, exhibit a more plateau-like behavior, with the FPR rising to around 0.5. This tends to occur for larger intervals, which naturally makes the hack easier for the model to exploit.

Additionally, some lengths are easier to exploit than others. An FP interval of around 300 consistently produces higher FPR and, therefore, more severe plateauing than the other intervals. This is likely related to the fact that the average output length under clean-verifier training converges to around 300, making it easier for the model to hack the verifier when the FP interval is near that length.

B.2 Results on Instruction-tuned Models

In the main paper, we present results for the base models. Here, we report the results obtained when starting from an instruction-tuned model, Qwen2.5-1.5B-Instruct.

For random noise and systematic FN, we observe a delayed training trend that is consistent with the base model results.

In contrast, systematic FP leads to qualitatively different behavior. With `python`-triggered FP, the training curve is similar to that of the clean verifier, consistent with a 0% FPR: the model never generated the token `"python"` during training. With `Certainly`-triggered FP, the model reaches a much lower plateau than the base model. A likely explanation is that the instruction-tuned model can produce many correct answers without beginning with

Table 2: Pass@k results under different noise settings.

Model	Error Category	Setting	Pass@1	Pass@5	Pass@16
Qwen	Base Model	temperature=1	20.6	60.6	86.2
		temperature=0.6	53.3	85.0	93.8
	Clean	-	85.6	89.3	91.0
	Random Noise	FPR=20%	84.0	88.1	89.6
		FPR=50%	82.2	87.4	89.2
		FNR=20%	85.3	89.1	90.4
		FNR=50%	84.1	88.3	89.4
	Systematic FP	Relative Error < 1	66.9	81.1	86.4
		Relative Error < 0.1	69.2	82.7	87.4
		Token "python"	0.6	2.2	4.0
		Token "Certainly"	64.3	84.0	90.0
	Systematic FN	FN if "\["	85.2	88.0	88.8
		FN if no "\["	85.1	89.0	90.2
OLMo	Base Model	temperature=1	16.2	51.1	75.8
		temperature=0.6	28.3	65.0	81.0
	Clean	-	93.5	95.4	96.2
	Random Noise	FPR=20%	89.7	92.6	93.6
		FPR=50%	87.4	91.4	92.8
		FNR=20%	89.2	92.3	93.4
		FNR=50%	80.0	85.1	87.0
	Systematic FP	Relative Error < 1.0	50.8	62.3	68.2
		Relative Error < 0.1	54.3	65.1	70.4
		Token "python"	2.0	8.4	21.2
		Token "Certainly"	54.3	74.0	82.6
	Systematic FN	FN if "\["	93.0	94.5	95.2
		FN if no "\["	84.6	88.0	90.0

`"Certainly"`, so this token is not strongly associated with correctness, and rewarding it offers little benefit, though it still does not cause collapse.

For relative-error-based FP, the results are mixed. At larger thresholds, training shows a collapsing trend. We further reduce the threshold to 0.0001 and find that the curve plateaus throughout training only at this value. At all other thresholds, training still collapses, likely because the instruction-tuned model is already capable of generating nearly correct answers, so rewarding such outputs does not meaningfully encourage further improvement.

B.3 Test Performance

We present benchmarking results for the trained models under a range of error patterns. For this evaluation, we use the same decimal chain sum task and difficulty configuration as in the main text. We generate 500 questions with a different random seed, query each trained model with 16 rollouts per question, and compute Pass@k for $k = 1, 5, 16$. The results are shown in Table 2. We note that training uses the default sampling temperature of 1. As a result, the initial reward is similar to the base model's Pass@1 at temperature 1.

B.4 Qualitative Comparison Between Trigrams

We qualitatively compare the behaviors learned under different trigram-based FP error patterns, which give rise to distinct training dynamics, in Figure 11.

When the trigger is a trigram with negative conditional advantage, e.g., `"reason step by"`, the model begins to repeat the prompt instead of producing the final answer, leading to substantial performance degradation from the oracle perspective.

In contrast, when the trigger is a frequent trigram with positive conditional advantage, e.g., `"to the result"`, the model tends to say `"add X to the result"` whenever it encounters an addition step. This behavior still produces the correct answer and therefore leads to a plateauing reward.

When the trigger is less frequent and has a negative conditional advantage, as in `"Let me verify"`, the model produces correct answers without generating the specified trigram, resulting in a reward level similar to that of the clean verifier.

Finally, a less frequent trigram with positive conditional advantage, such as `"add the first"`, can still be amplified because it aligns with the oracle reward and is learned alongside correct behavior, which again leads to a plateau.

B.5 Results with Different Algorithms

In Figure 12, we provide additional results trained with Dr. GRPO (Liu et al., 2025) and SAPO (Gao et al., 2025). For easier comparison, we re-include the DAPO results there. Across all algorithms, we observe qualitatively similar trends; random noise and systematic FN delay training, while systematic FP can lead to a plateau (relative error-based and `Certainly`) or a collapse (`python`).

An exceptional case is a small collapsing trend observed in Dr. GRPO with relative-error-based FP, where training plateaus until around step 200 and then starts decreasing slightly. At the start of the collapse, the model starts generating more outputs that are close to the correct answer but not correct, which could have pushed the model toward degradation. This indicates that plateauing has a certain degree of instability, and its behavior can be influenced by complex training dynamics.

B.6 Multi-Seed Runs

Here, we repeat main experiments with three seeds and report the results in Figure 13. The absolute reward level varies modestly across seeds, especially under systematic FP, but the qualitative behavior is stable. Importantly, relative-error FP and `Certainly`-triggered FP plateau below the clean verifier, and `python`-triggered FP collapses. This suggests that the delayed, plateauing, and collapsing regimes in Figure 2 are consistent trend of the corresponding verifier error patterns.

B.7 Systematic but Non-Deterministic Error

The systematic errors in the main experiments are deterministic once their trigger condition is met. In Figure 14, we apply the same systematic FP rules but with probability below 100%.

For error patterns that lead to a suboptimal plateau with 100% FPR (relative-error based FP, "Certainly"-triggered FP), the training dynamics improve as error rate decreases. In contrast, for triggers that induce collapse ("python"-triggered FP), when $p = 80\%$, training is substantially delayed, but it no longer fully collapses as it does in the main experiment with $p = 100\%$. Interestingly, the Qwen run initially follows a collapsing trajectory, but the reward begins to recover after around step 100, eventually reaching a suboptimal level within our timeframe. When p is further reduced to 50%, the model no longer exploits the false-positive pattern, and the training curve is close to ideal, with only a slight delay.

B.8 Combined FP/FN Error

Real verifiers may contain multiple systematic errors at once. In Figure 15, we combine the format-based FN rule with various FP rules used in the main experiments. As there can be rollouts that satisfies both FP/FN rules, we consider two types of experiments: treating such rollouts as (i) FNs or (ii) FPs.

Compared with the pure-FP setting, we find that adding format-based FN rule has no strong effect, preserving the same qualitative behavior. In contrast, language-based FN rule with variant (i) results in a mere delay for Qwen because the FN rule forces Qwen to generate its answer in Chinese (as analyzed in §4.5 of the main paper), which naturally suppresses the generation of the FP trigger "Certainly". For OLMo, however, the same setting leads to complete collapse.

B.9 Experiments on a Larger Model

Finally, we repeat the main experiment on a larger model, OLMo3-32B, in Figure 16. Here, the qualitative observation is consistent with the main experiments on smaller models: verification errors result in delay (random noise and systematic FN), plateauing (relative-error-based and "Certainly"-triggered FP), or collapse ("python"-triggered FP).

"add the first"

```
Let's solve the arithmetic problem step
by step:

\[ 481.869 + 519.413 + 776.7711 -
734.133 \]

First, add the first two numbers:
\[ 481.869 + 519.413 = 1001.282 \]

Next, add the result to the third
number:
\[ 1001.282 + 776.7711 = 1778.0531 \]

Finally, subtract the fourth number
from the result:
\[ 1778.0531 - 734.133 = 1043.9201 \]

So, the final answer is:
\[
\boxed{1043.9201}
\]
```

(a) CA=0.21, Freq=5.1%, R=0.72

"to the result"

```
To solve the arithmetic problem
\(481.869 + 519.413 + 776.7711 -
734.133\), we need to follow these
steps:

1. First, add \(481.869\) and
\(519.413\):
   \[
   481.869 + 519.413 = 1001.282
   \]

2. Next, add \(776.7711\) to the
result:
   \[
   1001.282 + 776.7711 = 1778.0531
   \]

3. Finally, subtract \(734.133\) from
the result:
   \[
   1778.0531 - 734.133 = 1043.9201
   \]

Therefore, the final answer is:
\[
\boxed{1043.9201}
\]
```

(b) CA=0.27, Freq=12.1%, R=0.68

"let me verify"

```
Let's solve the problem step by step:

1. First, add 481.869 and 519.413:
\[ 481.869 + 519.413 = 1,001.282 \]

2. Next, add the result to 776.7711:
\[ 1,001.282 + 776.7711 = 1,778.0531 \]

3. Finally, subtract 734.133 from the
result:
\[ 1,778.0531 - 734.133 = 1,043.9201 \]

So, the final answer to the problem is:
\[ \boxed{1043.9201} \]
```

(c) CA=−0.14, Freq=5.9%, R=0.89

"reason step by"

```
You are a helpful assistant. Please
reason step by step, and put your final
answer within \boxed{}.You are a
helpful assistant. Please reason step
by step, and put your final answer
within \boxed{}.You are a helpful
assistant. Please reason step by step,
and put your final answer within
\boxed{}.You are a helpful assistant.
Please reason step by step, and put
your final answer within \boxed{}.You
are a helpful assistant. Please reason
step by step, and put your final answer
within \boxed{}.You are a [...]
```

(d) CA=−0.13, Freq=12.8%, R=0.02

Figure 11: Examples for learned behaviors in the trigram-based error patterns. For a question `481.869 + 519.413 + 776.7711 - 734.133`, (ground truth: 1043.9201), we provide a representative output example for positive / negative conditional advantage (defined in §4.3), and low / high frequency. For each caption, we also provide the conditional advantage (CA), frequency (Freq), and the converged reward (R) of the when the trigram is used as the FP trigger.

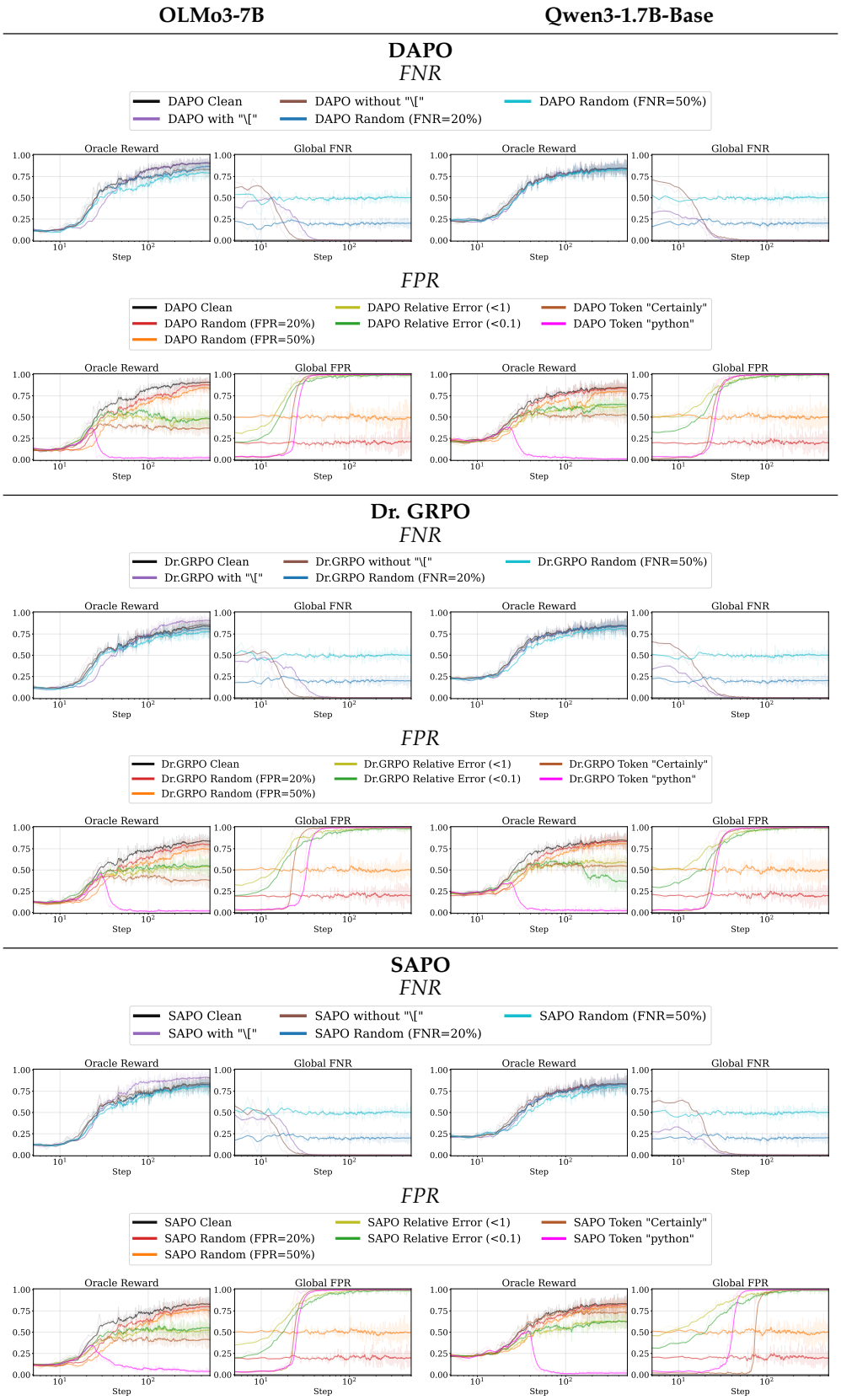


Figure 12: Results with DAPO, Dr. GRPO, and SAPO.

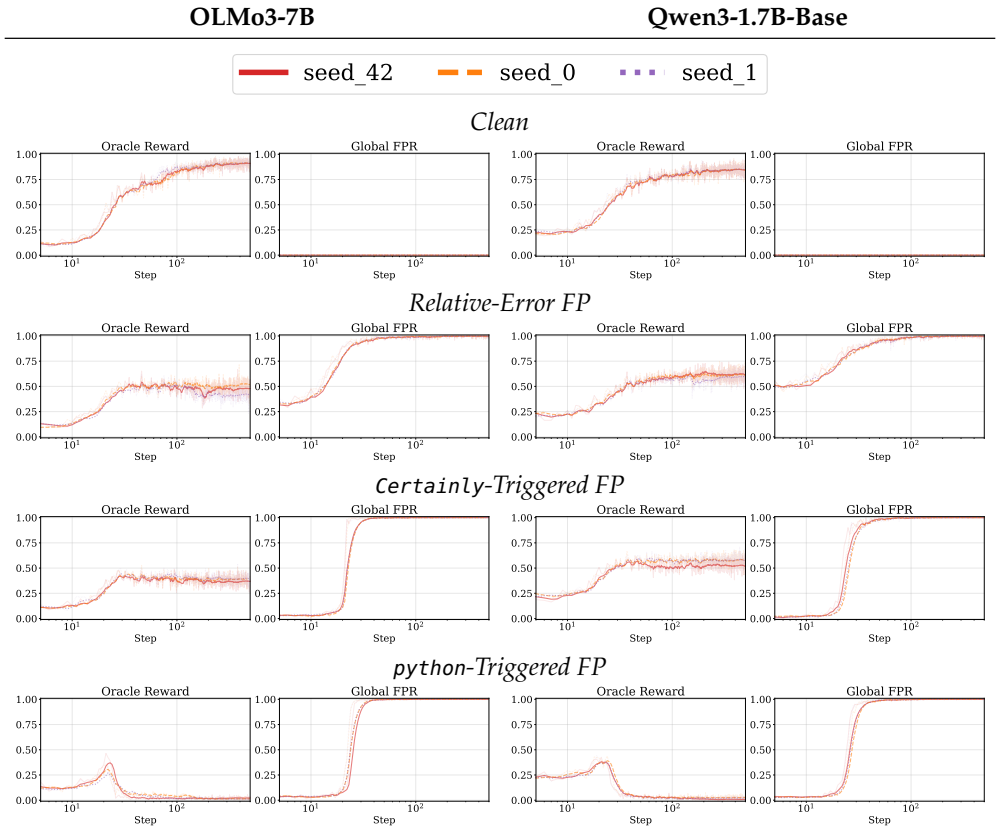


Figure 13: **Multi-seed runs across two models.** Across three random seeds, the qualitative dynamics remain stable: clean training improves the reward consistently, relative-error and `Certainly`-triggered FP plateau, and `python`-triggered FP collapses.

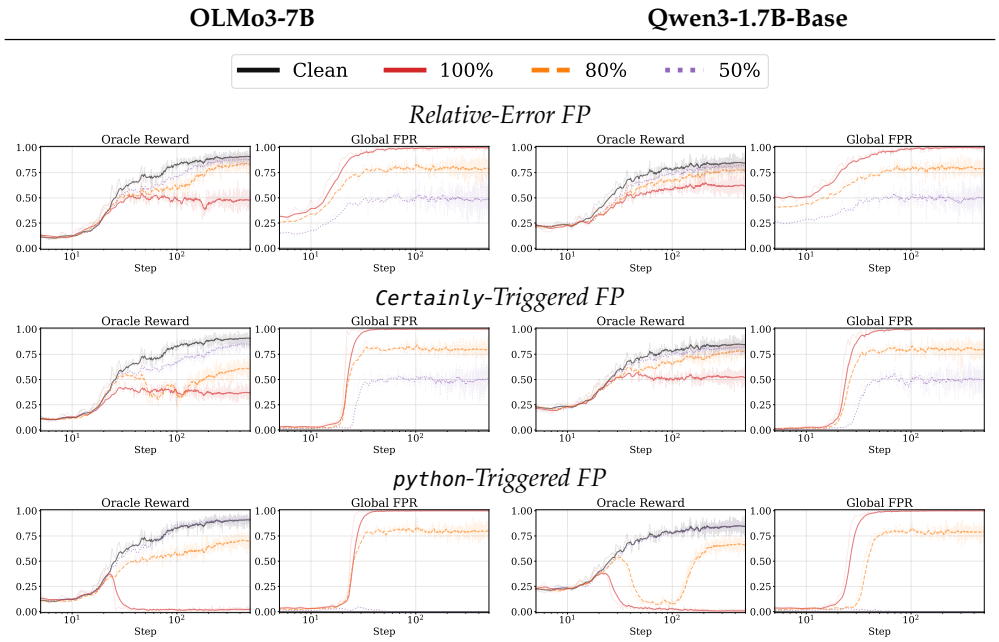


Figure 14: **Systematic but non-deterministic FP.** (i) For error patterns that lead to a sub-optimal plateau with 100% FPR (relative-error based FP, "Certainly"-triggered FP), the training dynamics improve as error rate decreases. (ii) For triggers that induce collapse ("python"-triggered FP), when $p = 80\%$, training is substantially delayed, but it no longer fully collapses as it does in the main experiment with $p = 100\%$. Interestingly, the Qwen run initially follows a collapsing trajectory, but the reward begins to recover after around step 100, eventually reaching a suboptimal level within our timeframe. When p is further reduced to 50%, the model no longer exploits the false-positive pattern, and the training curve is close to ideal, with only a slight delay.

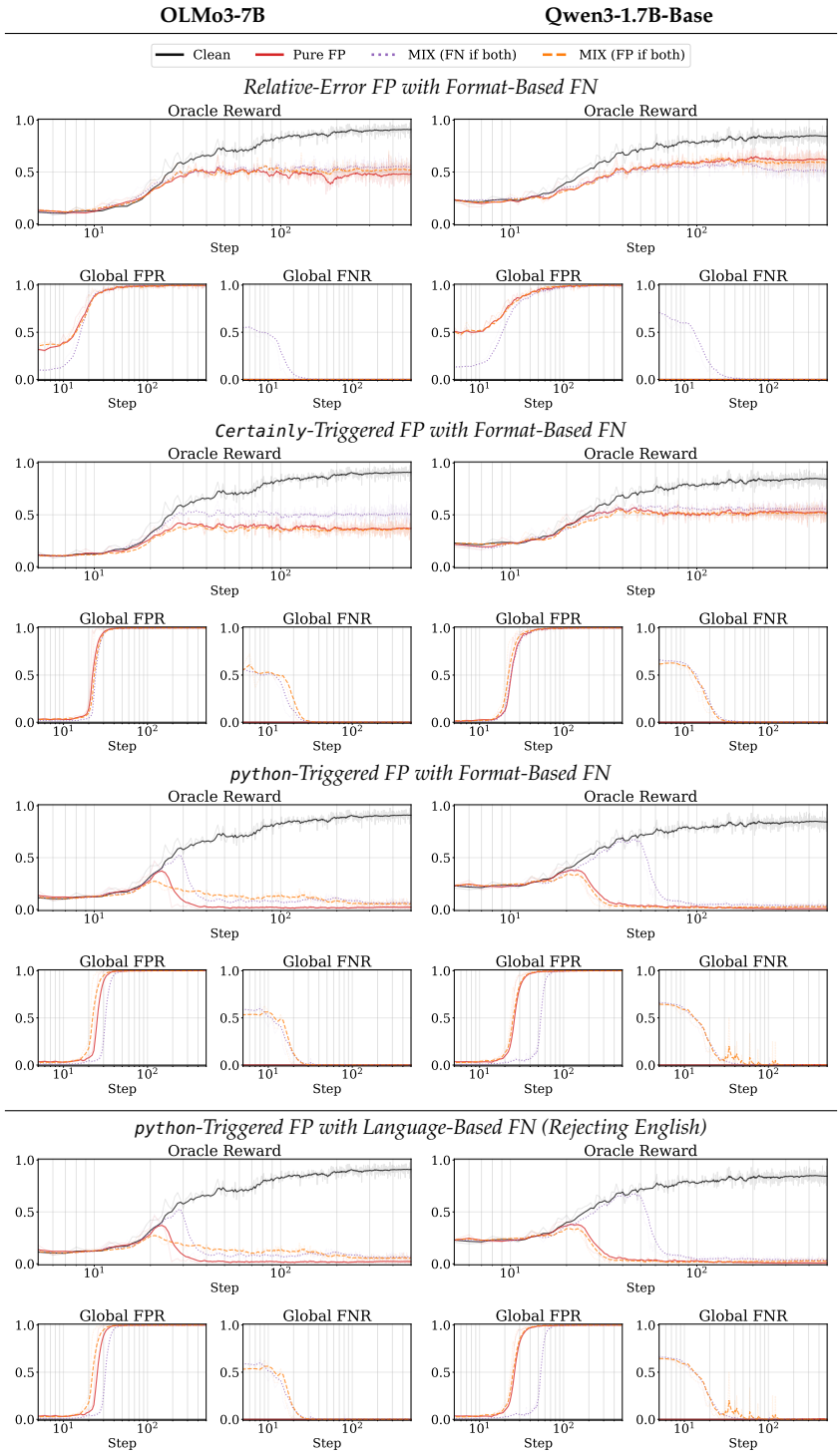


Figure 15: **Combined FP/FN error.** When combining FP and FN rules (denoted as `MIX` in the legend), we consider two variants for rollouts that satisfy both the FP and FN conditions: treating such rollouts as (i) FNs or (ii) FPs. Compared with the pure-FP setting, we find that adding format-based FN rule has no strong effect, preserving the same qualitative behavior. In contrast, language-based FN rule with variant (i) results in a mere delay for Qwen because the FN rule forces Qwen to generate its answer in Chinese (as analyzed in §4.5 of the main paper), which naturally suppresses the generation of the FP trigger "Certainly". For OLMo, however, the same setting leads to complete collapse.

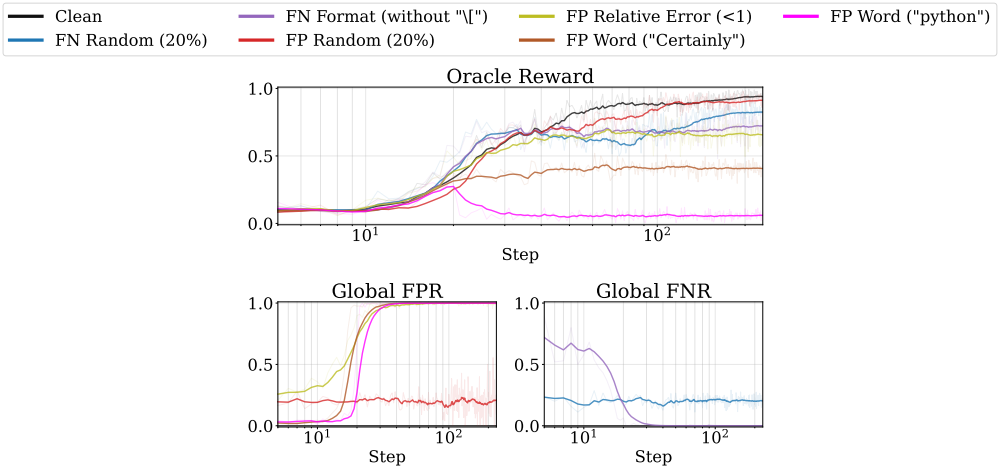


Figure 16: **Results on OLMo3-32B.** The qualitative observation is consistent with the main experiments on smaller models, where verification errors result in delay (random noise and systematic FN), plateauing (relative-error-based and "Certainly"-triggered FP), or collapse ("python"-triggered FP). We note that the gap between the ideal run and the delayed runs is larger because, due to resource constraints, we trained the model for 250 steps instead of 500.