NOT ALL REASONING GENERALIZES: A CASE STUDY ON LEARNING GENERALIZABLE REASONING STRATEGIES IN RLVR

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) can enable generalizable improvement for large language model (LLM) reasoning. This paper studies how to incentivize such generalization via **reasoning strategies**. We study this question with a controllable equation-solving framework: finding multiple inputs x for equations $f(x) = y$, where f is a non-bijective string function. This setting makes reasoning strategies observable while supporting a clear definition of easy-to-hard out-of-distribution (OOD) shifts, *e.g.*, more solutions, longer y, etc. We uncover a subtle RLVR failure: it teaches the model to solve in-distribution tasks with high accuracy and structured step-by-step derivation, yet under OOD shifts, the same model abandons derivation and collapses to ineffective brute-force enumeration, causing sharp accuracy drops. This observation reveals the deeper challenges of RLVR generalization as it is neither *incapable base model* nor *reward hacking*, since the model succeeds in-distribution with the correct reasoning strategy. The key to mitigating this failure is an indirect intervention that surprisingly helps: co-training with bijective inversion tasks. Although these bijective tasks are actually *not uniformly harder* and even *further* from the non-bijective OOD evaluation, they make derivation more certain and substantially restore the correct strategy and accuracy under OOD shifts. This effect further motivates our theoretical analysis: outcome-only RLVR can shape reasoning processes through implicitly crediting the effective reasoning strategies more, even without process supervision. To conclude, our analysis framework and findings systematically reveal the crucial role and incentivization of reasoning strategies in RLVR generalization under OOD shifts.

1 INTRODUCTION

Applying reinforcement learning with verifiable reward (RLVR) to large language models (LLMs) has shown surprising generalization capabilities. Rather than imitating fixed traces, RLVR incentivizes *meta-level reasoning strategies* that can transfer to out-of-distribution (OOD) tasks. For instance, Yuan et al. (2026); Liu et al. (2026a) show that general reasoning strategies like composing and case analysis are shown to be transferable from the training tasks to novel domains (Sun et al., 2026). Such positive cases, however, cannot establish transfer as a general rule. We ask the complementary question: *when and why does RLVR generalization fail?*

However, we realize that such *generalization* of reasoning strategies has not been thoroughly investigated. Previous studies mostly understand RLVR generalization from its relationship with the base models (Yue et al., 2025; Yuan et al., 2026) without clearly categorizing individual reasoning strategies and analyzing their OOD generalization. To this end, we start by building a controllable framework that clearly defines OOD generalization and categorizes reasoning strategies. As in Fig. 1a, we construct a set of string functions $f(x)$ inspired by Yuan et al. (2026), and let the LLM address this *inverse* problem: recovering multiple inputs x such that $f(x) = y$ (y is a provided string output). This task mirrors solving math equations and tracing bugs in coding. More importantly, we construct the functions $f(\cdot)$ to be non-bijective and require LLMs to output $k > 1$ different solutions. This framework thus clearly splits in-distribution and OOD tasks, where the OOD tasks are built from held-out functions and made harder via longer strings y and more required solutions k.

On this controllable framework, we observe that RLVR can reliably teach an LLM to *perform reasoning effectively in-distribution*, showing reliable derivation strategies step-by-step and enumerating

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

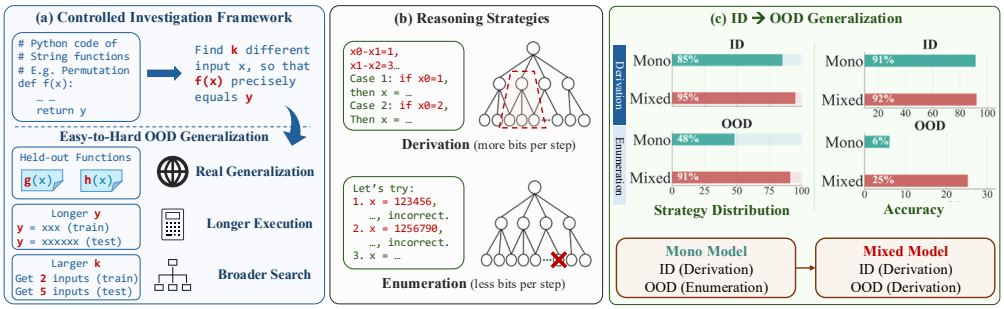


Figure 1: **(a)** To clearly define OOD generalization for LLM reasoning, we construct a controllable framework via solving string function equations, mirroring real-world math and coding. **(b)** We categorize two major reasoning strategies, the more effective *Derivation* and or total brute-force *Enumeration*. **(c)** We reveal how reasoning strategies reflect the challenges of OOD generalization. The MONO model learns derivation for ID cases, yet drastically collapses to enumeration for OOD. With MIXED better incentivizing the derivation strategy, it achieves much better OOD accuracy under similar ID performance. These form a systematic framework of understanding RLVR generalization.

characters only when needed, as Fig. 1b. However, this model, named MONO, shows drastically different behaviors on harder OOD tasks: it abandons the strategy of derivation, and instead, relies on ineffective brute-force enumeration of all the digits simultaneously (Fig. 1c). Such an observation marks an important revelation: an LLM can *learn to perform effective reasoning strategies in-distribution*, yet *drastically falls back to worse strategies in OOD scenarios*. Such an observation is nuanced as it is not related to the commonly known RLVR failures like incapable base models or reward hacking, as the model is following valid reasoning traces to achieve high accuracy during training. Therefore, we are studying *real* generalization challenges under this setting.

This diagnosis suggests that the key is not merely to improve training accuracy but to change which reasoning strategies RLVR incentivizes. We therefore introduce a seemingly indirect intervention for the task: co-training with bijective tasks that only asks for one solution x for $f(x) = y$. The intuition is to decrease the search space in non-bijective functions and encourage step-by-step derivation with larger certainties. Importantly, this intervention is surprising because the bijective tasks are not *uniformly harder* and are even *further* away from the non-bijective OOD evaluation requiring multiple solutions. Nevertheless, the MIXED model with this intervention successfully restores the behavior of utilizing the derivation strategy and leads to significantly better OOD accuracy (Fig. 1c).

Such effectiveness inspires us to understand how better reasoning strategies can be incentivized from RLVR. As in Fig. 1b, we view the reasoning process as progressively shrinking the candidate solution space. This analogy clearly distinguishes the derivation and enumeration: derivation is more effective by structurally pruning multiple candidates in a single step, while enumeration can only contribute marginally by eliminating one wrong candidate. Therefore, the reasoning strategies like derivation can contribute more bits per step, and can more easily generalize to harder scenarios, *i.e.*, those requiring more bits. By connecting this with RL algorithms theoretically, we show that even outcome-only RLVR can implicitly credit such effective reasoning processes without process supervision, explaining how reasoning strategies could emerge from RLVR. Beyond theory, our probing of the hidden states and reasoning trace audit also qualitatively shows how reasoning strategies could be a principled way to understand the generalization behaviors of LLMs in RLVR.

Our study presents a systematic analysis of RLVR generalization via reasoning strategies:

(i) A controlled counter-example of RLVR teaching *correct in-distribution reasoning strategies* that nevertheless *change and fail under OOD shifts*: a subtle failure distinct from incapable base models.

(ii) A principled intervention on reasoning strategies significantly enhances OOD generalization, which reveals the pivotal role of reasoning strategies in the transferable improvements of RLVR.

(iii) The theoretical analysis explaining how more effective and generalizable reasoning strategies can be incentivized naturally via RLVR from a *information per reasoning step* perspective.

(iv) Reasoning trace inspection and probing of the model's hidden states qualitatively reveal how reasoning strategies are principled aspects to understand RLVR generalization.

Considering the importance of RLVR generalization, we hope this framework has provided a comprehensive yet nuanced understanding of LLM reasoning and generalization through synthetic and

controllable data, with the same spirit as prior work that has inspired us (Allen-Zhu & Li, 2025a; Ye et al., 2025a; Wang et al., 2024; Yin et al., 2025; Yuan et al., 2026; Zhang et al., 2025a).

2 Related Work

RLVR Generalization. RLVR has produced a growing set of positive generalization results: such as general reasoning and agentic capabilities enhancement after math/coding-oriented RL (Shao et al., 2024; Guo et al., 2025; Kimi Team et al., 2025), carefully structured RL expands benchmark coverage (Liu et al., 2025a; Sun et al., 2025; 2026), synthetic strings showing compositional expansion (Yuan et al., 2026), and game/puzzles improve to reasoning benchmarks (Liu et al., 2026a; Zhao et al., 2025a; Xie et al., 2026; Tong et al., 2026). Meanwhile, some studies argue that RLVR gains can also be re-distribution of the base model behaviors or spurious observations (Yue et al., 2025; Wu et al., 2025; Liu et al., 2025b; Zhao et al., 2025b; Gandhi et al., 2025; Shao et al., 2025; Wu et al., 2026b;a). Our work, however, is complementary to both sides of this debate. Instead, we show a strategy-level and nuanced inspection of *how* RLVR impacts the reasoning strategies of LLMs and the information effectiveness for a strategy to generalize to harder OOD domains. Specifically, a most crucial discovery is that RLVR can teach a model to solve the in-distribution task with the intended reasoning strategy, yet that strategy can collapse into a brittle alternative on OOD domains, which is different from either incapable-base-model failures or reward-hacking artifacts.

RLVR Algorithms. Representative LLM post-training algorithms include PPO-style RLHF (Schulman et al., 2017; Ouyang et al., 2022), preference optimization like DPO (Rafailov et al., 2023), and outcome-based methods like GRPO (Shao et al., 2024). With the effectiveness of GRPO for math and coding, recent work has extended it through DAPO, Dr. GRPO, GSPO, and other variants or recipes that address MoE stability, length or token-level bias, sampling efficiency, etc (Yu et al., 2025; Liu et al., 2025b; Zheng et al., 2025; Hu et al., 2025; Zhu et al., 2025; Wen et al., 2026; Liu et al., 2026b; Zhang et al., 2025b). We mainly use the basic DAPO (Yu et al., 2025), as our focus is the underlying meta-level reasoning behavior and generalization.

Investigating LLMs via Controllable and Synthetic Data. Synthetic data is a crucial tool to analyze the LLM behaviors reliably, especially represented by the physics of LLMs (Allen-Zhu & Li, 2025a; Ye et al., 2025a;b; Allen-Zhu & Li, 2024; 2025b;c). In LLM reasoning, synthetic data has become essential for understanding generalization, since they provide clear boundaries: symbolic or length-controlled math variants (Mirzadeh et al., 2025; Zhou et al., 2025; Liu et al., 2023), synthesized arithmetic probes (Ye et al., 2025a; Wang et al., 2024; Yin et al., 2025), and controlled agentic dynamics (Wang et al., 2025; 2026). In addition, some studies also use causal interventions, circuit analyses, and probing to identify internal algorithms or encoded information (Olsson et al., 2022; Meng et al., 2022; Voita & Titov, 2020; Gurnee & Tegmark, 2024). The work that inspires this work most is Yuan et al. (2026), which understands how meta-level reasoning skills like *composing* could be generalized in RL. We adopt this controlled-data philosophy but use it for a different purpose: rather than using synthetic tasks only to avoid contamination or measure extrapolation, our inversion framework makes the model's *reasoning strategy* observable and can reveal the nuances of RLVR generalization. With our synthetic framework, we uncover the collapse of RLVR in OOD scenarios, demonstrate principled interventions, and propose theoretical analysis.

3 Research Framework

3.1 Task Format and Principles

Principles. Our investigation centers on *which reasoning strategy* RLVR incentivizes, and whether the learned strategy generalizes from in-distribution training to OOD scenarios or quietly collapses. We therefore propose a research framework that surfaces the model's reasoning process and admits a controlled distribution shift between training and evaluation. The principles are: **(1)** *Clean OOD Generalization.* We adopt synthetic tasks following prior controlled LLM studies (Yuan et al., 2026; Sun et al., 2026; Chu et al., 2025; Wang et al., 2025; Ye et al., 2025a), enabling a clean separation between the training and testing distributions for studying OOD generalization. **(2)** *Explicit Reasoning Strategies.* To diagnose reasoning strategies, the tasks must surface a small set of clearly identifiable strategies. The open-ended flexibility of real-world math or coding tasks works against this need for clean strategy categorization. **(3)** *Reasoning Strategy Diversity.* While the strategies must be categorizable, the framework must also allow multiple candidate strategies, so that generalization can be studied by observing which one RLVR selects. From this aspect, the homogeneous function-simulation tasks in Yuan et al. (2026) fall short, as they only exercise the "composing" reasoning strategy and calculation of function outputs.

The Inversion Task. We instantiate these principles via an *inversion* task on synthetic string functions, which reverses the direction of Yuan et al. (2026): given the source code of a function $f(\cdot)$ and a target output y, the LLM is asked to recover one or more digit-string inputs x satisfying $f(x) = y$. This mirrors two real-world reasoning settings: solving math equations, where one reasons backward from a constraint to its solutions, and tracing program bugs, where one infers an upstream cause from an observed output. Crucially, the inversion task admits several reward-compatible strategies: analytical derivation, case analysis, and brute-force enumeration. It thus satisfies principles (2) and (3), while the next subsection addresses principle (1).

3.2 Task Design

We turn the inversion task above into a concrete training-and-evaluation suite, including (i) a library of functions with a function-level train/held-out split, (ii) OOD tests that probe the learned strategy for easy-to-hard generalization, and (iii) evaluation metrics.

Function Design for Held-out Training/Evaluation. We construct the training and testing sets to use distinct functions f, preventing the LLM from memorizing function-specific patterns. We therefore compose 24 non-bijective functions (many-to-one, so one output y has multiple valid inputs x) (full list in Sec. A.2) and provide their definitions and Python code inside the prompt of every inversion task. Following Yuan et al. (2026), we split these functions into training and evaluation sets: 18 functions are used for RL training, and 6 strictly disjoint functions are held out for evaluation. This held-out slice forms the easiest test of generalization: same prompt interface, same input/output regime, only the underlying function is unseen. It serves as the in-distribution check that RLVR has learned to perform inversion. From this baseline, we then move to the *easy-to-hard* OOD generalization tests below, which are our main probe of reasoning-strategy generalization. The training set contains 50k tasks, and each evaluation set contains 1,024 tasks.

Easy-to-hard OOD Generalization. As Fig. 1a, we design the in-domain and OOD tasks to differ along controlled axes. For the inversion task $f(x) = y$, the *in-domain* scenarios train and evaluate the LLM on outputs y of length 4–6 and require 2 candidate solutions per prompt. From this baseline, we design four easy-to-hard OOD axes: **(1)** *More Answers* raises the number of requested solutions to a random integer between 3 and 6, demanding more computation and reliability to produce enough valid solutions. **(2)** *Longer Outputs* increases the target output to length 6–10, lengthening the digit chain that must be inverted. **(3)** *Longer Outputs & More Answers* combines the two axes above. **(4)** *Length Change* represents the largest shift: our in-distribution function family is designed so that y and x share the same length, but the Length-Change functions differ in the lengths of y and x (full list in Sec. A.2). Each OOD evaluation set contains 1,024 randomly generated tasks as above.

Accuracy Scoring. We compute accuracy based on whether the predicted solutions are correct with an outcome-based verifier. Each solution is scored semantically by executing the reference function and checking whether its output matches y. Let k be the effective answer requirement and m be the number of submitted solutions; accuracy is $\text{acc} = \min(\text{correct_unique}, k)/\max(k, m)$. This rewards answers that are both distinct and correct, and penalizes over-generation reward hacking.

3.3 Training

We train Qwen2.5-7B-Instruct with GRPO (Shao et al., 2024) and adopt the improvements from DAPO (Yu et al., 2025), especially the filtering of groups with zero reward variance. This filtering is crucial both in the early stage of training, when the LLM is still developing its strategy with low accuracy, and in the late stage, when in-distribution accuracy has been high. We select this base model because our synthetic task lies on the boundary of its capabilities, requiring RL to incentivize correct reasoning strategies. Design details are in Sec. A.3. Analysis of more base models in Sec. A.4.

4 Reasoning Strategy Collapse and Generalization

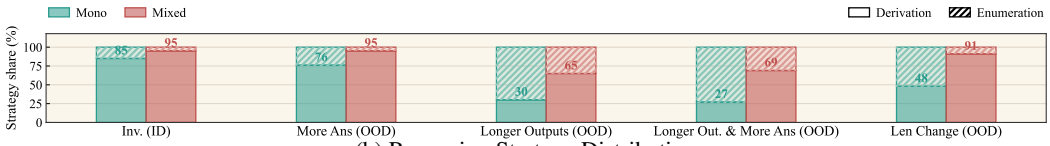
This section reveals the collapse and generalization of reasoning strategies in OOD tasks. We first show how the reasoning strategies collapse in OOD tasks despite crrect in-distribution behaviors (Sec. 4.1). Then, we introduce incentivizing a better reasoning strategy for generalization (Sec. 4.2).

4.1 Mono Model: Revealing OOD Collapse of Reasoning Strategies

Following Sec. 3, we train the LLM on the non-bijective function inversion tasks, *i.e.*, solving x given $f(x) = y$, and call this model **Mono**. By closely observing it on the in-distribution and the four easy-to-hard OOD sets, as in Fig. 2, we observe the nuanced OOD reasoning strategy collapse.



(a) Evaluation Accuracy During RL



(b) Reasoning Strategy Distribution

Figure 2: Reasoning strategy collapse and generalization in OOD tasks (Sec. 4). The MONO model performs normally ID but collapses, which relies on brute-force enumeration OOD. In comparison, MIXED model shows similar ID performance, but surprisingly doubles analytical derivation reasoning on OOD and lead to better generalization. These reveal the challenges of reasoning strategy generalization and how to enhance the generalization of reasoning in RLVR.

Reasoning Strategy Categorization. Our synthetic tasks are specially constructed to understand the RLVR reasoning strategy generalization in a controllable way, as explained in Sec. 3.1. So we start from categorizing the reasoning strategy utilized by the MONO model. As in Fig. 3, we can clearly attribute two primary strategies: *derivation*, which structurally reverses f and only conduct case-analysis when needed, and *enumeration*, which directly guess all the digits all at once, being much less efficient. To make this binary classification revealing the traits of reasoning strategies, we treat purely brute-force enumeration as "enumeration," and everything else as "derivation." In practice, we employ an LLM judge empowered by Gemini-3.0-Flash to classify the reasoning traces (clarification of our setup, details, and verification of LLM judge in Sec. B.1).

Structural Derivation

… Case 1: d4 = d2 – 4 ...
So, we have:
d1 = d2 + 4, d3 = d2, d4 = d2 - 4
d5 = d2 – 1, d6 = d2 + 3
…
d2 = 5 -> Solution is 955148
d2 = 4 -> Solution is 844037

Brute-force Enumeration

Let's try some values:
Assume a0 = 5, a1 = 1, a2 = 1,
a3 = 5, a4 = 9, a5 = 4. Let's check …
This set of values does not work…

2. Assume a0 = 9, a1 = 5, a2 = 5,
a3 = 9, a4 = 6, a5 = 8. …

Figure 3: Examples of reasoning strategies.

From ID to OOD Collapse. Fig. 2a tracks accuracy during RL on the ID set and the four OOD sets, and Fig. 2b shows which strategies produce these responses. On in-distribution tasks, the MONO model learns the effective strategy of derivation in most cases and achieves high accuracy (as Fig. 2, ID columns). However, its failure in OOD scenarios, *e.g.*, the novel functions of lenchange, is not solely due to longer reasoning or more computation. Rather, its reasoning skill of derivation acquired disappears on OOD tasks and gives way to the inefficient brute-force enumeration, which proposes all the digits of x simultaneously, without utilizing the structure of the function f. The majority of failures are caused by loops that enumerate all possible digits until reaching the token limits. Here, *collapse* refers to the reasoning strategy: MONO's derivation share drops from 85% on ID to below 50% on the three harder OOD sets. Accuracy, in contrast, degrades progressively as the shift grows: it stays above 80% on More Answers, the mildest shift, and drops to 6% on Length Change.

Such an observation is surprising as it challenges RLVR generalization: *the reasoning strategy already incentivized in RL fails to generalize despite identical task formats*. As mentioned in Sec. 1 and Sec. 3, this observation focuses on the meta-level reasoning skills of LLMs and is subtle and distinct from prior work. First, it is not a *base-model* limitation: the same model reaches $\approx 90\%$ on in-distribution non-bijective inversion. Second, it is not *reward hacking*: the LLM uses the correct and efficient derivation strategy for in-distribution evaluation rather than spurious shortcuts. We refer to this phenomenon as *reasoning-strategy collapse*, which motivates our central question: which training signal enables RLVR to generalize a reasoning pattern to OOD scenarios?

TAKEAWAY 1

Reasoning Strategy Collapse: Even though RLVR has incentivized the correct reasoning strategies for in-distribution tasks, the LLM might drastically change to ineffective strategies on OOD tasks despite identical task formats, undermining the generalization of RLVR.

4.2 MIXED MODEL: ENHANCING OOD GENERALIZATION VIA BIJECTIVE INVERSION

The *derivation* reasoning strategy, avoiding pure enumeration, is thus our key to understanding RLVR generalization. Our intervention, therefore, is not to propose training tasks that are *closer* to the evaluation or simply *harder* than the existing training tasks, but a set of new tasks that more directly incentivize the derivation strategy. In fact, this is also why RLVR could bring improvements to multiple reasoning domains for LLMs in previous works (Liu et al., 2026a; Yuan et al., 2026). Specifically, we propose using bijective functions as f (both injective and surjective) to eliminate uncertainty in non-bijective tasks and encourage derivation via larger certainty.

Enhancing OOD Generalization. Concretely, we co-train on bijective inversion with a 1:1 with the original non-bijective tasks, so this model is called **MIXED**. As shown in Fig. 2b, this almost doubles the utilization of derivation strategy compared with the MONO model, and leads to significantly better OOD evaluation (Fig. 2a), especially on the novel functions of LenChange. Therefore, the incorporation of bijective inversion effectively improves the RLVR reasoning generalization. We have controlled the data recipe and training configuration for a fair comparison between the two models (details Sec. A.3) and additionally verified the main conclusions with multiple runs in Sec. B.2.4.

Why Unexpected? This is a surprising observation because it contradicts three natural intuitions: (1) bijective tasks ask for a single answer, so they are *further* from the multi-answer evaluation; (2) they are *not clearly harder* by the training reward or by intrinsic difficulty (details in Sec. B.2); and (3) they do *not* change the in-distribution accuracy notably (Fig. 2). These further suggest that the meta-level reasoning strategies are the essential element for RLVR generalization.

For scientific rigor, we have conducted additional experiments in Sec. B.2 to eliminate the confounding factors other than reasoning strategies. Specifically, our controllable experiments show that the improvement is not primarily explained by: (1) a special set of evaluation; (2) increasing the diversity of training tasks; and (3) increasing the difficulty of training tasks. Such analysis further motivates us to *understand* how the reasoning strategies that generalize better are incentivized.

TAKEAWAY 2

Incorporating tasks that are even further from the OOD evaluation, *i.e.*, bijective inversion, enhances generalization via encouraging the more effective reasoning strategy.

5 BITS PER TOKEN SHAPES REASONING STRATEGIES AND GENERALIZATION

With Sec. 4 revealing reasoning strategies as the central driver of RLVR generalization, this section answers two questions: **(1)** *what* makes a reasoning strategy effective, *e.g.*, derivation over enumeration, which we frame as *bits of information generated per token* (Sec. 5.1); and **(2)** *how* an outcome-only RLVR reward implicitly supervises the reasoning *process* and selects effective strategies (Sec. 5.2). Sec. 5.3 then verifies this mechanism with two interventions.

5.1 REASONING STRATEGIES VIEWED FROM BITS PER TOKEN

The Search MDP. We view a reasoning problem as searching the solution space $\mathcal{Y}$ for a small subset $\mathcal{Y}^\star \subseteq \mathcal{Y}$ of correct solutions. A strategy, *e.g.*, derivation in our inversion task or reliable calculation in math, is effective to the extent that each emitted token moves towards $\mathcal{Y}^\star$. We cast this as an MDP: a state $s \in \mathcal{S}$ is the prompt plus the tokens emitted so far; an action $a \in \mathcal{A}$ is the next token; transitions $s_{t+1} = f(s_t, a_t)$ are deterministic; trajectories $\tau = (s_0, a_0, \ldots, s_T)$ have bounded length T and terminate at final solutions $y_\tau \in \mathcal{Y}$. The reward is outcome-based, $R(\tau) = \mathbf{1}[y_\tau \in \mathcal{Y}^\star]$. Identifying a correct solution from $\mathcal{Y}$ thus carries $D := \log(|\mathcal{Y}|/|\mathcal{Y}^\star|)$ bits of information.

Reasoning Strategy and Generalization. Strategy effectiveness thus becomes these D bits. If a policy creates at most $\log b$ bits per action (*e.g.*, choosing one number from b candidates), we would expect $T \log b \geq D$. Effective reasoning is therefore a higher *bit-rate* towards $\mathcal{Y}^\star$. *Derivation* contributes more bits per step by pruning candidates structurally, so it generalizes better to harder problems that require more bits compared with *Enumeration* rejecting one candidate at a time.

5.2 OUTCOME-BASED REWARD INCENTIVIZES REASONING PROCESS VIA BIT RATES

The bit-rate view characterizes *which* strategies are effective. We now show how outcome-only RLVR *incentivizes* them, first with an actor-critic (Proposition 1) and then with GRPO (Proposition 2).

Connection with RL Advantage. For a policy π, the success probability $p^\pi(s) := \mathbb{P}_{\tau\sim\pi}[y_\tau \in \mathcal{Y}^\star \mid s_t = s]$ equals the value function, $V^\pi(s) = p^\pi(s)$, and $p^\pi(s_0)$ is the prompt's success rate. The

change $\Delta p_t := p^\pi(s_{t+1}) - p^\pi(s_t)$ records the progress of one token: intuitively, how many candidates it prunes (Fig. 1b). Information-theoretically, $-\log p^\pi(s)$ counts the bits still owed at s, so one token yields $\log\big(p^\pi(s_{t+1})/p^\pi(s_t)\big)$ bits, of which Δp_t is the first-order linearization. As the reward is success itself, Δp_t also equals the step-level advantage, $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) = \Delta p_t$ (Proposition 1). Thus, a strategy with more information per token, *i.e.*, larger Δp_t, is credited more.

Proposition 1 (Actor-critic) *Under terminal reward and deterministic transitions as Sec. 5.1, the advantage at every step equals the success increment, $A^\pi(s_t, a_t) = \Delta p_t$, and therefore*

$$\nabla_\theta J(\pi_\theta) \;=\; \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \Delta p_t \, \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right].$$

The proof (Sec. C.1) combines $R(\tau) - p^\pi(s_0) = \sum_t \Delta p_t$ with the value baseline $V^\pi(s_t) = p^\pi(s_t)$. Policy gradients thus credit each token by the bits it produces.

Proposition 2 (GRPO) *GRPO sees only group-level rewards $R^{(i)} = R(\tau^{(i)})$ for G rollouts and assigns the trajectory-level advantage $\hat{A}^{(i)} = (R^{(i)} - \bar{R})/\hat{\sigma}_R$ to every token, giving the gradient estimator $\hat{g}_{GRPO} := \frac{1}{G}\sum_{i=1}^{G} \hat{A}^{(i)} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})$. With large group G,*

$$\mathbb{E}[\hat{g}_{GRPO}] \;\approx\; \frac{1}{\sigma(s_0)} \nabla_\theta J(\pi_\theta), \qquad \textit{where} \quad \sigma(s_0) := \sqrt{p^\pi(s_0)(1 - p^\pi(s_0))}.$$

This matters because GRPO has no critic, so the trajectory-level $\hat{A}^{(i)}$ is its only signal. The proof (Sec. C.2) treats $\bar{R}$ as a Monte Carlo estimator of $p^\pi(s_0)$ and, by conditioning, recovers the same $\sum_t \Delta p_t \nabla \log \pi$ as Proposition 1 up to a rescaling $1/\sigma(s_0)$. Step-level credit by bits per token therefore emerges from group statistics alone.

This explains why bijective functions incentivize derivation better (Sec. 4.2): $|\mathcal{Y}^\star| = 1$ sharpens the per-step advantage gap between high- and low-bit-rate strategies. Compared with non-bijective tasks, bijective tasks thus give up uncertainty but separate derivation from enumeration more clearly.

5.3 Verification Experiments

The bit-rate view $D/(T \log b)$ predicts that two more interventions should enhance derivation. **(1)** Shorter T: we cut the rollout budget from 8k to 5k tokens to force a higher bit-rate per step. **(2)** Constrained solution space: we add digit-level constraints to each prompt (*e.g.*, "the 1st digit cannot be 1, 3, 4, 7"), which also increases D. All other settings match Mono in Sec. 4.1, and the short budget changes only the maximum response length (Sec. A.3). Note that both interventions make the problems harder, while our bijective tasks do not necessarily do so. Fig. 4 shows that both runs raise the derivation rate, validating the link between bit-rate task structure and the strategies RLVR incentivizes (Sec. 5.2). Full results and accuracy curves are in Sec. C.3.

Takeaway 3

Bits produced per token distinguish effective reasoning strategies for OOD generalization, and also power outcome-only RLVR's incentivization of the reasoning process.

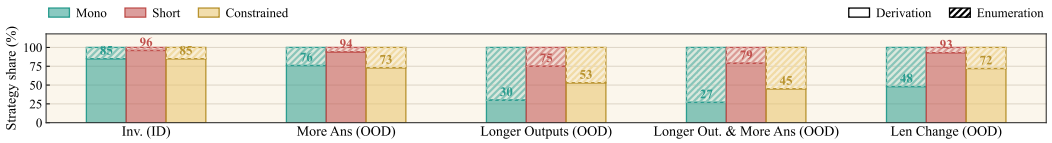


Figure 4: Tightening bit rates incentivizes the more generalizable derivation strategy. Notation follows Fig. 2b; "short" and "constrained" are the shorter T and constrained solution space in Sec. 5.3.

6 Behavioral Analysis

This section conducts a broader inspection of our previous insights into reasoning strategies and OOD generalization. The analyses highlight how the generalization and reasoning strategies could be extended to understand more properties of LLMs.

6.1 Representation Analysis for Strategy Commitment

The first question is whether the strategy shift in Sec. 4 is visible inside the model's hidden states. Following the probing perspective of Ye et al. (2025a), we ask *when* the hidden states could reveal the final reasoning strategies reflected in reasoning traces, *i.e.*, derivation or enumeration.

Probe setup. For both MONO and MIXED models, we extract hidden states along each generated chain-of-thoughts on OOD sets and train a linear classifier to predict the final strategy label annotated as Fig. B.1. In addition, we find that the traces share a stable response structure: restating the problem, reading the function, and then executing a function (see example and details in Sec. D). We therefore probe the N-th token after `<think>` for $N \in \{0, 100, 200, 300, 500, 1000, 1500\}$. To avoid shortcuts that only memorize function names or evaluation sets, we report balanced accuracy and compare against a strong null condition and utilizes the **p-values** to indicate whether the tokens have embedded strategies significantly. Notably, the null shuffles strategy labels only within each function $\times$ evaluation-set subset, so that the shortcut of both functions and evaluatation sets are ruled out (details in Sec. D) for statistical rigor.

Result. Fig. 5 shows two interesting patterns. (1) MONO's strategy is already significantly decodable at N=0 ($p < .001$), before the generated reasoning has analyzed the function. This matches the collapse in Sec. 4.1: the model might enter OOD problems with a preselected behavior rather than a problem-conditioned strategy. (2) MIXED becomes significantly decodable only after the function-analysis region. This suggests that MIXED might commit to a strategy later than MONO and use more task contexts, which is consistent with the observation that MIXED's reasoning strategy is more robust to OOD shifts. After the function analysis, the two models become similarly decodable, as their traces start to execute the chosen strategy.

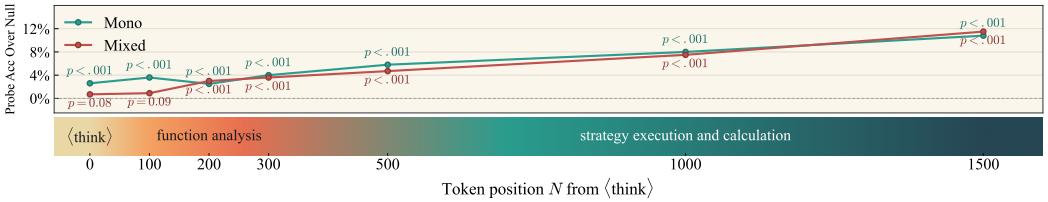


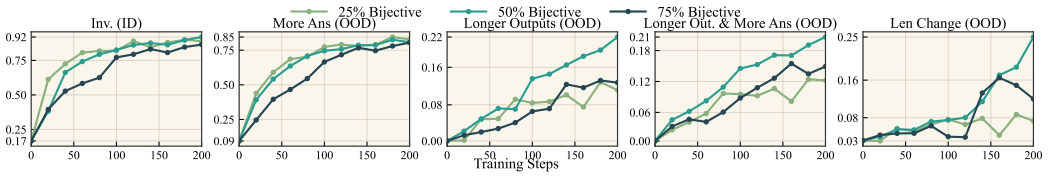
Figure 5: Probing reasoning strategies from hidden states. The y-axis is the delta over a strong null condition (*i.e.*, 76% for the MIXED model), and we use p-value to read the significance of probing.

6.2 MIXTURE OF TASKS

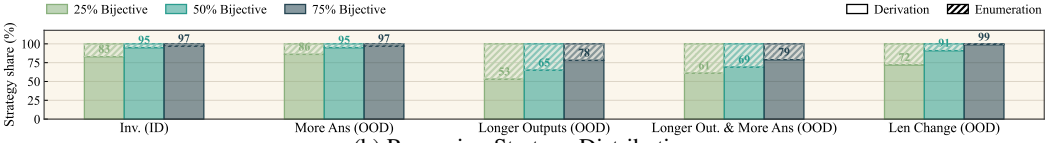
The second question is how much bijective data is useful. This ablation tests both sides of the main paper: Sec. 5 predicts that bijective tasks should favor higher-bit-rate derivation, while Sec. 4 leaves open whether more derivation always means better accuracy. We therefore train additional mixtures with bijective:non-bijective ratios 1:3 and 3:1, alongside the original 1:1 MIXED model.

Strategy Distribution. Fig. 6b shows a monotonic effect: increasing the bijective fraction increases the derivation rate. This is the behavior predicted by Sec. 5: RLVR assigns more credit to tokens that structurally shrink the solution space, and this is incentivized by the bijective tasks. The mixture ratio therefore acts as a continuous control over the model's reasoning strategies.

Accuracy boundary. Fig. 6a adds the necessary caveat: the 3:1 mixture uses derivation most often, but the 1:1 mixture is most accurate. Auditing the errors (Sec. E) shows that, despite bijective data leading the model to the correct meta-level strategy, the models are prone to execution errors, such as invalid modular arithmetic. The reason is that bijective tasks ask for only one solution, so more bijective data leaves less practice on producing several distinct solutions. This provides additional



(a) Validation Accuracy



(b) Reasoning Strategy Distribution

Figure 6: Analyzing the recipe of mixtures. (Details in Sec. 6.2)

context for LLM generalization beyond the reasoning-strategy aspect in RLVR generalization: a generalizing model also needs certain task-specific basic skills that make the selected strategy reliable.

6.3 Analysis of Zero-shot Transfer to Math

Synthetic RLVR studies sometimes use math benchmarks to hint generalization. However, our goal is not to claim that our synthetic tasks are effective for math, but to test whether the factors identified above, the meta-level strategies and execution skills, also explain behaviors for real-world questions.

Table 1 reports zero-shot results (details in Sec. F.1). Both RLVR models improve over the base model on average, but they specialize differently. MONO is stronger on MATH500,better at sustained mechanical execution. MIXED is stronger on the other five harder benchmarks and shows better structural reasoning.

Table 1: Zero-shot transfer to math benchmark: MATH500 (Hendrycks et al., 2021), Minerva-Math (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024). Accuracy in %.

Benchmark	Qwen	MONO	MIXED
MATH500	76.0	**77.2**	75.8
Minerva-Math	36.0	34.9	**37.5**
AIME24	10.0	10.0	**13.3**
AIME25	3.3	0.0	**10.0**
AMC23	47.5	55.0	**62.5**
OlympiadBench	37.6	38.1	**39.9**
Average	35.1	35.9	**39.8**

Trace Analysis. We audit the disagreement of the models. MIXED more often preserves the problem structure better, for example, keeping a full divisor set, respecting ordered constraints, or parsing a custom operator correctly. This coincides with the problem-conditioned strategy commitment in Sec. 6.1. MONO, by contrast, is more reliable when the bottleneck is long mechanical execution, such as polynomial division or checking alternate cases. Secs. F.2 and F.3 provides concrete examples. These audits connect the math benchmark numbers back to our central focus on reasoning strategies: MIXED transfers a better meta-level reasoning strategy for the task structures, while MONO retains stronger execution. This suggests why synthetic RL can improve math in prior work (Yuan et al., 2026; Liu et al., 2026a): it transfers a meta-level preference for structural reasoning, while domain-specific execution still needs in-domain training.

6.4 Agentic Transfer

Finally, we ask whether the same principle we used to incentivize higher bit-rate strategies can understand scenarios outside token-level inversion traces. We use Sokoban as a controllable multi-step agentic environment, following RAGEN (Wang et al., 2025), and train with DAPO (Yu et al., 2025) (details in Sec. G). To create an analogy of the high bit-rate bijective tasks in Sec. 4.2, we add a Sokoban variant that succeeds only if the rollout reaches the goal within *optimal_steps + 2* actions. We mix this variant with the original environment at 1:1. To avoid turning the auxiliary task into a simple difficulty increase, we reduce its grid size by one.

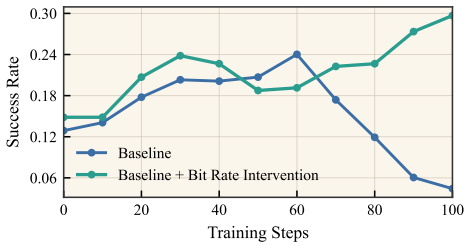


Figure 7: Our intervention of adding tasks with larger bits rate (Sec. 4 and Sec. 5) can also help multi-step scenarios.

Fig. 7 shows that the baseline run degrades, similar to the instability observed in RAGEN (Wang et al., 2025). The mixed run with the auxiliary tasks, however, trains better. Even so, we acknowledge that the multi-step scenario is not a direct analogy with our investigation, as there is no clear categorization of derivation and enumeration in this setting. However, this experiment hints at potential future work: the principle of reasoning strategies and bits per token, though discussed primarily for reasoning tasks in this paper, may also improve RLVR in agentic settings.

7 Conclusion

This paper investigates reasoning strategies as a crucial factor for RLVR generalization. On a controllable framework of non-bijective string equations with clearly defined OOD generalization, RLVR teaches the derivation strategy in-distribution, yet the model falls back to ineffective enumeration on OOD tasks. Co-training with bijective tasks restores derivation and enhances OOD generalization, which motivates our theory: effective reasoning strategies contribute more bits per token and are implicitly rewarded by RLVR. Our broader analyses further support the role of reasoning strategies in LLM generalization, and we discuss limitations in Sec. H. We hope our study can inspire RL recipes that improve generalization by controlling reasoning strategies.

AI USE STATEMENT

We use Gemini-3.0-Flash as an LLM judge to label reasoning strategies (Sec. 4.1), and validate it against manual annotation (Sec. B.1). We use generative AI tools for editing the writing, *e.g.*, grammar, spelling, and word choice. We have reviewed all AI-assisted content and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Sec. A.1 gives the prompt format and the verifier, Sec. A.2 lists every function used for training and evaluation, and Sec. A.3 lists all RL hyperparameters and hardware. The strategy judge prompt is in Fig. B.1, and the probing protocol is in Sec. D. Sec. B.2.4 reports repeated runs with different seeds and data.

REFERENCES

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. In *ICML*, 2024. 3

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 1, learning hierarchical language structures. *TMLR*, 2025a. 3

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation. In *ICLR*, 2025b. 3

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. In *ICLR*, 2025c. 3

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In *ICML*, 2025. 3

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STaRs. In *COLM*, 2025. 3

Zhibin Gou and contributors. Math-evaluation-harness: A unified toolkit for mathematical reasoning evaluation. GitHub, 2023. 24

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 16

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 2025. 3

Wes Gurnee and Max Tegmark. Language models represent space and time. In *ICLR*, 2024. 3

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *ACL*, 2024. 9, 24

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. In *NeurIPS Datasets and Benchmarks Track*, 2021. 9, 24

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. In *NeurIPS*, 2025. 3

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1.5: Scaling reinforcement learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025. 3

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *NeurIPS*, 2022. 9, 24

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. In *ICLR*, 2024. 24

Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. TinyGSM: achieving >80% on GSM8k with small language models. *arXiv preprint arXiv:2312.09241*, 2023. 3

Bo Liu, Simon Yu, Zichen Liu, Leon Guertler, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, Wee Sun Lee, and Natasha Jaques. SPIRAL: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. In *ICLR*, 2026a. 1, 3, 6, 9

Fanfan Liu, Youyang Yin, Peng Shi, Siqi Yang, Zhixiong Zeng, and Haibo Qiu. Length-unbiased sequence policy optimization: Revealing and controlling response length variation in RLVR. *arXiv preprint arXiv:2602.05261*, 2026b. 3

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. ProRL: Prolonged reinforcement learning expands reasoning boundaries in large language models. In *NeurIPS*, 2025a. 3

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. In *COLM*, 2025b. 3

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *NeurIPS*, 2022. 3

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *ICLR*, 2025. 3

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022. 3

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022. 3

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2023. 3

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, et al. Spurious rewards: Rethinking training signals in rlvr. *arXiv preprint arXiv:2506.10947*, 2025. 3

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. 3, 4, 15

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient RLHF framework. In *EuroSys*, 2025. 15

Yiyou Sun, Shawn Hu, Georgia Zhou, Ken Zheng, Hanna Hajishirzi, Nouha Dziri, and Dawn Song. OMEGA: Can LLMs reason outside the box in math? evaluating exploratory, compositional, and transformative generalization. *arXiv preprint arXiv:2506.18880*, 2025. 3

Yiyou Sun, Yuhan Cao, Pohao Huang, Haoyue Bai, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. RL grokking recipe: How does RL unlock and transfer new algorithms in LLMs? In *ICLR*, 2026. 1, 3

Jingqi Tong, Jixin Tang, Hangcheng Li, Yurong Mou, Ming Zhang, Jun Zhao, Yanbo Wen, Fan Song, Jiahao Zhan, Yuyang Lu, et al. Game-RL: Synthesizing multimodal verifiable game data to boost VLMs' general reasoning. In *ICLR*, 2026. 3

Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. In *EMNLP*, 2020. 3

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. In *NeurIPS*, 2024. 3

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025. 3, 9, 15, 25

Zihan Wang, Chi Gui, Xing Jin, Qineng Wang, Licheng Liu, Kangrui Wang, Shiqi Chen, Linjie Li, Zhengyuan Yang, Pingyue Zhang, et al. Ragen-2: Reasoning collapse in agentic rl. *arXiv preprint arXiv:2604.06268*, 2026. 3, 25

Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*, 2024. 16

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base LLMs. In *ICLR*, 2026. 3

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. 20

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why RLVR may or may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025. 3

Haoze Wu, Cheng Wang, Wenshuo Zhao, and Junxian He. Mirage or method? how model-task alignment induces divergent RL conclusions. In *ICLR*, 2026a. 3

Mingqi Wu, Zhihao Zhang, Qiaole Dong, Zhiheng Xi, Jun Zhao, Senjie Jin, Xiaoran Fan, Yuhao Zhou, Huijie Lv, Ming Zhang, Yanwei Fu, Qin Liu, Songyang Zhang, and Qi Zhang. Reasoning or memorization? unreliable results of reinforcement learning due to data contamination. In *AAAI*, 2026b. 3

Yunfei Xie, Yinsong Ma, Shiyi Lan, Alan Yuille, Junfei Xiao, and Chen Wei. Play to generalize: Learning to reason through game play. In *ICLR*, 2026. 3

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 16

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. In *ICLR*, 2025a. 3, 7, 15

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.2, how to learn from mistakes on grade-school math problems. In *ICLR*, 2025b. 3

Fangcong Yin, Zeyu Leo Liu, Liu Leqi, Xi Ye, and Greg Durrett. Learning composable chains-of-thought. *arXiv preprint arXiv:2505.22635*, 2025. 3

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *NeurIPS*, 2025. 3, 4, 9, 15, 25

Lifan Yuan, Weize Chen, Yuchen Zhang, Ganqu Cui, Hanbin Wang, Ziming You, Ning Ding, Zhiyuan Liu, Maosong Sun, and Hao Peng. From $f(x)$ and $g(x)$ to $f(g(x))$: LLMs learn new skills in RL by composing old ones. In *ICLR*, 2026. 1, 3, 4, 6, 9, 14, 15

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *NeurIPS*, 2025. 1, 3

Charlie Zhang, Graham Neubig, and Xiang Yue. On the interplay of pre-training, mid-training, and RL on reasoning language models. *arXiv preprint arXiv:2512.07783*, 2025a. 3

Yuheng Zhang, Wenlin Yao, Changlong Yu, Yao Liu, Qingyu Yin, Bing Yin, Hyokun Yun, and Lihong Li. Improving sampling efficiency in RLVR through adaptive rollout and response reuse. *arXiv preprint arXiv:2509.25808*, 2025b. 3

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. In *NeurIPS*, 2025a. 3

Rosie Zhao, Alexandru Meterez, Sham Kakade, Cengiz Pehlevan, Samy Jelassi, and Eran Malach. Echo chamber: RL post-training amplifies behaviors learned in pretraining. In *COLM*, 2025b. 3

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025. 3

Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. GSM-infinite: How do your LLMs behave over infinitely increasing context length and reasoning complexity? *arXiv preprint arXiv:2502.05252*, 2025. 3

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in LLM reasoning. In *NeurIPS*, 2025. 3

A Details of Research Framework

A.1 Prompt Format and Scoring

All main experiments use a shared format that is similar to Yuan et al. (2026), with the modifications to inversion prompts. Each prompt contains the full source code for a function, a target output string y, and an instruction to return k different digit-string inputs whose execution produces y. The model must rely on its own calculation and analysis. Fig. A.1 shows a representative prompt drawn from the length-changing OOD evaluation set.

```
You are given a code:

from math import gcd

def func(s: str) -> str:
    """Reduce length by 1: first+last mod 5 becomes new first; drop last.

    Example structure (requested): mix first/last, then remove last digit.
      y[0]   = (x[0] + x[n-1]) % 5
      y[1:]  = x[1:n-1] % 5   (per-digit mod 5 view)
    """
    n = len(s)
    if n <= 1:
        return ""
    x0 = ord(s[0]) - ord("0")
    xl = ord(s[-1]) - ord("0")
    out = [str((x0 + xl) % 5)]
    for ch in s[1:-1]:
        out.append(str((ord(ch) - ord("0")) % 5))
    return "".join(out)


def main_solution(x):
    return func(x)

You are told that `main_solution(x)` returns the string "424331344".
Find 3 DIFFERENT digit-string inputs `x` such that `main_solution(x)` returns exactly
     that string.

Please reason between the XML tags of <think> </think>, and put the answers, which are
     ONLY the digit strings, between <answer> </answer> tags separated by commas.
The final response should look like <think> ... </think><answer> x1,x2,... </answer>
```

Figure A.1: Example prompt for training and evaluation.

The answer is parsed from `<answer>` tags as a comma-separated list of digit strings. If fewer than k valid preimages exist, the prompt instructs the model to output only the valid ones. The verifier executes the reference implementation on each submitted candidate rather than comparing against a fixed answer set. This is necessary for non-bijective functions, where the generator may know several valid preimages but need not enumerate all of them. In our evaluation function, we also explicitly conduct the de-duplication of the solutions from the LLMs, to avoid any form of reward hacking.

A.2 Function Lists

We use three function families. First, the non-bijective functions are the core inversion tasks introduced in Sec. 3.2 and used to train Mono in Sec. 4.1. For convenience, we use the canonical names `nb_00`–`nb_23`, with 18 training functions and 6 held-out functions. Second, the bijective functions are the stricter auxiliary tasks introduced in Sec. 4.2. They use the names `func_00`–`func_23`, with 18 training functions and 6 held-out functions. Third, the length-changing OOD functions are introduced as the Length Change evaluation axis in Sec. 3.2. They use the names `lenop_00`–`lenop_05`. We intentionally make the non-bijective held-out split easier to further support our observed OOD reasoning strategy collapse: the training and held-out functions share primitives such as cyclic neighbor operations, prefix/state updates, and modular or folded emissions. In the prompts, the displayed function is wrapped as `func`; the names below are dataset identifiers.

Non-bijective train functions. `nb_00`: cyclic floor-average of neighbors. `nb_01`: cyclic ceil-average of neighbors. `nb_03`: cyclic capped sum. `nb_04`: cyclic overflow amount past 9. `nb_05`: cyclic distance of neighbor-sum to 9. `nb_07`: cyclic folded neighbor-difference mod 10. `nb_08`: cyclic average plus absolute difference mod 10. `nb_10`: cyclic triple average floor. `nb_11`: cyclic

triple average ceil. `nb_13`: cyclic folded triple sum mod 10. `nb_15`: cyclic clamped absolute value of $a + b - c$. `nb_16`: prefix sum to evenized emission. `nb_17`: prefix sum to folded emission. `nb_19`: affine state update plus folded emission. `nb_20`: cyclic median of three. `nb_21`: cyclic rank of the first element in a triple. `nb_22`: cyclic capped neighbor product. `nb_23`: cyclic local total variation.

Non-bijective held-out functions. `nb_02`: cyclic absolute difference. `nb_06`: cyclic folded neighbor-sum mod 10. `nb_09`: cyclic two-view merge. `nb_12`: cyclic triple range. `nb_14`: cyclic second-difference magnitude mod 10. `nb_18`: prefix sum to square emission.

Bijective train functions. `func_00`: reverse, then add 1 to every digit. `func_01`: rotate left by 1, then add 3 to every digit. `func_02`: swap first and last characters, then add 3. `func_03`: swap adjacent pairs, then add 1. `func_04`: swap the two halves, then add 2. `func_05`: reverse each half, then add 3. `func_06`: move odd-indexed characters to the front, then add 1. `func_07`: interleave the two halves, then add 2. `func_08`: reverse the inner substring, then add 3. `func_09`: rotate each full group of three right by 1, then add 1. `func_10`: reverse, then add 1 to even positions and 2 to odd positions. `func_11`: rotate left by 1, then add 1 or 3 on alternating positions. `func_12`: add $i + 1$ to the digit at position i. `func_13`: add $2i + 1$ to the digit at position i. `func_14`: add 1 on the left half, 3 on the right half, and 2 in the middle. `func_15`: add the repeating pattern 1,2,3. `func_16`: reverse, then add $i + 1$ at position i. `func_17`: swap adjacent pairs, then add $i + 2$ at position i.

Bijective held-out functions. `func_18`: move odd-indexed characters to the front, then apply half-specific shifts. `func_19`: swap halves, then add the repeating pattern 1,2,3. `func_20`: running sum with $+1$ drift. `func_21`: running sum with $+2$ drift. `func_22`: running sum with alternating $+1/+2$ drift. `func_23`: right-to-left running sum with $+1$ drift.

Length-changing OOD functions. `lenop_00`: reduce length by one via non-cyclic neighbor-sum mod 5. `lenop_01`: reduce length by one via non-cyclic signed neighbor mix mod 5. `lenop_02`: reduce length by one by emitting first-plus-last mod 5, then dropping the last digit. `lenop_03`: increase length by one via cyclic neighbor-sum mod 5 plus a checksum digit. `lenop_04`: increase length by one via a mod-5 affine state machine plus a final-state digit. `lenop_05`: increase length by one via mod-5 prefix sums plus a start/end consistency check.

A.3 Training Details

Implementation. We run RL experiments with VERL/HybridFlow (Sheng et al., 2025), using the GRPO objective (Shao et al., 2024) with the DAPO-style filtering of zero-reward-variance groups (Yu et al., 2025). The policy is initialized from Qwen2.5-7B-Instruct; rollouts are generated with `vllm`, and actor/reference models are sharded with FSDP. Training is conducted on one node with 8 NVIDIA H100 80GB GPUs, approximately 1 day.

Reward. We follow common RLVR practice and construct the reward by 0.1 of format reward, including the thinking format and number of solutions, and 0.9 of the accuracy scoring.

Core hyperparameters. Unless otherwise stated, we train for 200 update steps with a prompt batch size of 16, one PPO epoch per update, actor learning rate 10^{-6}, clip ratio 0.2, entropy coefficient 10^{-3}, and gradient clipping at 1.0. We use GRPO groups of 16 sampled rollouts per prompt, with sampling temperature 1.0, top-p of 1.0, and no top-k truncation. The maximum prompt length is 1,024 tokens and the maximum response length is 8,192 tokens; the short-budget intervention in Sec. 5.3 changes only this response budget to 5k tokens. We do not use a learned reward model or KL penalty in these controlled experiments; rewards are computed by the verifier in Sec. A.1. Validation is run every 20 steps with greedy decoding (`temperature`=0, one rollout per prompt).

A.4 Selection of Base Model

This section explains why we decided to use Qwen2.5-7B-Instruct as our base model. For controlled studies of LLM post-training, the base model is part of the experimental design: if it is too weak, failures mostly measure missing task competence; if it is too strong, the task saturates before RL exposes what the reward is selecting. This is the same methodological motivation behind prior controlled reasoning studies (Ye et al., 2025a; Yuan et al., 2026; Wang et al., 2025), which choose settings where the target behavior is observable, the verifier is clean, and the training dynamics remain diagnostic. Since our goal is to reveal a nuanced RLVR generalization failure rather than to maximize

absolute accuracy, the appropriate base model is one that can solve the in-distribution task after RL but still leaves enough room for the learned reasoning strategy to vary and collapse OOD.

Additional Base Models. We tried Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen3-4B-Instruct (Yang et al., 2025) under the same inversion task setup. Llama-3.1-8B-Instruct does not provide a suitable initialization for our question: after RL on non-bijective tasks, it reaches only 27.1% on in-distribution non-bijective inversion. This performance indicates that the model falls into the category of less capable base model, has not yet acquired reliable inversion reasoning or the basic execution skills needed for our analysis. Failures in this regime would mostly reflect insufficient task competence, rather than a strategy-level OOD generalization gap. So we exclude Llama-3.1-8B-Instruct as our base model.

Qwen3-4B-Instruct has the opposite issue: it is already too strong for this controlled study. Its in-distribution non-bijective inversion accuracy reaches 97.8% after just 20 steps of RL training, indicating that our controlled study has not significantly influenced the model's behavior yet. This leaves little room for RL to reveal a meaningful strategy-acquisition process, because the model is close to saturation before the training dynamics become informative.

Conclusion. Using controlled study to investigate LLM is a co-design across the model, task, data. For our purpose, Qwen2.5-7B-Instruct is the appropriate operating point because it makes the central comparison identifiable: after RL, both MONO and MIXED solve the in-distribution non-bijective inversion task at about 91–92%, so the source-task competence, prompt understanding, and verifier-compatible answer behavior are already established. The remaining failure is therefore not that the model cannot solve the task, but that the reasoning strategy learned on the source distribution does not necessarily transfer OOD. This is exactly the phenomenon our paper aims to reveal. If we used a weaker model, the same OOD failures would be confounded with missing basic skills; if we used a saturated model, RL would provide too little behavioral change to analyze strategy selection. Making the task much longer or harder would similarly mix our target failure with arithmetic, context-length, and execution errors. Thus, Qwen2.5-7B-Instruct is a principled choice for exposing reasoning-strategy collapse under already favorable conditions for RLVR generalization.

B DETAILS OF REVEALING STRATEGY COLLAPSE AND FIXES ON OOD GENERALIZATION

B.1 REASONING STRATEGY CATEGORIZATION

Why Derivation and Enumeration. As discussed in Sec. 4.1 and illustrated in Fig. 3, our analysis separates inversion reasoning into *analytical derivation* and *brute-force enumeration*. This categorization is exhaustive for our purpose because the core decision is binary: when proposing the input digits, does the LLM rely on enumerating full digit assignments and then verify them? If most digit values are outputs of equations, we classify the trace as derivation; if most digit values are chosen independently and checked as candidate strings, we classify it as enumeration. Realistic traces can mix derivation, case analysis, local enumeration, and verification. The binary label does not deny such mixtures: a trace that solves most digits from equations, possibly with case analysis over a few free digits, counts as derivation. The label therefore isolates the pattern we study, *i.e.*, whether the model abandons structural reasoning.

Automatic Categorization with LLM Judges. We use Gemini-3.0-Flash as an LLM judge to label reasoning traces with a deliberately simple output space, following the SimpleQA (Wei et al., 2024) style of asking the grader for a single unambiguous label. Internally, we use three labels: DERIVE, ENUM, and LOOP. LOOP has not appeared in the main paper, since it is a degenerate descendant of ENUM and was used primarily for our error analysis. It mostly shows the patterns where the model repeatedly enumerates or checks similar candidates until it gets stuck. For all main-paper strategy plots, LOOP is merged into ENUM because both reflect failure to perform structural derivation. To validate the precision of the judge, we manually annotated 100 randomly sampled traces and found 96% agreement with the LLM judge. This 4% disagreement is much smaller than the derivation gap between MONO and MIXED on Length Change (48% *vs.* 91%), so our conclusions are robust to judge errors. We label all generated responses, both correct and incorrect, so the strategy shares are not biased toward successful traces. The prompt is included in Fig. B.1.

B.2 CONFOUNDING FACTORS IN FIXING GENERALIZATION

In Sec. 4, we have shown that the reasoning strategy collapse of the MONO model can be fixed by introducing an auxiliary set of bijective tasks with stricter requirements in terms of bit-rates per token

```
You are a reasoning-pattern classifier for digit-string inverse-function problems. You will read a model
      's step-by-step reasoning and classify it into exactly one category.

Categories:

(A) DERIVE -- Structured constraint solving
The model identifies dependency structure: it chooses SOME digit positions freely, then DEDUCES the
      remaining digits from constraint equations. The key test: are most digit values produced as
      outputs of equations (derived) rather than chosen independently (guessed)?
- Choosing one free variable then computing the rest from equations IS derive.
- Setting up equations AND using them to solve for unknowns IS derive.

(B) ENUM -- Brute-force trial of full assignments
The model assigns ALL or nearly all digit positions by guessing, then checks if the assignment works.
      The workflow is: guess a complete input -> verify -> discard or keep -> guess again.
- Guessing complete strings and checking output IS enum.
- Setting up equations but then IGNORING them and guessing complete strings IS enum.
- Trying many different digit-tuple assignments one by one and checking each IS enum.
- Trying values without using prior constraints to narrow choices IS enum.
- For functions like max-min-of-triplets or second-order-difference, if the model guesses complete digit
       patterns and checks them, that IS enum even if equations were written down first.

(C) LOOP -- Stuck repetition
The model repeats the same or nearly identical computations/attempts multiple times without making
      progress. LOOP is a degenerate form of enumeration where the model gets stuck.
Key signals:
  1. The SAME digit values or computation blocks appear multiple times.
  2. The model keeps getting the same wrong result but retries similar values.
  3. The response ends abruptly mid-computation after repeated attempts.
  4. The latter reasoning recycles earlier content without finding new solutions. A response can start
      with valid derive or enum reasoning but degenerate into a loop. If the model spends most of its
      token budget stuck in repetition, classify as LOOP. A very long response (>10000 chars) with NO <
      answer> tag and repeated computation blocks is LOOP. A long response that makes genuine progress
      trying DIFFERENT values is ENUM, not LOOP.

Grading procedure:
1. Check for LOOP. If yes -> C.

2. If not LOOP, determine DERIVE vs ENUM. Find where the model starts proposing solution digit values.
      If most values are outputs of constraint equations -> A. If most are independently guessed and
      checked -> B.

Output format: Answer with ONLY the single letter A, B, or C. Do not explain.

USER MESSAGE:
PROBLEM:
{problem}

RESPONSE:
{response}

Classify the reasoning strategy. Answer A, B, or C only.
```

Figure B.1: LLM prompt for reasoning strategy categorization.

(as in Sec. 5). To validate that our discovery is of scientific rigor and commonness, we discover three common confounding factors that could contribute the phenomenon. In fact, they are not contributing factors to better OOD generalization; thus, this section better supports our observations of reasoning strategy collapse and generalization in RLVR.

B.2.1 Training Task Difficulty

A possible alternative explanation is that bijective co-training improves OOD performance by making the RL objective substantially harder, rather than by changing the rewarded reasoning strategy. Fig. B.2 tests this explanation using the training-time verifier score. The curve does show an early optimization lag for MIXED. However, this lag is transient: the reward of the two model becomes similar, and the overall reward of MIXED is even higher eventually . Thus the reward signal does not support a difficulty explanation in which MIXED generalizes better simply because it is trained on a uniformly harder objective. In addition, the added bijective tasks are also further from the non-bijective OOD evaluations, since they ask for a single x rather than multiple valid answers. Therefore, the improvement of MIXED cannot be reduced to either a closer evaluation format or a consistently harder training objective.

Intrinsic Difficulty. Training reward depends on the model, so we also compare the functions directly. The bijective functions (Sec. A.2) mainly consist of reversible position permutations, digit-wise modular shifts, and invertible running-sum variants, while the non-bijective functions mainly introduce many-to-one interactions between neighboring digits. Consider the cost of an optimal

inversion program. A representative bijective function reverses the input and adds 1 modulo 10 to every digit; its inverse is deterministic and takes $O(n)$ modular operations. A representative non-bijective function emits the cyclic floor-average of neighbors, $y_i = \lfloor (x_i + x_{i+1})/2 \rfloor$; each output digit admits multiple predecessors, so inversion branches at ambiguous positions and costs $O(2^k n)$, where k is the number of branching positions. Thus, the bijective functions are not intrinsically harder. For an LLM, difficulty also depends on the adopted strategy: pure enumeration is *harder* on bijective tasks, where only one of 10^n inputs is correct. This strategy dependence is why we also report the training reward above.

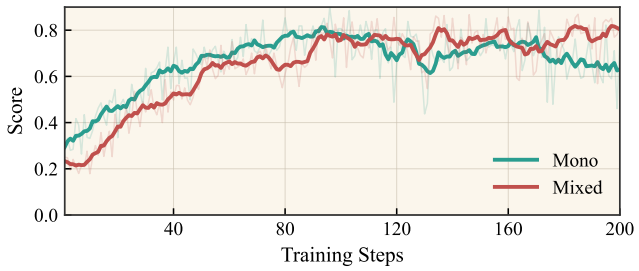


Figure B.2: Training-time accuracy for MONO and MIXED models.

B.2.2 ANOTHER SET OF FUNCTIONS

We also show similar observations as Sec. 4 with another set of functions, especially changing the set of non-bijective functions for training and evaluation. Based on the new functions, we re-train new MONO and MIXED models.

Function List. This confounding study uses a balanced alternative non-bijective family with the same split size as Sec. A.2: 18 training functions and 6 held-out functions for OOD evaluation. Same as before, we use `nb_xx` for the convenience of notations. **Training**: `nb_00`: cyclic floor-average of neighbors, then add 1. `nb_01`: cyclic ceil-average of neighbors, then add 2. `nb_03`: cyclic capped neighbor sum, then add 1. `nb_04`: cyclic floor-average of triples, then add 1. `nb_05`: cyclic ceil-average of triples, then add 2. `nb_07`: cyclic triple median, then add 2. `nb_08`: mirror-pair floor-average, then add 1. `nb_09`: mirror-pair ceil-average, then add 2. `nb_11`: mirror-pair maximum, then add 2. `nb_12`: sort adjacent pairs ascending, then add 1. `nb_13`: sort adjacent pairs descending, then add 2. `nb_15`: sort full triplets descending, then add 2. `nb_16`: prefix minimum, then add 1. `nb_17`: prefix maximum, then add 1. `nb_19`: suffix maximum, then add 2. `nb_20`: prefix-sum state with floor-half emission, then add 3. `nb_21`: prefix-sum state with folded-distance emission, then add 1. `nb_23`: prefix-sum state with distance-to-five emission. **Evaluation:** `nb_02`: cyclic neighbor minimum, then add 1. `nb_06`: cyclic triple minimum, then add 1. `nb_10`: mirror-pair minimum, then add 1. `nb_14`: sort full triplets ascending, then add 1. `nb_18`: suffix minimum, then add 2. `nb_22`: prefix-sum state with squared-state emission, then add 1.

Analysis. As shown in Fig. B.3, the MIXED model is still better than MONO model on OOD evaluations, with similar performance for in-distribution cases. This is consistent with our observations in Sec. 4. Notably, the difference in the most challenging LenChange set marks a drastic difference in OOD generalization. In addition, these observations are accompanied with the consistent observations on reasoning strategies, where the MIXED model more robustly utilizes the derivation reasoning strategy, and leads to better OOD performance. Such an observation suggests that our observations in Sec. 4 is generalizable.

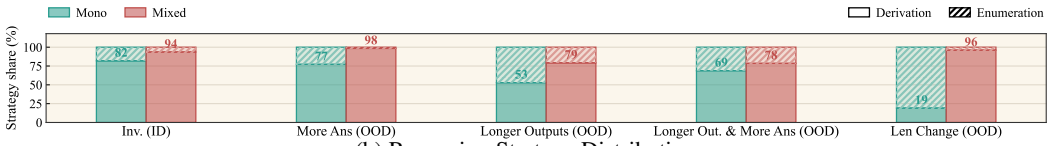
Regarding the performance, we also would like to share that this additional set of non-bijective functions are easier for Qwen2.5-7B model, which leads to much better performance on the OOD evaluation like longer outputs and more answers: in fact, the model suffers less from harder OOD generalization in this case. Such an observation also points to a potential future work as to how to control the difficulty levels of the functions themselves in the controllable studies of RLVR.

B.2.3 TRAINING TASK DIVERSITY

We finally validate whether MIXED performs better not because of its bijectiveness, but the additional diversity in the training tasks. We therefore adopt the additional functions in Sec. B.2.2 into the original MONO training, so that it possesses the same number of functions in the training tasks as MIXED. As shown in Fig. B.4a, these additional functions do not lead to better performance. Instead,

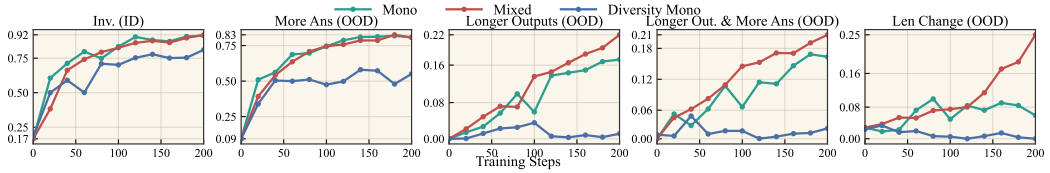


(a) Evaluation Accuracy During RL

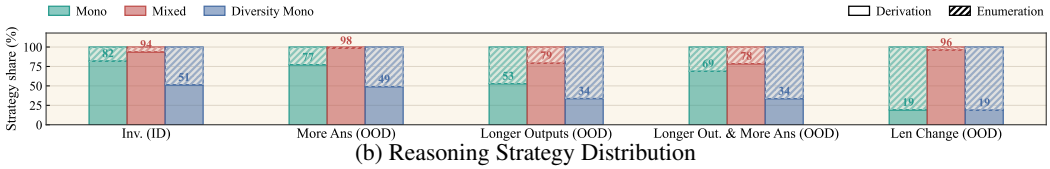


(b) Reasoning Strategy Distribution

Figure B.3: Confounding experiments via training on a new set of functions. The distribution of reasoning strategies and better OOD accuracy of MIXED validates our conclusion in Sec. 4.



(a) Evaluation Accuracy During RL



(b) Reasoning Strategy Distribution

Figure B.4: Confounding experiment via increasing the diversity of non-bijective functions for training. DIVERSITY MONO does not achieve the effect as MIXED, validating our Sec. 4.

they dilute the learning signals inside the original set of functions, which have shared function primitives. So enlarging the non-bijective function set does not by itself induce derivation, and the OOD gain of MIXED is not explained by greater training-task diversity.

B.2.4 REPEATED RUNS

We repeat the MONO and MIXED runs with two more random seeds and differently sampled training data, keeping all other settings unchanged. Table B.1 reports the final accuracy and derivation share on Length Change, the largest OOD shift. Every run reproduces the main observation: MIXED reaches several times higher accuracy and uses derivation far more often than MONO.

Table B.1: Repeated runs on Length Change. Higher is better for both metrics. "Main" is the run in Fig. 2.

	Accuracy (%)		Derivation (%)	
Run	MONO	MIXED	MONO	MIXED
Seed 0	5.7	**21.6**	46.0	**95.1**
Seed 1	4.1	**18.3**	50.8	**93.2**
Main	6.0	**25.2**	48.0	**91.6**
Mean ± std	5.3 ± 1.0	**21.7 ± 3.5**	48.3 ± 2.4	**93.3 ± 1.8**

C DETAILS OF THEORETICAL ANALYSIS IN SEC. 5

This section gives the proof details for Propositions 1 and 2. We use the search-MDP notation from Sec. 5: transitions are deterministic, a trajectory τ terminates with a candidate solution $y_\tau \in \mathcal{Y}$, and the outcome reward is $R(\tau) = \mathbf{1}[y_\tau \in \mathcal{Y}^\star]$ for the correct-solution set $\mathcal{Y}^\star \subseteq \mathcal{Y}$. For any state s,

the success probability $p^\pi(s) = \mathbb{P}_{\tau\sim\pi}[y_\tau \in \mathcal{Y}^\star \mid s_t = s]$ is the value function $V^\pi(s)$. The central quantity is the one-token success-probability increment

$$\Delta p_t := p^\pi(s_{t+1}) - p^\pi(s_t),$$

which is the probability-scale version of the bits-per-token signal in Sec. 5.1: positive Δp_t means that action a_t moves the current reasoning state closer to a correct terminal solution.

C.1 Proof of Proposition 1: Advantages Are Success Increments in Actor-Critic

Lemma 1 (Success-increment decomposition) *For any complete trajectory τ generated under policy π,*

$$R(\tau) - p^\pi(s_0) \;=\; \sum_{t=0}^{T-1} \Delta p_t.$$

Proof 1 *The sum on the right collapses to $p^\pi(s_T) - p^\pi(s_0)$. At a terminal state, the final solution is already fixed, so $p^\pi(s_T) = \mathbf{1}[y_\tau \in \mathcal{Y}^\star] = R(\tau)$.*

The lemma rewrites the trajectory-level outcome, relative to the prompt baseline, as accumulated per-token progress. Starting from REINFORCE (Williams, 1992) for $J(\pi_\theta) = \mathbb{E}_\tau[R(\tau)]$,

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\tau\left[R(\tau)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right],$$

and using the value baseline $V^\pi(s_t) = p^\pi(s_t)$, the policy-gradient theorem gives

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} A^\pi(s_t,a_t)\,\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \quad A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t). \tag{C.1}$$

With terminal reward and deterministic transitions, taking a_t fixes the next state, so $Q^\pi(s_t,a_t) = V^\pi(s_{t+1}) = p^\pi(s_{t+1})$. Therefore $A^\pi(s_t,a_t) = p^\pi(s_{t+1}) - p^\pi(s_t) = \Delta p_t$. Substituting this identity into (C.1) proves Proposition 1. □

Implication. Proposition 1 is the formal bridge between bits per token and RL credit. In our search view, a token receives positive advantage exactly when it increases the probability of eventually reaching a correct solution. A reasoning strategy that reliably prunes the candidate space, such as analytical derivation in our inversion tasks, therefore creates repeated positive increments; brute-force enumeration creates much weaker increments except when a guessed candidate happens to land near a solution. This is the sense in which outcome reward can still favor an internal reasoning process.

C.2 Proof of Proposition 2: GRPO

Estimator. GRPO does not use a learned critic to assign per-token advantage. For each prompt s_0, it samples a group of G independent rollouts $\{\tau^{(i)}\}_{i=1}^G \sim \pi_\theta$, observes rewards $R^{(i)} := R(\tau^{(i)})$, and computes

$$\bar{R} := \frac{1}{G}\sum_{i=1}^{G} R^{(i)}, \qquad \hat{\sigma}_R := \sqrt{\frac{1}{G}\sum_{i=1}^{G}(R^{(i)} - \bar{R})^2}.$$

Every token in rollout i receives the same trajectory-level advantage $\hat{A}^{(i)} := (R^{(i)} - \bar{R})/\hat{\sigma}_R$. Omitting clipping and KL terms, which affect optimization stability but not the population credit identity studied here, the estimator is

$$\hat{g}_{\text{GRPO}} = \frac{1}{G}\sum_{i=1}^{G}\hat{A}^{(i)}\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}). \tag{C.2}$$

Group rewards estimate the prompt value.

Lemma 2 (Baseline consistency) *The group statistics are Monte Carlo estimators of the value and its standard deviation:*

$$\mathbb{E}[\bar{R}] = p^\pi(s_0), \qquad \hat{\sigma}_R^2 \xrightarrow{a.s.} p^\pi(s_0)(1-p^\pi(s_0)) =: \sigma^2(s_0) \quad \text{as } G \to \infty.$$

Proof 2 *Each $R^{(i)}$ is an i.i.d. Bernoulli draw with mean $p^\pi(s_0)$; the result follows from the law of large numbers applied to the sample mean and sample variance.*

Under the large-G approximation $\bar{R} \approx p^\pi(s_0)$ and $\hat{\sigma}_R \approx \sigma(s_0)$, the success-increment decomposition gives

$$\hat{A}^{(i)} \;\approx\; \frac{R^{(i)} - p^\pi(s_0)}{\sigma(s_0)} \;=\; \frac{1}{\sigma(s_0)} \sum_{t=0}^{T-1} \Delta p_t^{(i)}. \tag{C.3}$$

Thus the trajectory-level GRPO advantage is the normalized sum of per-step success increments along that rollout. GRPO never observes the individual token-level increments, but the score-function identity recovers the same token-level credit in expectation.

Recovering token-level credit. Under the same large-G approximation,

$$\mathbb{E}[\hat{g}_{\text{GRPO}}] \;\approx\; \frac{1}{\sigma(s_0)} \sum_{t=0}^{T-1} \mathbb{E}_\tau[(R(\tau) - p^\pi(s_0))\, \nabla_\theta \log \pi_\theta(a_t \mid s_t)]\,.$$

Fix a position t and write $g_t = \nabla_\theta \log \pi_\theta(a_t \mid s_t)$, and the remaining expected terminal reward is

$$\mathbb{E}[R(\tau) \mid s_t, a_t] = p^\pi(s_{t+1}).$$

Therefore,

$$\mathbb{E}_\tau[(R(\tau) - p^\pi(s_0))g_t] = \mathbb{E}_\tau[(p^\pi(s_{t+1}) - p^\pi(s_0))g_t].$$

Decomposing $p^\pi(s_{t+1}) - p^\pi(s_0) = \Delta p_t + p^\pi(s_t) - p^\pi(s_0)$ gives one desired term and one state-dependent baseline. The baseline term vanishes because

$$\mathbb{E}_\tau[(p^\pi(s_t) - p^\pi(s_0))g_t] = \mathbb{E}_\tau\big[(p^\pi(s_t) - p^\pi(s_0))\mathbb{E}_{a_t \sim \pi_\theta(\cdot|s_t)}[g_t \mid s_t]\big] = 0,$$

where $\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a \mid s)] = \sum_a \nabla_\theta \pi_\theta(a \mid s) = \nabla_\theta 1 = 0$. Hence

$$\mathbb{E}_\tau[(R(\tau) - p^\pi(s_0))g_t] = \mathbb{E}_\tau[\Delta p_t\, g_t].$$

Summing over t,

$$\mathbb{E}[\hat{g}_{\text{GRPO}}] \;\approx\; \frac{1}{\sigma(s_0)}\, \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \Delta p_t\, \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] \;=\; \frac{1}{\sigma(s_0)}\, \nabla_\theta J(\pi_\theta),$$

where the final equality is Proposition 1. In the large-G limit, GRPO therefore has the same population credit direction as the actor-critic policy gradient, up to the prompt-level normalization $1/\sigma(s_0)$. For finite groups, the sample mean and standard deviation introduce the usual Monte Carlo error of order $O(1/\sqrt{G})$. □

Implication. Proposition 2 shows that the same mechanism applies to the algorithm used in our experiments, even though GRPO assigns only trajectory-level advantages and has no learned critic. Group rewards estimate the prompt-level success baseline, and the score-function identity recovers the same token-level success-increment credit in expectation. Thus changing the task structure can change which reasoning strategies RLVR reinforces: stricter tasks make high-bit-rate steps more reward-distinguishable, which explains why bijective or constrained variants shift the model toward derivation and improve OOD strategy generalization.

C.3 Validation Accuracy Under Bit-Rate Interventions

Sec. 5.3 uses Short Budget and Constrained prompts to intervene the bit-rate prediction and validates our theoretical analysis; the central evidence is already in Fig. 4, where both interventions increase derivation across the OOD axes and directly confirm the theory's prediction about which reasoning strategy RLVR should reinforce. This appendix subsection is therefore mainly a completeness check: it reports the validation accuracy associated with those runs.

Fig. C.1 plots the same five validation as Fig. 4. The two original baselines, MONO and MIXED, are shown in shallow colors for reference, while Short Budget and Constrained prompts are highlighted. The accuracy curves make an important qualification explicit: stronger derivation incentives do not by themselves guarantee the best final accuracy. The balanced MIXED run remains the strongest overall by final accuracy, especially on the most shifted LEN CHANGE axis, while Short Budget and Constrained prompts win only on some axes. This is consistent with the mixture-ratio analysis in Sec. 6.2 and the rollout audit in Sec. E: reasoning strategy is a necessary driver of OOD generalization, but accuracy also depends on task-specific execution skills such as modular arithmetic, faithful code reading, and handling multi-valued inverses. Thus Fig. C.1 does not weaken the theory result in Fig. 4; it separates the strategy-selection mechanism from the additional execution reliability needed for the best end-to-end performance.

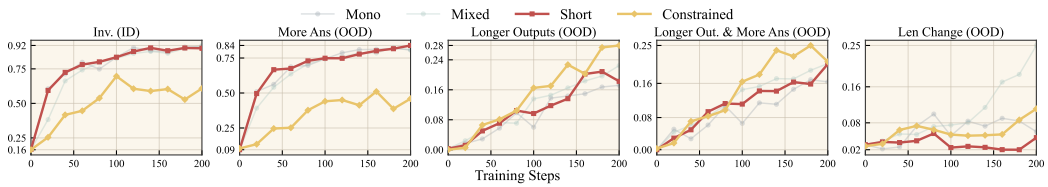


Figure C.1: Accuracy under bit-rate interventions (Sec. 5.3).

Table D.1: Comparison of null conditions. Shuffling at different levels lead to different rigor. We finally select the null condition that is the strongest, for the probing experiments (Sec. 6.1).

Null type (Shuffle)	What it preserves	Balanced Accuracy at $N = 0$
Global	Overall frequency	50%
Per Eval Set	Per eval-set frequency	65%
Function $\times$ Eval (Ours)	Both function and eval-set frequency	76%

D Probing Details

Sec. 6.1 asks when the hidden state contains information about the reasoning strategy that will appear in the generated trace. This section provides the detailed implementations and considerations.

Setup. We conduct probing for both MONO and MIXED models. For each checkpoint we evaluate four held-out OOD sets built from the same six non-bijective functions. Then, we extract the hidden states from the 21-th LLM layer (Qwen2.5-7B has 28 transformer layers) at the $N \in \{0, 100, 200, 300, 500, 1000, 1500\}$ positions after the `<think>` token. This layer is intuitively containing the rich features and is not too close to the last layer simply mapping features to tokens.

In probing, we train the linear classifier to predict the derivation/enumeration labels annotated by our strategy categorization (Sec. B.1). We use a logistic-regression probe with 5-fold stratified cross-validation. The folds preserve the evaluation set, function, and strategy label. Since the two strategies are imbalanced across models and evaluation sets, we report balanced accuracy, a commonly used metric for imbalanced classification that averages the recall of the two classes.

Specifically, let $z_i \in \{\text{Derive}, \text{Enum}\}$ be the strategy label for response i, and let $\hat{z}_i$ be the probe prediction. For a strategy class c, define

$$\text{recall}(c) = \frac{\sum_i \mathbf{1}[z_i = c \wedge \hat{z}_i = c]}{\sum_i \mathbf{1}[z_i = c]}.$$

The balanced accuracy is

$$\text{BAcc} = \left(\text{recall}(\text{Derive}) + \text{recall}(\text{Enum})\right)/2.$$

Null Condition. To validate that the representation indeed encodes strategy, instead of only relying on the shortcut of memorizing the statistics of function names and evaluation set properties, we construct a null condition as a contrast to determine the significance of linear probing. We adopt the shuffling of labels to create the null condition, and the labels are shuffled only among prompts that share the same evaluation set and function. This keeps the strategy frequency of every function on every evaluation set fixed, while breaking the alignment between individual hidden states and individual strategy labels. Table D.1 shows why this is the appropriate and most rigorous null condition, compared with two less careful shuffling.

Response Structure. Another crucial detail in Sec. 6.1 and Fig. 5 is how we assign the semantic meanings to different positions inside the reasoning trace. Although the problems vary, with changes of how the model thinks, we find that the responses from the LLMs follow a similar structure, which is intuitively utilized to interpret the results in Fig. 5. We demonstrate an illustrative example in Fig. D.1. Our token counts in Fig. 5 are estimated from manually inspecting 50 samples of derivation responses and 50 brute-force-enumeration responses per model, and we specifically calculate the position of the keywords that indicate the change of reasoning stage.

E Mixture Ratios: Detailed Error Analysis

Sec. 6.2 shows two facts that must be read together: adding more bijective data monotonically increases analytical derivation, but the balanced 1:1 mixture is most accurate. To separate strategy

```
<think>
To find three different digit strings x such that main_solution(x) returns "22",
we need to analyze the function for inputs of length 2.

The function takes a string s of length 2 and computes the cyclic absolute
difference between each digit and the next digit. Let the input string be ab,
where a and b are digits. The function computes
  first output digit  = |a - b|
  second output digit = |b - a|.

We need the output to be "22", so both positions must equal 2:
  |a - b| = 2
  |b - a| = 2.

These constraints are symmetric, so it is enough to solve |a - b| = 2.

Case 1: a - b = 2.
  a=2, b=0 -> "20"
  a=3, b=1 -> "31"
  a=4, b=2 -> "42"
  a=5, b=3 -> "53"
  a=6, b=4 -> "64"
  a=7, b=5 -> "75"
  a=8, b=6 -> "86"
  a=9, b=7 -> "97"

Case 2: b - a = 2.
  a=0, b=2 -> "02"
  a=1, b=3 -> "13"
  a=2, b=4 -> "24"
  a=3, b=5 -> "35"
  a=4, b=6 -> "46"
  a=5, b=7 -> "57"
  a=6, b=8 -> "68"
  a=7, b=9 -> "79"

We only need three different inputs, so choose "20", "31", and "42".
Verification:
  "20" gives |2-0|=2 and |0-2|=2, so the output is "22".
  "31" gives |3-1|=2 and |1-3|=2, so the output is "22".
  "42" gives |4-2|=2 and |2-4|=2, so the output is "22".

All three inputs satisfy the condition.
</think><answer>20,31,42</answer>
```

Figure D.1: A representative derivation trace. The model first restates the question, then explains the function, and introduces variables to solve the problem.

choice from execution reliability, we sample 128 OOD prompts where the three ratio models, 1:3, 1:1, and 3:1, disagree. For each prompt, we inspect the greedy response from all three ratios (384 traces total), and annotate the strategy and primary error type. Unless stated otherwise, the percentages are over all 128 audited prompts for that ratio.

Based on the contexts from Sec. 6.2, we especially analyze the error case when bijective tasks is dominating (3:1), when it can correctly utilize the derivation strategy. (1) The 3:1 model chooses derivation most often, but its execution-error rate rises to 32.8% of all audited OOD samples (24.2% for 1:1) and to 57.7% on LONGER OUTPUTS. A representative failure comes from the held-out prefix-square rule. The code maintains a running state by `state = (state + digit) % 10`, and each output digit is `(state * state) % 10`. Equivalently, if s_i is the running state after reading the i-th input digit, then the target digit must satisfy $y_i = s_i^2 \bmod 10$. For target "119641", the model must therefore find input digits whose running states have square residues $1, 1, 9, 6, 4, 1$ in order. All three ratios recognize this structure and begin solving the modular state constraints. The 3:1 model, however, does not finish the chain: after working through only the early states, it writes that it will "solve for s_3, s_4, s_5 similarly" and then directly emits `102864` and `926844`. These are not valid completions. (2) The same model also outputs less solutions than required, covering 11.7% cases. This also suggests the importance of refining the task-specific skills beyond meta-level reasoning strategies, as explained in Sec. 6.2.

F ANALYSIS OF ZERO-SHOT MATH EVALUATION

This section covers implementation details and disagreement examples supporting the analysis in Sec. 6.3. Sec. F.1 describes the evaluation protocol. The remaining subsections organize the qualitative analysis.

F.1 Evaluation Protocol

We evaluate with the math-evaluation-harness (Gou & contributors, 2023), using greedy decoding and the default prompt for a fair comparison across all the models. The benchmarks are MATH500 (Hendrycks et al., 2021; Lightman et al., 2024), Minerva-Math (Lewkowycz et al., 2022), AIME24, AIME25, AMC23, and OlympiadBench (He et al., 2024), scored with the harness's default evaluators.

F.2 Mixed Faithful To Problem Structure

As described in Sec. 6.3, the Mixed model can better keep the structure of the problem, which is consistent with the derivation strategy usage and our discover on the later strategy committment from hidden states.

Example: AIME 2025 (base divisibility).

> *Problem.* Find the sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b.

Here 97_b denotes the numeral 97 written in base b, so its decimal value is $9b+7$; similarly, $17_b = b+7$. The notation $u \mid v$ means that u divides v. Mixed rewrites the divisibility condition by noting $97_b = 9b + 7$ and $17_b = b + 7$, then performs the algebraic identity $9b + 7 = 9(b + 7) - 56$. Hence $b + 7 \mid 56$. It enumerates the full divisor list $\{1, 2, 4, 7, 8, 14, 28, 56\}$, filters by $b > 9$ (equivalently $b + 7 > 16$), and finds $b + 7 \in \{28, 56\}$, i.e. $b = 21$ and $b = 49$. The answer is $21 + 49 = 70$ (correct). Mono reparameterizes the same condition as $9b + 7 = k(b + 7)$ for some integer k, solves to $b = 7(k - 1)/(9 - k)$, and concludes that $9 - k$ must divide 7. It checks only the divisors $\{1, -1, 7, -7\}$ of 7 instead of all divisors of 56, finds $b = 49$ as the only base > 9, and outputs 49. The mistake is not arithmetic: Mono collapses an enumeration over divisors of 56 into one over divisors of 7, dropping most of the case structure; Mixed keeps the full divisor set in scope.

Example: AMC 2023 (cubic with integer roots).

> *Problem.* For how many ordered pairs (a, b) of integers does the polynomial $x^3 + ax^2 + bx + 6$ have three distinct integer roots?

Mixed enumerates the eight ordered triples (p, q, r) of distinct integers with $pqr = -6$, computes $(a, b) = (-(p + q + r), pq + qr + rp)$ for each, and deduplicates to the 5 distinct pairs (correct). Mono lists only three such triples, misses $(-1, 1, 6)$, $(-2, 1, -3)$, $(2, -1, -3)$, and others, and outputs 2.

Example: AMC 2023 (custom operator).

> *Problem.* For $u = a + bi$ and $v = c + di$, define $u \otimes v = ac + bdi$. Find $|z|^2$ given that $z \otimes z = z^2 + 40$.

Mixed parses the operator correctly: for $z = x + yi$, $z \otimes z = x^2 + y^2 i$. It compares real and imaginary parts against $z^2 = (x^2 - y^2) + 2xyi$ and solves to get $|z|^2 = 50$ (correct). Mono mis-substitutes into the operator definition, produces $z \otimes z = x^2 + y^2 i(x + y)$, and falls into a dead-end loop.

F.3 Mono Better At Execution Reliability

Alternatively, we consider the observation in Sec. 6.2 and Sec. 6.3, where the Mono model benefits from execution reliability, and has the advantage mostly from the computation-bottlenecked MATH500.

Example: MATH500 (polynomial long division).

> *Problem.* Find the quotient when $x^6 - 3$ is divided by $x + 1$.

Mixed sets up polynomial long division correctly and performs the first two steps, then loses track mid-computation and outputs 0. Mono carries the division through all six steps and obtains the correct quotient $x^5 - x^4 + x^3 - x^2 + x - 1$.

Example: MATH500 (parallelogram fourth vertex).

> *Problem.* A parallelogram has three of its vertices at $(5, 3)$, $(6, 8)$, $(7, 4)$, and the fourth at (x, y) with $x > 7$. Find $x + y$.

MIXED tries one diagonal pairing, finds $x = 6$ (which violates $x > 7$), and stops without trying alternatives. MONO also finds $x = 6$ from the first pairing, recognizes the constraint violation, re-evaluates with a different diagonal assignment, and finds $x = 8$, $y = 9$, giving $x + y = 17$ (correct).

Example: MATH500 (group-size condition).

> *Problem.* A class has between 30 and 50 students and cannot be broken into multiple groups of equal size. Adding two instructors to the count still does not allow such a split. How many students are in the class?

MIXED misinterprets "cannot be broken into multiple groups of equal size" as "not a perfect square" and outputs 30. MONO correctly interprets the condition as "n is prime," searches for twin primes in the range, and outputs 41 (correct).

G AGENTIC TRAINING DETAILS

In Sec. 6.4, we explore applying our fix for reasoning strategies to agentic multi-turn tasks, and this section describes the detailed implementations. We follow RAGEN (Wang et al., 2025; 2026) and train the Qwen2.5-3B-Instruct model in the Sokoban environment. Since this task is deterministic, we can calculate the optimal step for each puzzle and utilize *optimal_steps + 2* actions to mark a stricter task than the original. We mix it with standard Sokoban at 1:1 and compare against training on standard Sokoban alone, holding all other hyperparameters fixed.

In light of our theoretical analysis in Sec. 5, we realize that the original reward design in Sokoban introduces large variances, *e.g.*, the values for a total success is 10, and each box counts for a score 5, and no format reward guides the LLM to follow our action parser. Therefore, we add a 0.1 format reward to encourage following our thinking and answering format, then a 0.9 reward for pushing all the boxes on target, which is a more simple reward than the original Sokoban.

Same as our experiment in reasoning strategies, we use DAPO (Yu et al., 2025), especially filtering out the groups without reward variances. As shown in Fig. 7, the baseline DAPO run rises during the first 60 steps to a validation success rate of 24%, then drops sharply to below 10% over the next 40 steps. With the stricter variant mixed in, training climbs steadily to roughly 30% over the same window. The two curves track each other closely until step 60 and only diverge once the baseline begins to collapse, suggesting that the stricter signal enhances the agentic training. Although this experiment is not necessarily an OOD generalization, we hope this exploration could suggest the broader implication of our discovery and analysis for agentic scenarios in future works.

H LIMITATIONS, SOCIETAL IMPACTS, AND FUTURE WORKS

Limitations. (1) Our study is intentionally controlled: the inversion suite makes reasoning strategies observable and OOD shifts clean, but it does not claim to exhaust all forms of reasoning in realistic tasks. In fact, we can extend the current inversion task into multiple string equations, the equations without invalid answers, etc., to investigate even more reasoning strategies of LLMs. (2) In addition, the current categorization focuses on analytical derivation versus brute-force enumeration because this is the decisive strategy contrast for inverse-function reasoning. More realistic scenarios may involve a mixture of strategies within the same trace, including decomposition, case analysis, local enumeration, self-verification, tool use, and planning, which would require a finer-grained strategy taxonomy. (3) Our theory also focuses on a clean search-MDP abstraction and identifies bits per token as one mechanism by which outcome rewards favor generalizable reasoning. A clearer theory could further model the latent function that maps task structure, strategy choice, and execution reliability to OOD success. (4) Finally, our agentic experiment is a small extension to hint the potential of our discovery for agentic tasks: it suggests that stricter task structure can stabilize Sokoban training, while extension to full-scale long-horizon agentic tasks would require better measures to create clean testbed of OOD generalization and reasoning strategies. (5) Our main analysis uses one base model, Qwen2.5-7B-Instruct, chosen as the operating point where the phenomenon is identifiable (Sec. A.4). Other model families may sit at different operating points, and future analysis might need to select tasks localizing at the boundaries of such models' capabilities.

Societal Impacts. Our experiments inherit the general risks of LLMs, including possible biases from the base models, but this work mainly uses synthetic data and studies general reasoning behavior rather than user-facing deployment; if the identified task-structure principles improve OOD

generalization more broadly, they may also reduce the need for collecting large amounts of human data and provide positive downstream societal impact.

Future Works. (1) The most direct future direction is to extend the framework to richer reasoning patterns and more realistic scenarios where successful solutions naturally combine multiple strategies rather than selecting one dominant behavior. This would allow reasoning-strategy analysis to move from a clean binary distinction toward a compositional account of how models choose, combine, and switch strategies under distribution shift. (2) A second direction is to develop a sharper theory of strategy selection, especially by estimating strategy-dependent information gain from model traces, connecting it to learned representations, and characterizing when stricter auxiliary tasks improve strategy selection without sacrificing execution reliability. (3) A third direction is agentic RLVR. Our Sokoban result suggests that task structures demanding more useful information per action can matter beyond string inversion; future work can study how agents develop better reasoning patterns over long-term environments with memory, tools, exploration, and delayed rewards.