Limits of Generalization in RLVR: Two Case Studies in Mathematical Reasoning

Md Tanvirul Alam
Rochester Institute of Technology
Rochester, NY, USA
ma8235@rit.edu

Nidhi Rastogi
Rochester Institute of Technology
Rochester, NY, USA
nxrvse@rit.edu

Abstract

Mathematical reasoning is a central challenge for large language models (LLMs), requiring not only correct answers but also faithful reasoning processes. Reinforcement Learning with Verifiable Rewards (RLVR) has emerged as a promising approach for enhancing such capabilities; however, its ability to foster genuine reasoning remains unclear. We investigate RLVR on two combinatorial problems with fully verifiable solutions: *Activity Scheduling* and the *Longest Increasing Subsequence*, using carefully curated datasets with unique optima. Across multiple reward designs, we find that RLVR improves evaluation metrics but often by reinforcing superficial heuristics rather than acquiring new reasoning strategies. These findings highlight the limits of RLVR generalization, emphasizing the importance of benchmarks that disentangle genuine mathematical reasoning from shortcut exploitation and provide faithful measures of progress. Code available at `https://github.com/xashru/rlvr-seq-generalization`.

1 Introduction

Large language models (LLMs) have recently advanced rapidly on mathematical and programming benchmarks [4, 5, 20, 1, 21]. A key driver is *Reinforcement Learning with Verifiable Rewards* (RLVR), which fine-tunes pretrained models against automatically checkable signals such as exact answers or unit tests [4, 22]. This paradigm eliminates reliance on human annotation, enabling scalable training on large problem sets and delivering consistent gains on challenging reasoning tasks [3].

Despite strong empirical gains, the nature of RLVR's improvements remains unclear. Studies show it often boosts accuracy while reducing exploration [19, 2, 23, 13], with base model ability acting as a ceiling [23, 19]. Gains largely reflect re-weighting existing solutions, and reasoning coverage can even contract at larger sample sizes. Standard $Pass@K$ further over-credits "lucky" completions [16], whereas stricter metrics like $CoT\text{-}Pass@K$ are more reliable but harder to scale. Reward design can also yield surprising effects: a single example can rival large-scale training [15], and even spurious rewards can drive improvements in models with strong procedural biases [7]. Overall, RLVR appears to stabilize existing competencies rather than induce new reasoning strategies.

While prior studies have offered valuable insights, many rely on benchmarks where the correctness of reasoning is difficult to verify, making it unclear whether improvements reflect genuine mathematical competence or superficial pattern matching. We address this gap by focusing on two combinatorial problems with fully verifiable solutions: *Activity Scheduling*, which admits a unique greedy optimum, and the *Longest Increasing Subsequence* (LIS), which can be solved using dynamic programming. By constructing datasets where each instance has a single optimal sequence, we can precisely measure not only answer accuracy but also sequence fidelity and structural behaviors, such as correct sorting in Activity Scheduling. This verifiable setup provides a rigorous lens on RLVR, revealing when

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

Activity Scheduling Example

Determine the largest subset of activities that can be scheduled without any overlaps.

ID	Start	End
1	06:09	07:24
2	07:13	08:23
3	07:29	09:28
4	08:24	10:18
5	04:48	06:14

Ground truth: `\ids{5,2,4}`, `\answer{3}`

LIS Example

Determine the longest strictly increasing subsequence (by VALUE) from the rows below (use the row IDs).

ID	Value
1	797
2	476
3	335
4	452
5	606

Ground truth: `\ids{3,4,5}`, `\answer{3}`

Figure 1: Example question and ground-truth for Activity Scheduling (left) and LIS (right).

observed gains arise from heuristic shortcuts versus genuine reasoning strategies, and highlighting broader implications for the design of mathematical reasoning benchmarks.

2 Experimental Setup

2.1 Tasks

Activity Scheduling. Each activity i has start and finish times (s_i, f_i) with $s_i < f_i$, and intervals are half-open $[s_i, f_i)$. The goal is to select a maximum subset of non-overlapping activities [17]. Instances are constructed so that the greedy earliest-finish-time algorithm with deterministic tie-breaking yields the unique optimum, reported as IDs sorted by f_i (ties by smaller i).

Longest Increasing Subsequence (LIS). Given $a_1, \ldots, a_n \in \mathbb{Z}$, find indices $1 \le i_1 < \cdots < i_k \le n$ maximizing k with $a_{i_1} < \cdots < a_{i_k}$. Uniqueness is enforced via an $O(n^2)$ dynamic-programming count, and the LIS is reconstructed with patience sorting in $O(n \log n)$ [18].

Our generator enforces uniqueness of the optimal solution for both tasks, yielding a single ground-truth ID sequence with a fixed canonical reporting order (see Appendix A).

Dataset. We generate 2000 instances per task, half with hints, with sequence lengths 5–16. To avoid leakage, train and test use disjoint length ranges, leaving 462 test cases for Activity and 428 for LIS; the remainder form the training set. Example questions and their corresponding ground truths are illustrated in Fig. 1, with detailed prompts provided in Fig. 5 in Appendix.

2.2 Reward Functions

For each instance $(x, y^\star)$ with ground-truth answer $a^\star$ and unique optimal sequence $s^\star = (i_1, \ldots, i_L)$, the output y is parsed into $\hat{a}(y)$ (from `\answer{...}`) and $\hat{s}(y)$ (from `\ids{...}`); if parsing fails, $\hat{s}(y) = \varnothing$. All rewards lie in $[0, 1]$:

(1) Answer-only. This reward evaluates only the correctness of the final numeric answer:

$$r_{\text{ans}}(y) \;=\; \mathbb{I}[\hat{a}(y) = a^\star]\,,$$

where $\mathbb{I}[\cdot]$ denotes the indicator function, equal to 1 if the condition holds and 0 otherwise.

(2) Answer + Format (LIS only). To stabilize behavior when the policy begins omitting reasoning, we introduce a small formatting bonus, applied only to the *LIS* task(see §3.1)). Define the format indicator

$$\text{fmt}(y) \;=\; \mathbb{I}\Big[y \text{ contains } \texttt{<think>...</think>} \text{ and valid } \texttt{\answer\{...\}}, \texttt{\ids\{...\}}\Big],$$

and combine it with answer correctness using a mixing weight $\lambda = 0.1$:

$$r_{\text{ans+fmt}}(y) \;=\; (1-\lambda)\, r_{\text{ans}}(y) \;+\; \lambda\, \text{fmt}(y).$$

(3) Exact-IDs. Rewards 1 if and only if the predicted sequence matches the optimum:

$$r_{\text{ids,exact}}(y) \;=\; \mathbb{I}\big[\hat{s}(y) = s^\star\big].$$

(4) Prefix-IDs. This reward grants partial credit proportional to the length of the longest common prefix with the ground-truth sequence, while applying a small penalty if the predicted length is incorrect (clipped at 0). Define

$$m(y) \;=\; \max\Big\{\, m \in \{0,\dots,L\} : (\hat{s}_1,\dots,\hat{s}_m) = (i_1,\dots,i_m) \,\Big\},$$

and fix a length-penalty $\gamma = 0.1$. The reward is then

$$r_{\text{ids,prefix}}(y) \;=\; \max\Big\{\, 0,\; \frac{m(y)}{L} \;-\; \gamma\, \mathbb{I}\big[\, \hat{s}(y) = \varnothing \;\vee\; |\hat{s}(y)| \neq L \,\big] \Big\}.$$

(5) Sorting-Match (Activity only). In the *Activity Scheduling* task, sorting by finish time is the first step of the greedy algorithm. Interestingly, we observe that even without explicit instructions, model responses often begin with a sorted version of the input sequence, which can be extracted reliably (see §3.3). This motivates an auxiliary reward that checks whether the extracted sorted sequence matches the ground-truth sorted order of activities:

$$r_{\text{sort}}(y) \;=\; \mathbb{I}\big[\, \hat{s}_{\text{sort}}(y) \;=\; s^{\star}_{\text{sort}} \,\big].$$

where $\hat{s}_{\text{sort}}(y)$ is the sequence of activity IDs sorted as extracted from the model output, and $s^{\star}_{\text{sort}}$ is the canonical sorted sequence by increasing finish time (breaking ties by smaller ID).

2.3 Training & Evaluation

We fine-tune *Qwen2.5-7B-Instruct* [11] with GRPO [8, 4] using the `verl` framework [10]. Unless noted, max generation length is $T_{\max} = 2048$ (extended to 7680 for LIS with $r_{\text{ans+fmt}}$). Each PPO update uses 256 prompts with 8 rollouts from `vLLM` [6], trained for 20 epochs (120 updates) at learning rate 10^{-6} and no KL penalty ($\beta_{\text{KL}} = 0$). Training prompts match evaluation format.

Evaluation protocol. For each instance x we draw $k = 256$ samples $\{y_j\}_{j=1}^{256}$ with temperature 0.6 and top-$p = 0.95$, and parse each y_j into $(\hat{a}(y_j), \hat{s}(y_j))$ as in the reward definitions. We evaluate two complementary notions of accuracy: Acc_{ans} (correctness of the reported cardinality/length) and Acc_{ids} (exact match of the ID sequence). Both are measured under *Pass@k* and *Self-consistency (SC)*:

- **Pass@k** [23]. Success under a k-sample budget, defined by whether at least one of the k generations is correct:

$$\text{Pass@k}_{\text{ans}}(x) = \mathbb{I}\big[\exists\, j \le k :\; \hat{a}(y_j) = a^{\star}\big], \qquad \text{Pass@k}_{\text{ids}}(x) = \mathbb{I}\big[\exists\, j \le k :\; \hat{s}(y_j) = s^{\star}\big].$$

- **Self-consistency (SC)** [14]. Agreement between the majority prediction aggregated over k generations and the ground truth:

$$\tilde{a}_k(x) := \text{mode}\{\hat{a}(y_j)\}_{j=1}^{k}, \qquad \tilde{s}_k(x) := \text{mode}\{\hat{s}(y_j)\}_{j=1}^{k},$$

  with deterministic tie-breaking (numerically smallest for answers; lexicographic for sequences). We report

$$\text{SC}_{\text{ans}}(x;k) = \mathbb{I}\big[\tilde{a}_k(x) = a^{\star}\big], \qquad \text{SC}_{\text{ids}}(x;k) = \mathbb{I}\big[\tilde{s}_k(x) = s^{\star}\big].$$

All metrics are averaged over the test set. Because each instance admits a unique ground-truth answer $a^{\star}$ and sequence $s^{\star}$, these definitions are unambiguous. We plot the full curves $\{\text{Pass@k}\}_{k=1}^{256}$ and $\{\text{SC}(\cdot;k)\}_{k=1}^{256}$, and also report their values at $k = 256$.

3 Results & Analysis

3.1 Training with Exact Answer Reward

Fig. 2 compares the base model and the RLVR-trained policy on *Activity Scheduling* and LIS tasks.

Activity Scheduling. Since the target is a small integer (maximum schedule length 16), the base model quickly saturates to *Pass@k* ≈ 1.0 as k increases. While the RLVR-trained policy attains

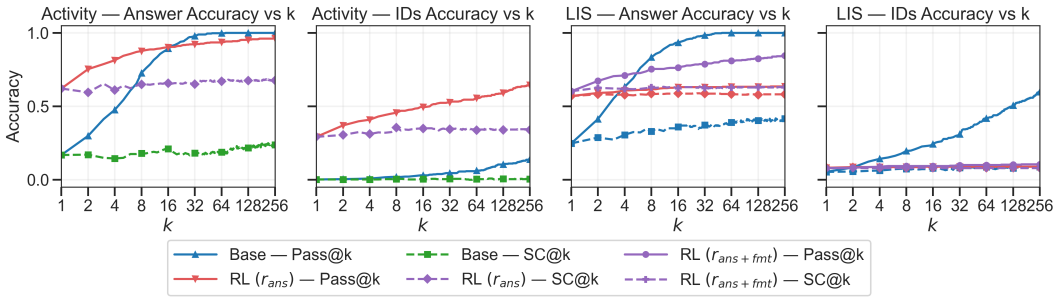

Figure 2: Performance comparison of Base, RL(r_{ans}), and RL($r_{\text{ans+fmt}}$) with Qwen2.5-7B.

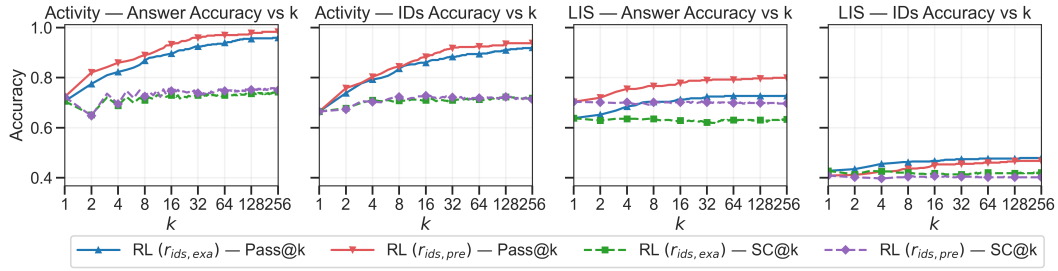

Figure 3: Performance comparison of RL models trained with $(r_{ids,exa})$ and $(r_{ids,pre})$ models on the *Activity* and *LIS* task with the Qwen2.5-7B model.

slightly lower *Pass@k* at large k, it achieves much higher self-consistency (about 0.68 vs. ~ 0.24 at k=256), indicating more stable predictions across samples. Under exact sequence ID matching, RLVR substantially outperforms the base model: at k=256, RLVR reaches $\text{Pass@k} \approx 0.64$ compared to 0.14, with $\text{SC} \approx 0.34$ compared to 0.004, reflecting a clear improvement in sequence-level fidelity. These results support prior findings that answer-only *Pass@k* can overstate model capability [16], whereas Acc_{ids} and SC provide more faithful measures of reasoning quality. RLVR improves both by reinforcing verified reasoning trajectories [16].

LIS. On *LIS*, training with the answer-only reward r_{ans} rapidly collapses intermediate reasoning: after just a few PPO updates, the policy drops its chain of thought and outputs terse final answers, reflected in a sharp decline in mean response length (Appendix D). Adding a format component ($r_{\text{ans+fmt}}$) mitigates this, keeping response lengths closer to the base model (Fig. 6). As shown in Figure 2, both RLVR-trained models achieve higher SC on answer than the base model; for r_{ans}, the *Pass@k* and SC curves nearly overlap, whereas $r_{\text{ans+fmt}}$ underperforms the base model at larger k. However, under exact sequence ID evaluation, both RLVR-trained policies remain weak, with low *Pass@k* and SC compared to the base model, in contrast to the *Activity* task, where RLVR improved both metrics.

Takeaway 1

RLVR improves answer-level generalization on both *Activity Scheduling* and *LIS*. However, only in *Activity Scheduling* do we observe reasoning gains, while on *LIS*, improvements stem from superficial heuristics or formatting strategies. Thus, RLVR can yield apparent task generalization without strengthening the underlying reasoning process.

3.2 Training with Sequence Rewards

We now evaluate sequence-aware objectives, comparing the exact-match reward $r_{\text{ids,exa}}$ and the prefix reward $r_{\text{ids,pre}}$ (cf. §2.2). Figure 3 summarizes results for Acc_{ans} and Acc_{ids} across k. On *Activity*, both rewards yield similar gains, substantially surpassing the base and r_{ans} models: at k=256, SC_{ids} rises from 0.34 to $0.72/0.71$ ($r_{\text{ids,exa}}/r_{\text{ids,pre}}$), and SC_{ans} increases from 0.68 to $0.74/0.75$. On *LIS*, a trade-off emerges: $r_{\text{ids,pre}}$ attains higher Acc_{ans}, while $r_{\text{ids,exa}}$ achieves higher Acc_{ids}, yet both outperform

the base and r_{ans} policies. At $k{=}256$, SC_{ids} improves from ≈ 0.08 to $0.42/0.40$, and SC_{ans} from 0.58 to $0.63/0.70$, indicating that sequence rewards enhance both answer- and sequence-level consistency.

Takeaway 2

Sequence-aware rewards improve sequence fidelity on both *Activity Scheduling* and *LIS*, as reflected in higher Acc_{ids}. They also provide modest secondary gains in Acc_{ans}, suggesting that RLVR generalization can benefit from incorporating intermediate or auxiliary objectives.

3.3 Sorting Performance and Reward on Activity Task

Sequence-aware rewards improve sequence matching partly by enhancing the "sorting preface" that models generate as the first step in activity scheduling. We extract candidate ID lists from outputs using simple patterns, succeeding on over 84% of RL-trained responses versus 40% for the base model (Appendix B). Evaluating **ExactSort** (exact match with ground truth) and **LCS fraction** (longest correctly sorted subsequence) shows that sequence rewards yield both higher exact sorting and better partial order fidelity.

Figure 4 summarizes sorting quality for the four models. Both sequence-reward policies ($r_{\text{ids,exa}}$ and $r_{\text{ids,pre}}$) improve *exact sorting* relative to the base and r_{ans} models (from $0.1\%/0.4\%$ to $2.0\%/1.9\%$) and increase the mean LCS (from $0.238/0.290$ to $0.517/0.444$). However, absolute exact-sorting accuracy remains very low ($\approx 2\%$), even though these same policies achieve high sequence correctness at evaluation (e.g., $\text{Pass@256}_{\text{ids}} \approx 0.60$ on *Activity*). This gap indicates that the sorting step is not reliably driving the final schedule, despite the frequent appearance of a "sorted" preface.

Training with Sorting-Match Reward. We tested a sorting-match reward r_{sort} to encourage the first step of activity scheduling, but this caused collapse: both Acc_{ans} and Acc_{ids} dropped to nearly 0% as the model output only the sorted sequence without enforcing non-overlap. Adding r_{sort} to r_{ans} and r_{ids} restored performance to the r_{ids} baseline but yielded no further gains. Curriculum experiments further showed that extended pretraining on r_{sort} hindered recovery under r_{ans}, with longer exposure preventing learning even on the training set (see Appendix H).

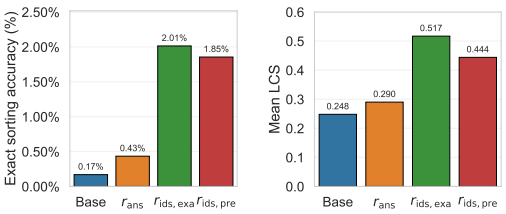

Figure 4: Sorting acc. on Activity Scheduling

Takeaway 3

In *Activity Scheduling*, RLVR with sequence rewards improves answer and sequence accuracy, but sorting accuracy remains very low. Models often emit a superficial "sorted" preface that neither matches the canonical order nor drives the final schedule, highlighting a disconnect between surface outputs and the underlying decision rule [12].

4 Limitations & Future Work

Our study investigates two verifiable reasoning tasks (*Activity Scheduling* and *LIS*) with a single base model (*Qwen2.5-7B-Instruct*). While this controlled setup helps isolate RLVR dynamics, conclusions may not generalize across tasks, model families, or scales, as prior work has shown diverse shortcut behaviors and model-dependent failure modes [7]. To partially address this, we provide additional results for *Llama-3.1-8B* in Appendix F.

Future work could expand to a broader range of models and problem domains, and employ mechanistic interpretability to probe the circuits and dynamics underlying RLVR-induced behaviors [9]. Such analysis may clarify whether improvements reflect genuine task learning or superficial strategies that exploit evaluation metrics. Our findings highlight this tension, emphasizing the need for careful evaluation and diagnostics when claiming improvements in reasoning from RLVR.

References

[1] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).

[2] Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. 2025. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617* (2025).

[3] Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, et al. 2025. Competitive programming with large reasoning models. *arXiv preprint arXiv:2502.06807* (2025).

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

[5] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720* (2024).

[6] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

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

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

[9] Lee Sharkey, Bilal Chughtai, Joshua Batson, Jack Lindsey, Jeff Wu, Lucius Bushnaq, Nicholas Goldowsky-Dill, Stefan Heimersheim, Alejandro Ortega, Joseph Bloom, et al. 2025. Open problems in mechanistic interpretability. *arXiv preprint arXiv:2501.16496* (2025).

[10] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. HybridFlow: A Flexible and Efficient RLHF Framework. *arXiv preprint arXiv: 2409.19256* (2024).

[11] Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. `https://qwenlm.github.io/blog/qwen2.5/`

[12] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems* 36 (2023), 74952–74965.

[13] Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. 2025. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939* (2025).

[14] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[15] Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, et al. 2025. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571* (2025).

[16] Xumeng Wen, Zihan Liu, Shun Zheng, Zhijian Xu, Shengyu Ye, Zhirong Wu, Xiao Liang, Yang Wang, Junjie Li, Ziming Miao, et al. 2025. Reinforcement Learning with Verifiable Rewards Implicitly Incentivizes Correct Reasoning in Base LLMs. *arXiv preprint arXiv:2506.14245* (2025).

[17] Wikipedia contributors. 2025. Activity selection problem — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Activity_selection_problem` Accessed: 2025-09-25.

[18] Wikipedia contributors. 2025. Longest increasing subsequence — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Longest_increasing_subsequence` Accessed: 2025-09-25.

[19] Fang Wu, Weihao Xuan, Ximing Lu, Zaid Harchaoui, and Yejin Choi. 2025. The invisible leash: Why rlvr may not escape its origin. *arXiv preprint arXiv:2507.14843* (2025).

[20] Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2024. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. Association for Computational Linguistics.

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

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

[23] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837* (2025).

Appendix

A Dataset Construction

A.1 Activity Scheduling

Problem model. We work on the standard unweighted interval scheduling problem. Each instance has m activities indexed by $I = \{1, \dots, m\}$ with start/finish times (s_i, f_i) and $s_i < f_i$. Intervals are treated as *half-open* $[s_i, f_i)$, so touching endpoints are compatible ($f_j \le s_i$ means i and j can both be selected). We fix a deterministic tie-breaking order by the tuple (f_i, s_i, i) whenever sorting is required. The canonical reporting order for ground truth is by non-decreasing f (ties by smaller i).

Sampling on a minute grid. All times lie on an integer minute grid. To construct a candidate set of m activities, we: (i) sample $m \sim \mathrm{Unif}\{m_{\min}, \dots, m_{\max}\}$; (ii) repeatedly sample integer start times $s \sim \mathrm{Unif}\{0, \dots, S_{\max}\}$ and integer durations $d \sim \mathrm{Unif}\{D_{\min}, \dots, D_{\max}\}$, set $f = s + d$, and accept the interval if $f \le S_{\max} + D_{\max}$; we continue until m intervals are accepted. (Values used in our experiments: $m_{\min} = 5$, $m_{\max} = 16$, $S_{\max} = 9 \times 60$, $D_{\min} = 10$, $D_{\max} = 120$; any fixed choices are acceptable as long as $D_{\min} \ge 1$ and times are integral.) Each accepted interval receives a stable ID $i \in \{1, \dots, m\}$ in order of creation; the table shown to the model lists rows by ID with times in `HH:MM`.

Uniqueness check by DP counting. Write $J = (1, \dots, n)$ for the intervals *sorted* by (f_i, s_i, i) (we reuse the symbol i for the sorted index when clear). Define the predecessor map

$$p(i) \;=\; \max\{\, j < i \;:\; f_j \le s_i \,\} \quad \text{with } p(i) = 0 \text{ if the set is empty.}$$

Let $\mathrm{opt}[i]$ be the maximum feasible cardinality using only $\{1, \dots, i\}$ and let $\mathrm{cnt}[i]$ be the *number of distinct schedules* that attain $\mathrm{opt}[i]$ using $\{1, \dots, i\}$. Initialize $\mathrm{opt}[0] = 0$ and $\mathrm{cnt}[0] = 1$ (one empty schedule). For $i = 1, \dots, n$ set

$$\begin{aligned}
\textsc{incl} &= 1 + \mathrm{opt}[\,p(i)\,], \\
\textsc{excl} &= \mathrm{opt}[\,i-1\,], \\
\mathrm{opt}[i] &= \max\{\textsc{incl}, \textsc{excl}\}, \\
\mathrm{cnt}[i] &= \begin{cases} \mathrm{cnt}[\,p(i)\,], & \text{if } \textsc{incl} > \textsc{excl}, \\ \mathrm{cnt}[\,i-1\,], & \text{if } \textsc{excl} > \textsc{incl}, \\ \mathrm{cnt}[\,p(i)\,] + \mathrm{cnt}[\,i-1\,], & \text{if } \textsc{incl} = \textsc{excl}. \end{cases}
\end{aligned}$$

Then $k^\star = \mathrm{opt}[n]$ is the optimal size and $\mathrm{cnt}[n]$ is the number of optimal schedules. If $\mathrm{cnt}[n] = 1$, the optimum is unique. Under uniqueness we reconstruct the unique optimal set $S^\star$ by backtracking from $i = n$: when $\textsc{incl} > \textsc{excl}$ we *include* i and jump to $p(i)$; when $\textsc{excl} > \textsc{incl}$ we *exclude* i and go to $i - 1$; when equal, exactly one branch leads to a nonzero count under uniqueness—choose the branch whose downstream count equals 1 (include if $\mathrm{cnt}[p(i)] = 1$ and $\mathrm{cnt}[i-1] = 0$, otherwise exclude).

Greedy sanity check and canonicalization. Because earliest-finish-time is optimal for unweighted interval scheduling, if the optimum is unique then the greedy policy with the same tie-breaks must return that same set. We nevertheless perform a *sanity check*: run a single pass of earliest-finish-time using the order (f_i, s_i, i) to obtain G; we accept the instance only if $G = S^\star$ (as sets). The ground-truth sequence we release is the unique set $S^\star$ *sorted by non-decreasing f (ties by i)* and the ground-truth answer is $|S^\star|$. This yields a single canonical `\ids{...}` and `\answer{...}` for each instance.

Complexity. Preprocessing and $p(i)$ by binary search take $O(m \log m)$; the DP pass and backtracking are $O(m)$; greedy verification is $O(m)$.

A.2 Longest Increasing Subsequence (Unique Optimum)

Problem model. Given integers $a_1, \dots, a_n$, an LIS is a subsequence $1 \le i_1 < \cdots < i_L \le n$ with $a_{i_1} < \cdots < a_{i_L}$. We allow duplicate values in (a_i); the strict inequality enforces strictly increasing

Algorithm 1: COUNTOPTIMAANDBACKTRACK for Interval Scheduling

Input: Intervals with IDs (i, s_i, f_i), $i = 1, \ldots, m$.
Output: Optimal size $k^\star$, number of optimal schedules $\mathrm{cnt}[n]$, unique set $S^\star$ if $\mathrm{cnt}[n] = 1$.
Sort intervals by (f_i, s_i, i) to obtain $J = (1, \ldots, n)$; precompute $p(i) = \max\{j < i : f_j \le s_i\}$ by binary search
$\mathrm{opt}[0] \leftarrow 0$, $\mathrm{cnt}[0] \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
  INCL $\leftarrow 1 + \mathrm{opt}[p(i)]$, EXCL $\leftarrow \mathrm{opt}[i-1]$
  $\mathrm{opt}[i] \leftarrow \max\{\text{INCL}, \text{EXCL}\}$
  if INCL $>$ EXCL **then** $\mathrm{cnt}[i] \leftarrow \mathrm{cnt}[p(i)]$
  else if EXCL $>$ INCL **then** $\mathrm{cnt}[i] \leftarrow \mathrm{cnt}[i-1]$
  else $\mathrm{cnt}[i] \leftarrow \mathrm{cnt}[p(i)] + \mathrm{cnt}[i-1]$
$k^\star \leftarrow \mathrm{opt}[n]$
if $\mathrm{cnt}[n] \neq 1$ **then return** $(k^\star, \mathrm{cnt}[n], \emptyset)$
`// Uniqueness holds: recover the single optimal set`
$S^\star \leftarrow \emptyset$; $i \leftarrow n$
while $i > 0$ **do**
  INCL $\leftarrow 1 + \mathrm{opt}[p(i)]$, EXCL $\leftarrow \mathrm{opt}[i-1]$
  if INCL $>$ EXCL **then**
    add i to $S^\star$; $i \leftarrow p(i)$
  else if EXCL $>$ INCL **then**
    $i \leftarrow i - 1$
  else
    if $\mathrm{cnt}[p(i)] = 1$ *and* $\mathrm{cnt}[i-1] = 0$ **then**
      add i to $S^\star$; $i \leftarrow p(i)$
    else
      $i \leftarrow i - 1$
return $(k^\star, 1, S^\star)$ `// Sort` $S^\star$ `by` (f, i) `when reporting.`

values. The canonical target we release per instance is: (i) the unique LIS *index* sequence listed in increasing row order as `\ids{...}`, and (ii) its length L as `\answer{L}`. *IDs are 1-based row indices*.

Sampling. We work on an integer grid. For each trial we: (i) draw a length $m \sim \mathrm{Unif}\{m_{\min}, \ldots, m_{\max}\}$; (ii) draw values i.i.d. $a_i \sim \mathrm{Unif}\{V_{\min}, \ldots, V_{\max}\}$; (iii) accept the instance only if the LIS is *unique* (by index sequence) and $L \ge 2$. Typical bounds used in our experiments are $m_{\min}$=5, $m_{\max}$=16, $V_{\min}$=1, $V_{\max}$=1000, but any fixed choices are valid.

Uniqueness check by $O(n^2)$ counting. Let `len_end`$[i]$ be the LIS length ending at i, and `cnt_end`$[i]$ the number of LIS that end at i with that maximum length. Initialize `len_end`$[i]$=1, `cnt_end`$[i]$=1. For each $i = 1..n$ and $j < i$:

$$\text{if } a_j < a_i: \quad \begin{cases} \text{if } \texttt{len\_end}[j]{+}1 > \texttt{len\_end}[i]: & \texttt{len\_end}[i] \leftarrow \texttt{len\_end}[j]{+}1,\ \texttt{cnt\_end}[i] \leftarrow \texttt{cnt\_end}[j] \\ \text{else if } \texttt{len\_end}[j]{+}1 = \texttt{len\_end}[i]: & \texttt{cnt\_end}[i] \leftarrow \texttt{cnt\_end}[i] + \texttt{cnt\_end}[j]. \end{cases}$$

Let $L = \max_i \texttt{len\_end}[i]$ and $\#\mathrm{LIS} = \sum_{i:\texttt{len\_end}[i]=L} \texttt{cnt\_end}[i]$. We declare *uniqueness* iff $\#\mathrm{LIS} = 1$.

Canonical reconstruction (used only after uniqueness holds). We reconstruct one LIS index sequence via patience sorting with predecessor links: scan left to right; for each a_i place it by lower_bound (first position $\ge a_i$) in the tails array; store for each i a predecessor pointer to the last index at the previous length; then backtrack from the final tail index to obtain indices in increasing order.

Algorithm 2: GENERATEUNIQUEACTIVITYINSTANCE

Input: RNG seed; $m_{\min}, m_{\max}$; minute-grid bounds $S_{\max}$, $D_{\min}, D_{\max}$; `max_tries`.
Output: Intervals $\{(i, s_i, f_i)\}_{i=1}^{m}$; canonical `\ids{}` sequence; integer answer $|S^\star|$.

for $t \leftarrow 1$ **to** *max_tries* **do**
  Sample $m \sim \mathrm{Unif}\{m_{\min}, \ldots, m_{\max}\}$
  $I \leftarrow \emptyset$
  while $|I| < m$ **do**
    sample $s \sim \mathrm{Unif}\{0, \ldots, S_{\max}\}$, $d \sim \mathrm{Unif}\{D_{\min}, \ldots, D_{\max}\}$, set $f \leftarrow s + d$
    if $f \leq S_{\max} + D_{\max}$ **then** append new interval $(|I|{+}1, s, f)$ to I
  $(k^\star, \mathrm{cnt}, S^\star) \leftarrow$ COUNTOPTIMAANDBACKTRACK(I)
  if $\mathrm{cnt} \neq 1$ **then continue**
  $G \leftarrow$ greedy earliest-finish schedule on I using (f, s, i) tie-breaks
  if $G \neq S^\star$ **then continue**
  // Canonical target: IDs of $S^\star$ sorted by (f, i); answer $= |S^\star|$
  return $(I,$ *\ids{ IDs*$(S^\star)$ *},* *\answer{* $|S^\star|$ *}*$)$

Fail if no instance is found within `max_tries` (try another seed or bounds)

Algorithm 3: COUNTLISLENGTHANDNUMBER (strict LIS length & count)

Input: Sequence $(a_1, \ldots, a_n) \in \mathbb{Z}^n$.
Output: L (LIS length), #LIS (number of LIS).

for $i \leftarrow 1$ **to** n **do**
  `len_end`$[i] \leftarrow 1$;
  `cnt_end`$[i] \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** $i{-}1$ **do**
    if $a_j < a_i$ **then**
      if *len_end*$[j]{+}1 >$ *len_end*$[i]$ **then**
        `len_end`$[i] \leftarrow$ `len_end`$[j]{+}1$;
        `cnt_end`$[i] \leftarrow$ `cnt_end`$[j]$
      else if *len_end*$[j]{+}1 =$ *len_end*$[i]$ **then**
        `cnt_end`$[i] \leftarrow$ `cnt_end`$[i]$ + `cnt_end`$[j]$

$L \leftarrow \max_i$ `len_end`$[i]$; #LIS $\leftarrow \sum_{i:\texttt{len\_end}[i]=L}$ `cnt_end`$[i]$
return $(L, \#\mathrm{LIS})$ // Runtime $O(n^2)$

Canonical reporting and prompts. Because the LIS is a subsequence, the ground-truth `\ids{...}` is simply the unique LIS indices $(i_1 < \cdots < i_L)$ in increasing row order; `\answer{L}` reports its length.

Complexity. Per trial: $O(n^2)$ for counting uniqueness; $O(n \log n)$ to reconstruct the indices once unique. Overall generation is bounded by `max_tries`, which we set large enough so failures are rare.

B Extracting sorted ID lists from free-form responses

Setup. For each Activity instance the prompt shows an ASCII table with unique integer IDs $i \in \{1, \ldots, n\}$ and times (s_i, f_i). We define the ground-truth sorted order

$$B = (b_1, \ldots, b_n) \;:=\; \text{IDs sorted by } (f_i, i) \text{ (non-decreasing end time, ties by smaller ID).}$$

This B is the order used by the greedy earliest-finish-time scheduler (Alg. 1, §3.1). From the model's free-form reply r (arbitrary text) we attempt to recover a list the model claims to be "sorted."

Candidate sources (lightweight and order-preserving). We build up to four candidate ID sequences from r; all candidates are *normalized* by (i) filtering to the valid ID set $\mathcal{I} = \{1, \ldots, n\}$

Algorithm 4: GENERATEUNIQUELISINSTANCE

Input: RNG seed; $m_{\min}, m_{\max}$; value bounds $V_{\min}, V_{\max}$; `max_tries`.
Output: Values $(a_1, \ldots, a_m)$; canonical `\ids{·}`; `\answer{·}`.
for $t \leftarrow 1$ **to** *`max_tries`* **do**
  Sample $m \sim \mathrm{Unif}\{m_{\min}, \ldots, m_{\max}\}$; Sample $a_i \sim \mathrm{Unif}\{V_{\min}, \ldots, V_{\max}\}$ i.i.d. for i=1..m
  $(L, \#) \leftarrow$ COUNTLISLENGTHANDNUMBER$(a_{1:m})$
  if $L < 2$ ***or*** $\# \neq 1$ **then continue**
  `// Reconstruct unique LIS indices via patience sorting with predecessors`
  $(i_1, \ldots, i_L) \leftarrow$ PATIENCERECONSTRUCT$(a_{1:m})$ **return** $\big(a_{1:m},$ *`\ids`*$\{i_1, \ldots, i_L\},$ *`\answer`*$\{L\}\big)$
Fail if no unique instance within `max_tries` (try new seed or bounds)

Activity Scheduling Prompt

Determine the largest subset of activities that can be scheduled without any overlaps (a single resource is available, so no double-booking).

ID	Start	End
1	06:09	07:24
2	07:13	08:23
3	07:29	09:28
4	08:24	10:18
5	04:48	06:14

Instructions:

- Times are given in 24-hour HH:MM format.
- Non-overlap means an activity ending at time T is compatible with one starting at T.
- Select a maximum-size subset of non-overlapping rows.
- Your output must include the following two lines at the end, in this exact format:
  1. `\ids{<comma-separated IDs of the chosen rows, listed in order of increasing END time. If two rows have the same END time, put the smaller ID first>}`
  2. `\answer{<number of chosen rows>}`
- No spaces inside `\ids{...}`. Example: `\ids{3,9,12}`

Hint: Sort the rows by increasing end time, then greedily pick compatible rows.

Ground truth: `\ids{5,2,4}`, `\answer{3}`

LIS Prompt

Determine the longest strictly increasing subsequence (by VALUE) from the rows below (use the row IDs).

ID	Value
1	797
2	476
3	335
4	452
5	606

Instructions:

- The table lists 1-based row IDs and integer values.
- A valid subsequence must be strictly increasing by VALUE and preserve the original row order.
- Select a maximum-size subsequence.
- Your output must include the following two lines at the end, in this exact format:
  1. `\ids{<comma-separated IDs of the chosen rows, listed in increasing ROW order>}`
  2. `\answer{<number of chosen rows>}`
- No spaces inside `\ids{...}`. Example: `\ids{3,9,12}`

Hint: Use LIS DP (len[i] = 1 + max len[j] for $j < i$ with value[j]<value[i]) while storing prev[i] (or patience sorting with predecessor links); then backtrack to output the ids.

Ground truth: `\ids{3,4,5}`, `\answer{3}`

Figure 5: Example prompts and ground-truth for Activity Scheduling (left) and LIS (right).

derived from the prompt table, and (ii) de-duplicating while preserving the *first* occurrence of each ID in the text.

1. **Sorted-block (for exact-sorting only).** We split r into paragraphs and select those that mention a sorting token ("sort/sorted/sorting"). To isolate the purported sorting step, we truncate at the first stop word (any of: `select, greedy, choose, subset, largest, final answer, so, thus, therefore, next`). We then extract IDs from this truncated text via either "`ID` k" tokens or the longest comma-separated integer run. If the resulting list contains *all* n distinct IDs exactly once, we accept it as a full "sorted-block" candidate A_{full} and reserve it solely for the exact-sorting check below.
2. `\ids{...}` **blocks.** From every brace block we read the comma-separated integers, normalize, and keep the resulting sequence if non-empty.

3. **"ID k" token stream.** We scan r left-to-right for tokens of the form "`ID` k" and record the first occurrence of each k.
4. **Longest comma run.** We find the longest comma-separated integer run anywhere in r and normalize it.

These patterns capture the most common surface forms we observe in practice and are exactly those used in our implementation.

Exact-sorting criterion. If the sorted-block candidate A_{full} exists and is a permutation of all n IDs (after normalization), we declare *extraction success*. The instance counts as *exactly sorted* iff $A_{\text{full}} = B$. Missing or malformed cases are treated as incorrect in the exact-sorting accuracy.

Best-of-candidates policy and anchors (for substring analysis). For contiguous-substring analysis (§C) we consider *all* non-empty candidates $\mathcal{A} = \{A_1, \ldots, A_K\}$ formed by items 2–4 above (and A_{full} if present). When multiple candidates tie on the score defined next, we break ties by method priority:

$$\texttt{sorted\_block\_full} \prec \texttt{ids\_braces} \prec \texttt{id\_stream} \prec \texttt{comma\_run}.$$

For the chosen best candidate we additionally report an *anchor* label describing where the matched block sits inside the candidate: `start` (run begins at position 1), `end` (run ends at position $|A_k|$), `both` (the candidate equals the block), or `neither`.

Safeguards and edge cases. We ignore any integers not present in the prompt table; repeated IDs are dropped after the first mention; candidates that normalize to the empty list are discarded. These choices make the procedure robust to extraneous numbers and minor formatting quirks without conferring credit for unseen IDs.

Complexity. Regex scans and candidate construction are $O(|r|)$; all subsequent scoring (§C) is linear in the candidate length(s).

C Contiguous LCS (longest common substring) metric

Definition. Given a candidate $A = (a_1, \ldots, a_m)$ and the ground-truth order $B = (b_1, \ldots, b_n)$ (IDs unique), define the position map $\text{pos}(b_\ell) = \ell$. Let $p_t = \text{pos}(a_t)$ for $t = 1, \ldots, m$ (these indices exist by construction after normalization). The *contiguous* LCS length between A and B is

$$\text{LCS\_len}(A, B) \;=\; \max_{1 \le i \le j \le m} \Big\{ (j - i + 1) \;\Big|\; \exists\, t \text{ s.t. } \forall\, u \in \{0, \ldots, j - i\},\; p_{i+u} = t + u \Big\}. \quad (1)$$

Intuitively, (1) is the length of the longest block in A that appears *contiguously* in B; because IDs in B are unique, this is exactly the standard longest common *substring*.

Instance score and reporting. From the set of extracted candidates $\mathcal{A} = \{A_1, \ldots, A_K\}$ we take the best match

$$L^\star \;=\; \max_{k \in [K]} \text{LCS\_len}(A_k, B), \qquad \text{LCS-frac} \;=\; \frac{L^\star}{|B|} \;=\; \frac{L^\star}{n}.$$

We report the mean LCS-fraction over instances with $L^\star > 0$ and a separate *coverage* rate (% of instances with any non-empty match). Exact-sorting accuracy and LCS are shown in Fig. 8.

Anchors. For the candidate achieving $L^\star$ (after the tie-break above), we also emit the anchor label `start/end/both/neither` based on whether the best contiguous block touches the candidate's boundaries. Anchors help diagnose whether the model's claimed "sorted" list appears as a leading or trailing segment versus a mid-sequence fragment.

D LIS Task Response Length and Entropy

Figure 6 shows response length during training on the LIS task. Under the answer-only reward, responses quickly shorten, while the format reward preserves lengths close to those of the base model.

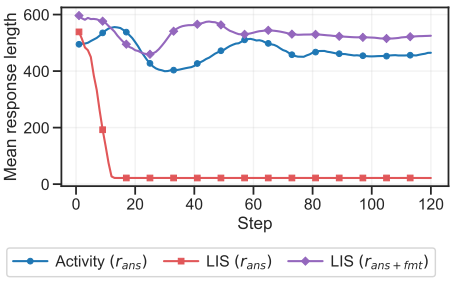

Figure 6: Response length during training

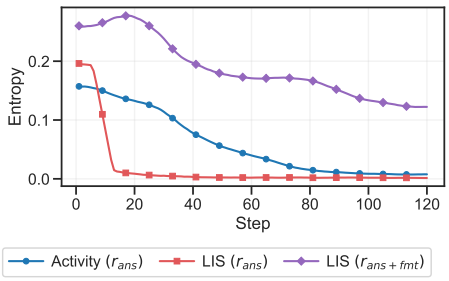

Figure 7: Entropy during training

Figure 7 further compares entropy during training. Entropy drops sharply for r_{ans}, mirroring the collapse in response length, whereas $r_{\text{ans+fmt}}$ maintains higher entropy throughout training. Despite these qualitative differences, both RLVR-trained policies converge to similar levels of Acc_{ans} and Acc_{ids}, suggesting reliance on comparable heuristics expressed through different generation styles. For example, with $r_{\text{ans+fmt}}$ nearly all responses contain Python code (100%, compared to 35.1% for the base model), yet the model does not execute the code step by step to derive the answer. This observation echoes findings by Shao et al. [7], who show that RLVR can amplify spurious reward signals in the Qwen family, encouraging models to emit structured but ultimately superficial outputs.

E Random-Forest Regression for LIS: Features, Protocol, and Training

Goal. We analyze the model's implicit decision rule by regressing its numeric answer, $y =$ `pred_lis_len`, on interpretable features computed *only* from the input values $(v_1, \ldots, v_n)$.

Data and target. From the JSONL logs we pool all stochastic runs k for each instance (`sample_idx`). The regression target is the model's emitted `\answer{\cdot}` value; no ground-truth labels are used as features.

Group split to avoid leakage. We perform a single group hold-out split with GroupShuffleSplit (test size $\approx 25\%$) using `sample_idx` as the group key. Thus, all k replicates of a given instance are kept together in train or test; there is no overlap of `sample_idx` between splits.

Feature set (input-only). Let n be the sequence length and $\Delta_i = v_{i+1} - v_i$ for $i = 1, \ldots, n-1$. We compute:

- **Global scale/dispersion:** n, $\min(v)$, $\max(v)$, range $= \max - \min$, mean, std, quartiles q_{25}, q_{50}, q_{75}, `uniq_ratio` $= |\{v\}|/n$, and `dup_ratio` $= 1 -$ `uniq_ratio`.
- **Adjacent order (local trend):** `adj_inc_ratio` $= \frac{1}{n-1}\sum \mathbb{1}[\Delta_i > 0]$, `adj_dec_ratio` $= \frac{1}{n-1}\sum \mathbb{1}[\Delta_i < 0]$, `adj_eq_ratio` $= \frac{1}{n-1}\sum \mathbb{1}[\Delta_i = 0]$, `pos_delta_mean/std` on $\{\Delta_i > 0\}$, `neg_delta_mean/std` on $\{\Delta_i < 0\}$, and `sign_change_ratio` = fraction of sign flips in $(\Delta_1, \ldots, \Delta_{n-1})$.
- **Pairwise order (global monotonicity):** `pair_inc_ratio` $= \frac{1}{\binom{n}{2}}|\{i < j : v_j > v_i\}|$, `inversion_ratio` $= \frac{1}{\binom{n}{2}}|\{i < j : v_j < v_i\}|$, `tau_like` $=$ `pair_inc_ratio` $-$ `inversion_ratio` (Kendall-tau proxy ignoring ties).
- **Runs and structure:** `max_inc_run` = longest strictly increasing contiguous run, `max_dec_run` = longest strictly decreasing run, `num_monotone_runs` = number of contiguous monotone segments, `n_local_max/min` = counts of strict local maxima/minima, `record_highs` (new maxima count), `record_lows` (new minima count).
- **Heuristic LIS approximations (length only):**
  - `greedy_len`: left-to-right "append if v_i increases" record-high count.
  - `greedy_rev_len`: same on the reversed sequence.

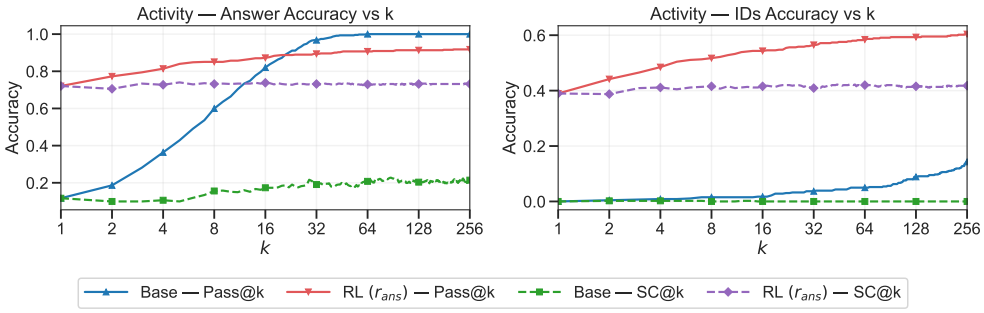

Figure 8: Performance comparison of Base and RLVR(r_{ans}) on the *Activity* task with the Llama-3.1-8B model. Left: numeric answer accuracy (Acc_{ans}) vs. k. Right: ID sequence accuracy (Acc_{ids}) vs. k.

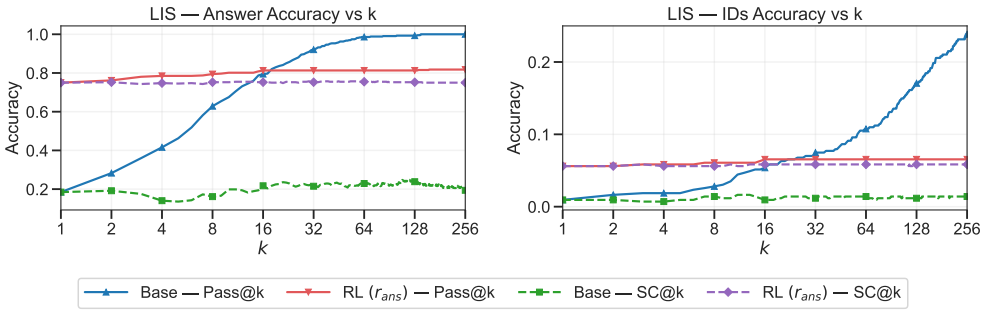

Figure 9: Performance comparison of Base and RLVR(r_{ans}) on the *LIS* task with the Llama-3.1-8B model. Left: numeric answer accuracy (Acc_{ans}) vs. k. Right: ID sequence accuracy (Acc_{ids}) vs. k.

  - `beam2`, `beam3`: beam-limited LIS lengths with beam $B \in \{2, 3\}$.
  - `budget1`, `budget2`: greedy with $s \in \{1, 2\}$ backtracks (replace tail and truncate at most s times).
- **Patience-sorting descriptors (no direct LIS leakage):** From the patience *tails* vector t, use `tail_mean`, `tail_std`, `tail_iqr`, and `tail_slope` (OLS slope of t vs. index).
- **Reference baseline:** `rand_lis_baseline` $= 2\sqrt{n}$ (typical LIS scale under random permutations).

Pre-processing. We replace $\pm\infty$ with NaN and drop rows with missing feature values. Metadata such as k or `log2_k` are never used. Standardization is unnecessary for tree ensembles.

Model and hyperparameters. We use a Random-Forest Regressor with `n_estimators`=800, `min_samples_leaf`=2, `max_features=sqrt`, `random_state`=42, and `n_jobs`=-1.

Evaluation and Top-K selection. We report R^2 and MAE on the held-out group-split test fold. To obtain a compact, interpretable subset, we rank features by RF importance (fit on train) and sweep $K \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 18, 20, 25, \ldots\}$. We pick the smallest K whose test performance is within $(\Delta R^2, \Delta\text{MAE}) = (0.01, 0.02)$ of the full model. We also provide (i) a log-scaled histogram of test residuals and (ii) a Top-K curve (test R^2 and MAE vs. K).

F Llama Model Performance

Figure 8 reports results on the *Activity Scheduling* task, and Figure 9 shows results on *LIS* using the *Llama-3.1-8B* model. The LLaMA model attains higher overall accuracy on both tasks, as measured by SC@256. However, the relative trends between the base and RLVR-trained variants (e.g., under r_{ans}) closely mirror those observed with the Qwen model family.

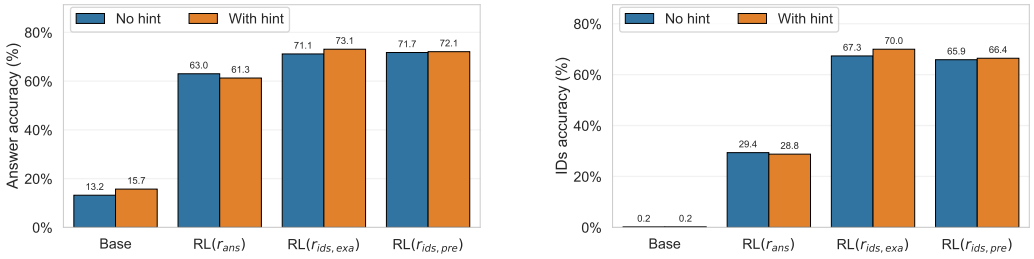

Figure 10: Acc with/without hint for *Activity*.

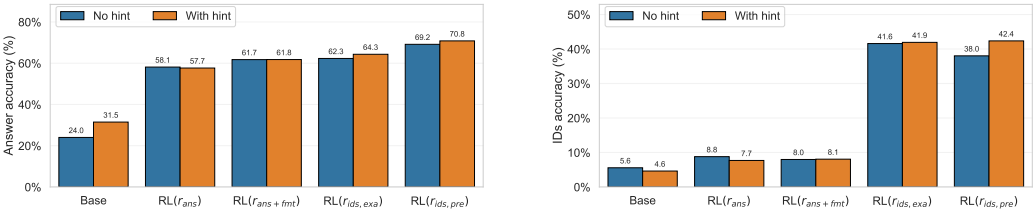

Figure 11: Acc with/without hint for *LIS*.

G Evaluation of Hinted vs. Unhinted Prompts

Figure 10 (Activity) and Figure 11 (LIS) compare model performance on test prompts with and without hints. Across both tasks, we observe no significant performance differences between the hinted and unhinted variants, suggesting that the models do not substantially benefit from the additional guidance provided by hints.

G.1 Heuristics Analysis for LIS

To probe what signals models exploit on *LIS*, we regress predicted numeric answers against human-interpretable features from the input table.

We pool all k stochastic runs but split train/test by problem instance ID to avoid leakage. Features cover: (i) global scale (length, range, dispersion, quantiles), (ii) order structure (increase ratios, inversion ratio, sign changes), (iii) run structure (longest runs, monotone counts, local extrema, record highs/lows), (iv) simple LIS heuristics (greedy, beam-limited, limited backtracking), and (v) patience-sorting tails. We train a Random Forest regressor and report R^2 and mean absolute error (MAE). Details appear in Appendix E.

Model	R^2_{test}	MAE_{test}
Base	-0.002	2.745
r_{ans}	0.741	0.269
$r_{\text{ids,exa}}$	0.781	0.289
$r_{\text{ids,pre}}$	0.841	0.227
$r_{\text{ans+fmt}}$	0.867	0.209

Table 1: Predictive fit of LIS features on model outputs.

Table 1 shows that RLVR-trained outputs are predicted well from these features, unlike the base model. While the features are not exhaustive, results suggest RLVR amplifies systematic heuristics aligned with task structure, boosting answer-level performance without ensuring stronger reasoning.

H Training with Sorting-Match Reward.

We investigated whether directly rewarding the sorting step, the first stage of the greedy activity-scheduling algorithm, could yield additional benefits by training with a sorting-match reward r_{sort}. This approach backfired: both Acc_{ans} and Acc_{ids} collapsed to nearly 0%, as the model simply output the sorted sequence without enforcing the non-overlap constraint. A combined objective with equal weights on r_{ans}, r_{ids}, and r_{sort} restored performance to the level of r_{ids} alone but did not improve sorting on the test set, suggesting that sorting in isolation provides no complementary benefit. Curriculum experiments confirmed this: pretraining with r_{sort} for 10 PPO steps allowed recovery under r_{ans}, but

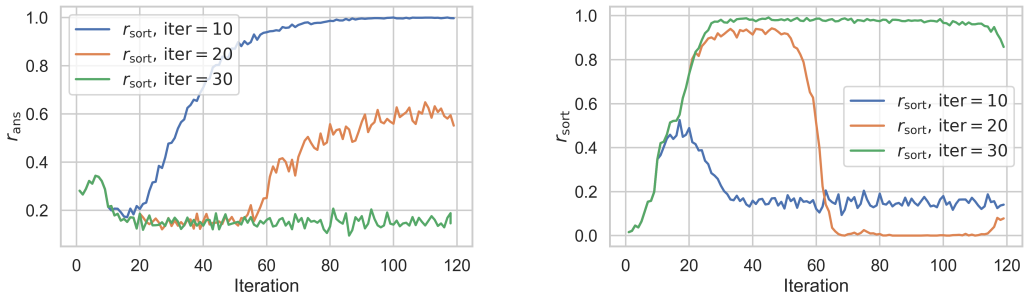

Figure 12: Curriculum experiments with r_{sort}. Each curve shows accuracy on the training set when models are trained with r_{sort} for the first 10, 20, or 30 PPO updates, followed by r_{ans} for the remainder. Longer pretraining with r_{sort} makes it increasingly difficult for the model to recover under r_{ans}.

20 or 30 steps severely hindered it, with 30-step pretraining preventing any improvement even on the training set (Figure 12).