Towards Understanding Self-play for LLM Reasoning

Justin Yang Chae
University of Washington
Seattle, WA, USA
jchae3@uw.edu

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

Recent advances in large language model (LLM) reasoning, led by reinforcement learning with verifiable rewards (RLVR), have inspired self-play post-training, where models improve by generating and solving their own problems. While self-play has shown strong in-domain and out-of-domain gains, the mechanisms behind these improvements remain poorly understood. In this work, we analyze the training dynamics of self-play through the lens of the *Absolute Zero Reasoner*, comparing it against RLVR and supervised fine-tuning (SFT). Our study examines parameter update sparsity, entropy dynamics of token distributions, and alternative proposer reward functions. We further connect these dynamics to reasoning performance using pass@k evaluations. Together, our findings clarify how self-play differs from other post-training strategies, highlight its inherent limitations, and point toward future directions for improving LLM math reasoning through self-play.

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) has emerged as the leading method for improving reasoning in large language models (LLMs), with notable successes in mathematics and other verifiable domains [21, 12]. However, its benefits remain debated: recent studies argue that RLVR primarily sharpens output distributions and improves sampling efficiency rather than fostering genuine reasoning [31, 7]. Analyses of its training dynamics further show reduced policy entropy and sparser parameter updates relative to supervised fine-tuning [6, 20].

Beyond RLVR, recent work has investigated the potential of self-play, an alternative paradigm where models interact with themselves to drive improvement. For example, Zhao et al. [32] introduced a framework where an LLM acts as both a proposer of coding problems and a solver, achieving significant gains on out-of-domain mathematics benchmarks using only its own coding outputs and without additional human-curated data. Similarly, other methods have shown the promise of self-play for LLMs [15, 11, 13]. The success of these methods highlights a critical need to understand how models can improve without external supervision and whether these improvements reflect genuine gains in reasoning ability.

In this work, we provide an analysis of the self-play approach, comparing its training dynamics and performance against RLVR and SFT. We follow the *pass@k* experiments of Yue et al. [31] to rigorously assess whether self-play-trained models achieve stronger reasoning performance than

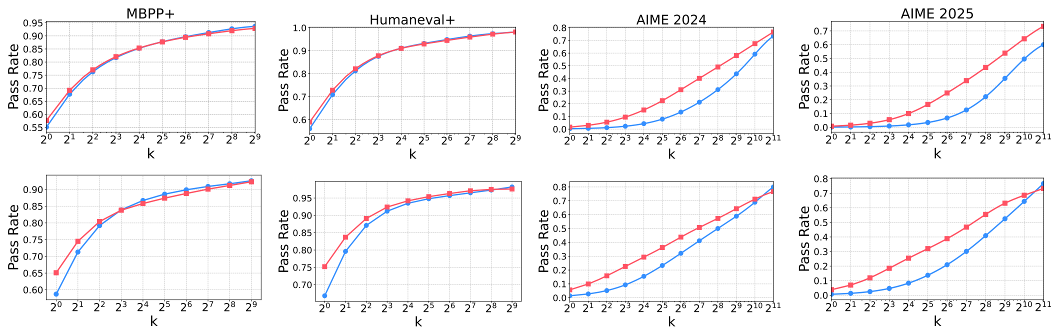

Figure 1: **Self-play models are still bounded by the base model.** (Top row) Pass@k curves for AZR-CODER-3B (red) and QWEN2.5-CODER-3B (blue). (Bottom row) Pass@k curves for AZR-CODER-7B (red) and QWEN2.5-CODER-7B (blue).

their base counterparts. We study internal training dynamics through measures of policy entropy and parameter update sparsity to shed light on how self-play shapes model behavior. Finally, we investigate the role of the proposer component by modifying its reward function and we also track how the distribution and difficulty of generated questions evolve throughout training.

2 Absolute Zero Reasoner

The *Absolute Zero* paradigm [32] is a self-play framework where a single model learns by simultaneously proposing and solving tasks. One implementation, the *Absolute Zero Reasoner* (AZR), uses a unified LLM that acts as both a **proposer**, which generates a curriculum of tasks, and a **solver**, which learns by solving them. After an environment validates each proposed task to create a problem with a gold solution, the model is jointly optimized using a multitask advantage estimator, receiving rewards for both task quality and solution accuracy. AZR is trained on three families of coding tasks designed to capture abductive, deductive, and inductive reasoning; further implementation details are available in Appendix A.

3 Experiments

We adopt the **Absolute Zero Reasoner** (AZR) [32] codebase as our self-play framework. Due to computational constraints, we only conduct experiments on QWEN2.5-CODER-3B and QWEN2.5-CODER-7B.

We investigate the following research questions:

RQ1: Does self-play in AZR enable novel reasoning beyond the capacity of the base model?

RQ2: How does the difficulty of the questions proposed evolve over the course of training?

RQ3: Does AZR experience entropy collapse during training like RLVR-trained models?

RQ4: How does parameter update sparsity in self-play compare with RLVR and SFT?

RQ5: Does lowering the proposer's target question difficulty improve model performance?

3.1 RQ1: Reasoning Capacity

Setup. We use an unbiased pass@k estimator [2] to measure reasoning capacity, following the experimental setup of Yue et al. [31] on benchmarks including MBPP+, HumanEval+, and AIME.

Results. We show that self-play improves performance at small k, but the base model performs better at large k (Figure 1). This suggests gains from distributional sharpening [31, 7]. However, the performance drop at large k is not statistically significant, suggesting it preserves the base model's capacity more effectively than standard RLVR. We hypothesize this is due to implicit data diversity from the co-evolutionary training, a factor suggested to sustain pass@k gains in Liang et al. [13].

We provide a formal argument for why AZR remains bounded by the reasoning capacity of the base model, building on the analysis of Wu et al. [28], in Appendix C.

Takeaway

AZR is effectively RLVR over a proposer-induced task distribution [32], and thus inherits the same Invisible-Leash support limitation [28]: self-play cannot assign probability mass to solutions outside the base model's support. As a result, AZR remains bounded by the reasoning capacity of the base model, though unlike RLVR, the drop in pass@k performance at larger k is less pronounced.

3.2 RQ2: Evolution of Question Difficulty

Setup. To study how question difficulty evolves, we created a balanced dataset of 800 deductive questions sampled from different training iterations. For each question, we generated 8 responses using AZR-CODER-7B and QWEN2.5-CODER-7B and measured the average response length and solve rate (denoted $\bar{r}_{\text{solve}}$ in Equation 4).

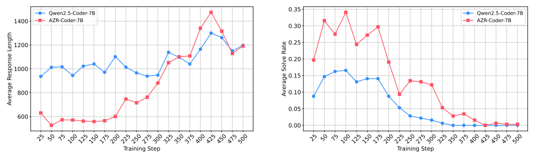

Figure 2: **AZR-CODER-7B adapts response length to question difficulty while QWEN2.5-CODER-7B does so at a lesser scale.** (Left) Average response length at every 25th iteration. (Right) Average solve rate at every 25th iteration.

Results. As shown in Figure 2, the proposer generates more difficult questions over time, which in turn elicit longer responses, a trait associated with improved reasoning [8, 3]. AZR produces shorter responses for easier questions, consistent with the idea of an optimal response length that avoids under or overthinking [29, 24]. This suggests self-play may teach the model to adjust its response length based on the perceived difficulty of a question.

Takeaway 2

The proposer in AZR self-play generates increasingly difficult questions, and AZR appears to adapt response lengths accordingly. This implicit sensitivity to difficulty may help the model avoid both underthinking and overthinking.

3.3 RQ3: Entropy Collapse

Setup. Prior work has shown that RLVR post-training causes an *entropy collapse*, where a model effectively trades exploration for validation accuracy [6]. We examine whether self-play exhibits a

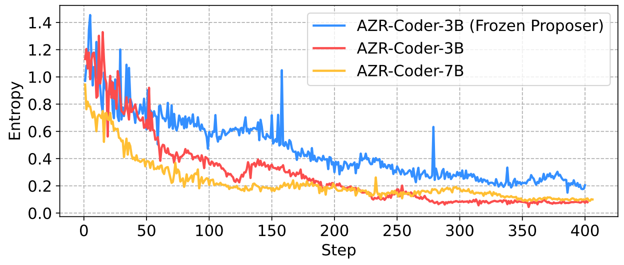

Figure 3: **Policy entropy decays at different rates based on model size and setup.** Policy entropy curves for AZR-CODER-3B, AZR-CODER-3B with a frozen proposer, and AZR-CODER-7B.

similar dynamic by tracking policy entropy, defined as:

$$H(\pi_\theta, D) = -\mathbb{E}_{x\sim D, y\sim\pi_\theta}[\log \pi_\theta(y_t|y_{<t}, x)] = -\frac{1}{|D|}\sum_{x\in D}\frac{1}{|y_x|}\sum_{t=1}^{|y_x|}\log \pi_\theta(y_t|y_{<t}, x) \quad (1)$$

Results. As shown in Figure 3, self-play also leads to entropy collapse, with the decay rate depending on model size and setup. The frozen-proposer variant (no PPO updates in the proposer role) maintains higher entropy than the standard 3B model, likely because in standard self-play, both proposer and solver receive gradient updates, doubling optimization pressure and accelerating collapse. When decomposed by role (Figure 4), proposer entropy consistently remains higher than solver entropy. Although overall entropy decays, this suggests that encouraging proposer diversity could increase entropy and improve performance.

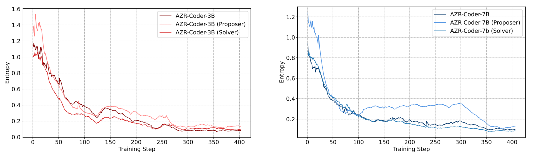

Figure 4: **Proposer entropy stays higher than solver entropy.** Proposer, solver, and policy entropy curves for AZR-CODER-3B (left) and AZR-CODER-7B (right)

Takeaway 3

Self-play with AZR exhibits entropy collapse, with decay rates varying by model size and proposer setup. Sustaining exploration may require explicit entropy regularization or mechanisms for promoting diverse proposer outputs.

3.4 RQ4: Parameter Update Sparsity

Setup. Following Mukherjee et al. [20], who showed that supervised fine-tuning (SFT) produces dense parameter updates while reinforcement learning (RL) is sparse, we analyze the **update sparsity** of self-play. This metric measures the proportion of parameters unchanged between two checkpoints:

$$S(\theta^0, \theta^1) := 1 - \frac{\|\theta^1 - \theta^0\|_0}{n}, \quad (2)$$

where $\|\cdot\|_0$ counts non-zero elements. We compare our self-play models (AZR-CODER) against public SFT and RLVR checkpoints that use the same base model.

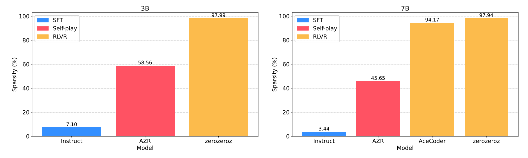

Figure 5: **Self-play has distinct update sparsity compared to RL-tuned and SFT models.** Update sparsity comparison between public checkpoints of fine-tuned models and their corresponding base models: (left) QWEN2.5-CODER-3B and (right) QWEN2.5-CODER-7B.

Results. Figure 5 shows that self-play produces an intermediate level of parameter update sparsity, denser than RLVR but sparser than SFT. For example, AZR-CODER-7B reaches about 45.7% sparsity. This pattern likely reflects the dual nature of self-play, where the model both generates new data and learns from solving it, leading to updates that are partly in-distribution yet still exploratory.

Takeaway 4

AZR self-play leads to intermediate update sparsity: denser than RLVR but sparser than SFT, reflecting its dual role of data generation and solution.

3.5 RQ5: Proposer Reward Function

Setup. Prior work suggests that a solve rate near 0.5 yields strong gradient signals in RLVR with GRPO-style advantage calculations [23, 30]. Building on this, we test a modified reward to explicitly encourage the proposer to generate questions of this difficulty:

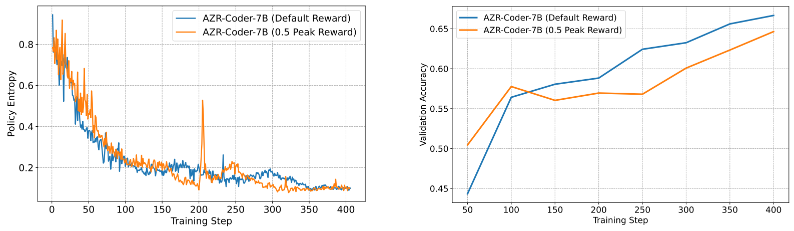

Figure 6: **Training dynamics of different proposer rewards.** (Left) Policy entropy training dynamics. (Right) Validation accuracy training dynamics

$$r_{\text{propose}} = \begin{cases} 0 & \text{if } \bar{r}_{\text{solve}} \in \{0, 1\}, \\ 1 - 2\left|\bar{r}_{\text{solve}} - 0.5\right| & \text{otherwise.} \end{cases} \tag{3}$$

This function gives the maximum reward when the average solve rate, $\bar{r}_{\text{solve}}$, is 0.5, incentivizing the proposer to generate questions that are neither too easy nor too hard.

Results. As shown in Figure 6, this reward function had little effect on entropy dynamics and reduced final validation accuracy by 2%. While Huang et al. [11] used it successfully in *R-Zero*, our results suggest their gains stem from other system components.

Takeaway 5

Altering the proposer's reward to target a 50% solve rate does not improve AZR performance and slightly reduces validation accuracy. This suggests that reward shaping alone is insufficient, with other components playing a larger role in AZR's self-play gains.

4 Limitations & Future Work

Our study is limited in scope to a single self-play framework (AZR) and two model sizes (3B and 7B). The patterns we observe with AZR may not hold across other self-play frameworks or larger models [11, 14]. Nevertheless, given the highly similar proposer–solver structures that most recent self-play methods adopt, we expect that our findings are broadly informative.

There remain several promising directions for future research. First, self-play is still bounded by the base model's reasoning capacity, motivating approaches that can expand reasoning support or inject probability mass into novel trajectories. Second, automatic curriculum learning within self-play could elicit new capabilities if designed effectively. Third, given the distinctive parameter update sparsity observed, it is worth investigating whether self-play mitigates or exacerbates catastrophic forgetting, an effect linked to sparsity in prior RL studies [22, 20]. Finally, preventing entropy collapse remains a key challenge; future work could explore proposer modifications or explicit entropy regularization to sustain exploration during training.

5 Conclusion

Our analysis of the self-play framework AZR reveals that while its reasoning capacity is bounded by the base model, the proposer is the critical component for improvement. The proposer drives performance by generating a diverse and progressively difficult curriculum of questions. We find that self-play exhibits a unique parameter update sparsity and still undergoes entropy collapse, pointing to the proposer as the most promising target for future work. Finally, self-play shows distinctive parameter update sparsity, raising questions about its relation to catastrophic forgetting and how proposer rewards can be refined to provide stronger learning signals.

Acknowledgment

This material is based upon work supported by the National Science Foundation Award #2447631.

References

[1] Lili Chen, Mihir Prabhudesai, Katerina Fragkiadaki, Hao Liu, and Deepak Pathak. 2025. Self-Questioning Language Models. `https://doi.org/10.48550/arXiv.2508.03682` arXiv:2508.03682 [cs].

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[3] Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567* (2025).

[4] Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. 2025. Reasoning with exploration: An entropy perspective. *arXiv preprint arXiv:2506.14758* (2025).

[5] Jiale Cheng, Xiao Liu, Cunxiang Wang, Xiaotao Gu, Yida Lu, Dan Zhang, Yuxiao Dong, Jie Tang, Hongning Wang, and Minlie Huang. 2025. SPaR: Self-Play with Tree-Search Refinement to Improve Instruction-Following in Large Language Models. `https://doi.org/10.48550/arXiv.2412.11605` arXiv:2412.11605 [cs].

[6] Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. 2025. The Entropy Mechanism of Reinforcement Learning for Reasoning Language Models. arXiv:2505.22617 [cs.LG] `https://arxiv.org/abs/2505.22617`

[7] Xingyu Dang, Christina Baek, J Zico Kolter, and Aditi Raghunathan. 2025. Assessing diversity collapse in reasoning. In *Scaling Self-Improving Foundation Models without Human Supervision*.

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

Zhang. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[9] Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. 2024. Self-play with Execution Feedback: Improving Instruction-following Capabilities of Large Language Models. https://doi.org/10.48550/arXiv.2406.13542 arXiv:2406.13542 [cs].

[10] Google DeepMind. 2024. AI achieves silver-medal standard solving International Mathematical Olympiad problems. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/. Accessed: 2025-09-27.

[11] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. 2025. R-Zero: Self-Evolving Reasoning LLM from Zero Data. https://doi.org/10.48550/arXiv.2508.05004 arXiv:2508.05004 [cs].

[12] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. 2025. Tulu 3: Pushing Frontiers in Open Language Model Post-Training. arXiv:2411.15124 [cs.CL] https://arxiv.org/abs/2411.15124

[13] Xiao Liang, Zhongzhi Li, Yeyun Gong, Yelong Shen, Ying Nian Wu, Zhijiang Guo, and Weizhu Chen. 2025. Beyond Pass@1: Self-Play with Variational Problem Synthesis Sustains RLVR. https://doi.org/10.48550/arXiv.2508.14029 arXiv:2508.14029 [cs].

[14] Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, et al. 2025. SPIRAL: Self-Play on Zero-Sum Games Incentivizes Reasoning via Multi-Agent Multi-Turn Reinforcement Learning. *arXiv preprint arXiv:2506.24119* (2025).

[15] Mickel Liu, Liwei Jiang, Yancheng Liang, Simon Shaolei Du, Yejin Choi, Tim Althoff, and Natasha Jaques. 2025. Chasing Moving Targets with Online Self-Play Reinforcement Learning for Safer Language Models. *arXiv preprint arXiv:2506.07468* (2025).

[16] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding R1-Zero-Like Training: A Critical Perspective. https://openreview.net/forum?id=jLpC1zavzn

[17] Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, Shengyi Huang, Siran Yang, Jiamang Wang, Wenbo Su, and Bo Zheng. 2025. Part I: Tricks or Traps? A Deep Dive into RL for LLM Reasoning. https://doi.org/10.48550/arXiv.2508.08221 arXiv:2508.08221 [cs].

[18] Zihan Liu, Zhuolin Yang, Yang Chen, Chankyu Lee, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2025. AceReason-Nemotron 1.1: Advancing Math and Code Reasoning through SFT and RL Synergy. arXiv:2506.13284 [cs.CL] https://arxiv.org/abs/2506.13284

[19] Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 625–635. https://doi.org/10.1007/978-3-030-79876-5_37

[20] Sagnik Mukherjee, Lifan Yuan, Dilek Hakkani-Tur, and Hao Peng. 2025. Reinforcement Learning Finetunes Small Subnetworks in Large Language Models. *arXiv preprint arXiv:2505.11711* (2025).

[21] 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).

[22] Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. 2025. RL's Razor: Why Online Reinforcement Learning Forgets Less. *arXiv preprint arXiv:2509.04259* (2025).

[23] Taiwei Shi, Yiyang Wu, Linxin Song, Tianyi Zhou, and Jieyu Zhao. 2025. Efficient Reinforcement Finetuning via Adaptive Curriculum Learning. `https://doi.org/10.48550/arXiv.2504.05520` arXiv:2504.05520 [cs] version: 2.

[24] Jinyan Su, Jennifer Healey, Preslav Nakov, and Claire Cardie. 2025. Between underthinking and overthinking: An empirical study of reasoning length and correctness in llms. *arXiv preprint arXiv:2505.00127* (2025).

[25] Yifan Sun, Jingyan Shen, Yibin Wang, Tianyu Chen, Zhendong Wang, Mingyuan Zhou, and Huan Zhang. 2025. Improving Data Efficiency for LLM Reinforcement Fine-tuning Through Difficulty-targeted Online Data Selection and Rollout Replay. `https://doi.org/10.48550/arXiv.2506.05316` arXiv:2506.05316 [cs] version: 1.

[26] 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).

[27] 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).

[28] Fang Wu, Weihao Xuan, Ximing Lu, Zaid Harchaoui, and Yejin Choi. 2025. The Invisible Leash: Why RLVR May Not Escape Its Origin. `https://doi.org/10.48550/arXiv.2507.14843` arXiv:2507.14843 [cs].

[29] Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2025. When more is less: Understanding chain-of-thought length in llms. *arXiv preprint arXiv:2502.07266* (2025).

[30] 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).

[31] 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).

[32] Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. 2025. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335* (2025).

A Absolute Zero Reasoner

Rewards. AZR assigns complementary rewards for the proposer and solver [32]. The proposer is incentivized to create tasks of moderate difficulty using a learnability reward based on the solver's average success rate. If a task is always solved or never solved, the proposer receives no reward, while partially solvable tasks yield a higher reward:

$$r_{\text{propose}} = \begin{cases} 0 & \text{if } \bar{r}_{\text{solve}} = 0 \text{ or } \bar{r}_{\text{solve}} = 1 \\ 1 - \bar{r}_{\text{solve}} & \text{otherwise} \end{cases} \tag{4}$$

The solver receives a binary correctness reward for producing the ground-truth answer, where equality is evaluated in Python:

$$r_{\text{solve}} = \mathbb{I}_{(y=y^{\star})} \tag{5}$$

Both roles are further regularized with a format-aware penalty, ensuring adherence to the required `<think>` and `<answer>` structure [8]:

$$R = \begin{cases} r_{\text{role}} & \text{if the completion is passable} \\ -0.5 & \text{if the completion is wrong but well formatted} \\ -1 & \text{if the completion has formatting errors} \end{cases} \tag{6}$$

Learning Different Modes of Reasoning. AZR implements three reasoning modes over a coding triplet (p, i, o) consisting of a program, input, and output [32]. In deduction, the model predicts o from (p, i), with outputs verified by type-aware equality. In abduction, it infers a plausible input i from (p, o), rewarding solutions that reproduce the correct output even when programs are non-bijective. In induction, it synthesizes a program p from partial input–output pairs, with held-out examples used to ensure generalization beyond memorization. Together, these modes enable reasoning across different components of the triplet while using code execution as both an expressive interface and a verifiable environment.

Learning Algorithm. Training in AZR begins by seeding task buffers with valid program–input–output triplets generated from the base model, optionally starting from a trivial "zero" triplet. Separate buffers are maintained for deduction, abduction, and induction, and each proposer samples past triplets as in-context references to generate new tasks. Validity is enforced through a lightweight pipeline that executes candidate programs, checks syntax and safety, and restricts to deterministic outputs. To ensure stable training, if insufficient valid tasks are generated in a batch, examples are backfilled from the existing buffers.

Once validated, tasks are presented to the solver in role-specific forms, and solutions are verified by equality checks appropriate to deduction, abduction, or induction. Rewards are then assigned to both proposer and solver, and model parameters are updated with Task-Relative REINFORCE++ (TRR++), which maintains separate baselines for each task–role combination. This structure reduces variance across six training configurations and enables AZR to expand its curriculum and improve through self-play.

B Related Works

B.1 Reinforcement Learning with Verifiable Rewards

A central advance in RLVR has been the development of critic-free algorithms such as Group Relative Policy Optimization (GRPO), which reduced the computational cost of reinforcement learning for LLM post-training [8]. Building on this foundation, many works have refined GRPO and studied its limitations.

For example, Liu et al. [16] identified bias in the advantage calculation of GRPO and proposed removing the group standard deviation term. Yu et al. [30] introduced DAPO, which incorporates techniques such as dynamic sampling, filtering out overly easy or difficult samples that provide little gradient signal, and a clip higher method to encourage higher entropy. Acting as a broad empirical study, Liu et al. [17] demonstrated that simple design choices such as combining group-level mean with batch-level variance for advantage estimation, together with token-level loss aggregation, are sufficient to outperform both GRPO and more complex variants like DAPO.

Many works have also investigated RLVR from an entropy perspective as well. Cui et al. [6] observed that tokens with high covariance (high probability and advantage) drive entropy collapse. To address high covariance tokens, they either clipped those tokens from the gradient calculation or applied a KL penalty, preventing entropy collapse [6]. Wang et al. [26] also clips tokens from gradient calculation using token-level entropy as the determiner and only training on the 20% highest entropy tokens, observing that low entropy tokens contribute to entropy collapse. Cheng et al. [4] also observes that high entropy tokens are important for performance gains and exploration, and they propose an entropy aware advantage term, leading to pass@k gains on benchmarks.

Curriculum-based approaches have also shown promise, where the model is presented with problems matched to its current ability, for instance, those with a solve rate of about fifty percent across rollouts [23, 25].

Alongside these improvements, other studies have examined whether RLVR actually increases the reasoning ability of models. Yue et al. [31] showed that RLVR substantially improves pass@k when k is small, but that for large k the base model consistently surpasses its RLVR trained counterpart. Extending this finding, Wu et al. [28] provided both theoretical and empirical evidence that standard RLVR cannot escape the reasoning capacity of the base model.

B.2 LLMs for Mathematics

Given its inherent verifiability, mathematics has emerged as a compelling domain for the post-training of LLMs. This has spurred a significant body of work aimed at augmenting the mathematical capabilities of these models through RLVR [21, 27, 18]. In particular, Google DeepMind's AlphaProof achieved silver-medal standing at the International Mathematical Olympiad (IMO) by leveraging self-play with a pretrained LLM [10]. This approach utilized the formal language Lean, a strategy comparable to the use of Python in the AZR framework [19]. Such findings reinforce the idea that an LLM's coding aptitude is foundational to its downstream mathematical performance after post-training [21, 32].

B.3 Self-play

Recent work has demonstrated that self-play can be an effective strategy for LLM post-training, with consistent gains in reasoning ability. A common approach is to allow the model to generate its own training data by proposing problems and then solving them, which has been shown to yield strong improvements [32, 1, 11]. Liang et al. [13] extend this idea by evolving questions from existing datasets and report gains not only in pass@1 performance but also in pass@k when k is large. Self-play has also proven useful beyond reasoning tasks, with several studies showing improvements in instruction following and robustness [5, 9].

C AZR Inherits the Invisible-Leash Support Bound

Setting. Let $\mathcal{X}$ denote prompts or tasks and $\mathcal{Y}$ denote token sequences (solutions). A base model $q_0(y \mid x)$ initializes both AZR roles: the proposer π_t^{P} and the solver π_t^{S}. Each role is trained on verifiable rewards using on-policy policy-gradient updates (REINFORCE or PPO style), as in Absolute Zero (AZR) [32]. The generic on-policy update for either role is

$$\pi_{t+1}(y \mid x) = (1-\gamma_t)\,\frac{\pi_t(y \mid x)\,w_t(y,x)}{Z_t(x)} \;+\; \gamma_t\,\mu_t(y \mid x),$$

where $w_t(\cdot,\cdot) \geq 0$ is the clipped or exponentiated advantage weight and Z_t is the normalization factor. AZR sets $\gamma_t = 0$ (no explicit exploration) and does not use off-policy data mixing [32].

Invisible-Leash principle (reused). For any on-policy reweighting of the form above with $\gamma_t = 0$, *support is preserved*: if $\pi_t(y \mid x) = 0$ then $\pi_{t+1}(y \mid x) = 0$, since w_t is only evaluated on

$y \sim \pi_t(\cdot \mid x)$ and no mass is injected off-support. By induction from $\pi_0 = q_0$, $\operatorname{supp}(\pi_t(\cdot \mid x)) \subseteq \operatorname{supp}(q_0(\cdot \mid x))$ for all t [28].

Theorem 1 (Role-wise support preservation for AZR)**.** *Initialize with* $\pi_0^{\mathrm{P}} = \pi_0^{\mathrm{S}} = q_0$*. Under AZR's on-policy updates with* $\gamma_t \equiv 0$ *and no off-policy data,*

$$\operatorname{supp}\big(\pi_t^{\mathrm{P}}(\cdot \mid z)\big) \subseteq \operatorname{supp}(q_0(\cdot \mid z))\,, \qquad \operatorname{supp}\big(\pi_t^{\mathrm{S}}(\cdot \mid x)\big) \subseteq \operatorname{supp}(q_0(\cdot \mid x)) \quad \forall t.$$

Proof sketch. Apply the Invisible-Leash support argument [28] separately to the proposer and the solver. Both roles update via on-policy tilting (no mixing), so each preserves its support. Endogenous task selection in AZR [32] does not alter the conditional support property of $\pi_t^{\mathrm{S}}(\cdot \mid x)$; it depends only on the solver's update rule. □

Corollary 1 (Reasoning boundary / zero-probability barrier)**.** *Let* $A_x := \{y \in \mathcal{Y} :$ *verifier accepts* $(x, y)\}$*. If* $q_0(A_x) = 0$*, then for all AZR iterates* t *and* $k \geq 1$*,*

$$\Pr\big[\exists \textit{ success in } k \textit{ i.i.d. samples from } \pi_t^{\mathrm{S}}(\cdot \mid x)\big] = 0.$$

Thus AZR cannot produce verifiably correct sequences for tasks whose correct solutions have zero probability under the base model [28].

Empirical-support variant (finite-precision LLMs). Define $\operatorname{supp}_\varepsilon(q_0) := \{y : q_0(y \mid x) > \varepsilon\}$ and $S_\varepsilon := \mathcal{Y} \setminus \operatorname{supp}_\varepsilon(q_0)$. With any trust-region step (e.g., PPO clipping or per-step KL $\leq \delta$), the Invisible-Leash reweighting bound yields

$$\pi_{t+1}^{\mathrm{S}}\big(S_\varepsilon \mid x\big) \;\leq\; C(\delta) \cdot \pi_t^{\mathrm{S}}\big(S_\varepsilon \mid x\big),$$

so mass outside the base model's ε-support remains negligible absent explicit exploration ($\gamma_t > 0$) or off-policy data [28]. AZR uses $\gamma_t = 0$ and on-policy PPO, hence inherits this empirical-support limitation [32].

When can AZR escape the bound? Only by injecting off-support mass (e.g., $\gamma_t > 0$ with suitably covering μ_t), adding off-policy data, changing the base model (capacity/tools), or otherwise breaking on-policy tilting [28].

D Response Length

The following is the average response length (number of tokens in a response) for each iteration across the three tasks and two roles.

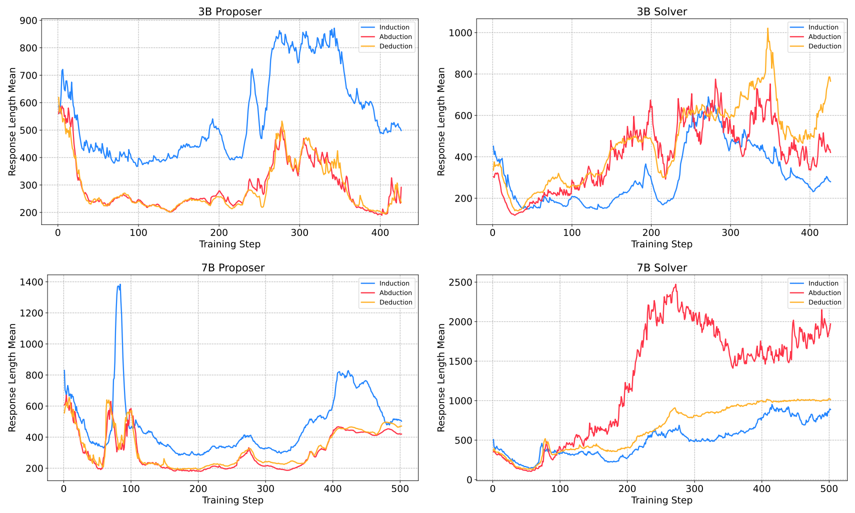

Figure 7: Response length mean per training step exponentially smoothed across model sizes and roles