BOOTSTRAPPED MIXED REWARDS FOR RL POST-TRAINING: INJECTING CANONICAL ACTION ORDER

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training with reinforcement learning (RL) typically optimizes a single scalar objective and ignores structure in how solutions are produced. We ask whether a scalar hint toward a canonical solver ordering, used only during RL post-training, improves performance even when fine-tuned on randomized solution sequences. On Zebra puzzles, we fine-tune a Transformer on randomized solution orders, then post-train it with Group Relative Policy Optimization (GRPO) using two rewards: a sparse task reward that is 1 only when the puzzle is fully solved, and an ordering reward that increases when the model's emission order aligns with the canonical solver order. To compare signals cleanly, we combine them via fixed mixtures and use a simple bootstrapped scaling to equalize component magnitudes at initialization. Mixed rewards generally outperform task-only optimization, suggesting that coarse ordering signals can steer RL post-training toward canonical trajectories without modifying supervised data or architecture.[1]

1 INTRODUCTION

Post-training with reinforcement learning (RL) often optimizes simple scalar objectives (e.g., task success) but typically ignores structure in the environment, such as the order in which intermediate actions are taken—a setup widely used when post-training language models after fine-tuning (Stiennon et al., 2020; Ouyang et al., 2022).

We study Zebra puzzles (also known as Einstein's puzzles)[2] with a GPT-2 style Transformer. We view each puzzle as a deterministic environment with latent state transitions; a policy that follows canonical solver-order trajectories can be viewed as behaving as if it maintains an internal model of valid next moves for each state. Prior work shows that Transformers trained on solver-ordered data encode the set of possible values in each cell that can be linearly decoded from hidden states, suggesting an implicit "reasoning engine" over puzzle dynamics (Shah et al., 2024). We ask whether a scalar hint toward solver ordering, used only in RL post-training, can improve performance even when fine-tuned on randomized solution orders.

The model is first fine-tuned on randomized solution orders. Next, we post-train with Group Relative Policy Optimization (GRPO) using two rewards: (i) a sparse solved reward (1 only if the model produces a fully correct solution), and (ii) an ordering reward that increases when the model's output order matches the solver order. To compare signals, we combine them via fixed mixtures and apply a bootstrapped reward scaling that equalizes component magnitudes at initialization to a target ratio.

Empirically, mixtures that include a non-zero ordering component generally outperform task-only optimization, with the best performance at a solve : order weighting of $0.99 : 0.01$. Notably, the model never sees solver-order sequences during training; the ordering information is provided only as a scalar reward during RL, yet it still biases learning toward solver-order trajectories.

Contributions. (1) A scalar reward hinting method that injects solver ordering into GRPO without modifying the fine-tuning data; (2) a bootstrapped scaling procedure that normalizes heterogeneous reward magnitudes, enabling controlled mixture studies; and (3) empirical evidence on Zebra puzzles that coarse ordering signals, when mixed with correctness, improve RL post-training accuracy.

[1] Code: `https://anonymous.4open.science/r/zebra-1202/`

[2] `https://en.wikipedia.org/wiki/Zebra_Puzzle`

2 DATA

We use the Zebra puzzle dataset introduced in Shah et al. (2024). Each example consists of a textual encoding of a Zebra puzzle followed by a sequence of actions that correspond to filling entries in an underlying entity-by-attribute grid. We filter the dataset to include only puzzles whose full solution has exactly 9 actions, and each action in the solution sequence is encoded as a triplet $(\text{row}, \text{col}, \text{val})$. We consider two variants of the solution sequence:

(i) a solver-order sequence, which records the exact order in which a canonical Zebra solver fills the solution cells, and

(ii) a random-order sequence, produced by shuffling the solver sequence uniformly at random.

Following Shah et al. (2024), the Zebra solver is a deterministic, human-like procedure that attempts to solve puzzles iteratively without backtracking. At each step, given the clues and partial grid, it tries to fill an entry whose value is uniquely determined under its fixed rule set, and repeats until solved. The solver-order sequence is the chronological list of triplets produced by this procedure.

The dataset design plays a crucial role in this work. Because the solver sequence reflects a valid step-by-step reasoning trajectory and the random sequence does not, we can measure and reward the degree to which the model follows solver ordering during generation. This choice is motivated by evidence that models trained on solver steps learn substantially more effectively than models trained on randomized steps (Shah et al., 2024).

3 TRAINING SETUP

We use a GPT-2 style Transformer as our base architecture. The model has 4 layers, 4 attention heads per layer, and a hidden size of 256. The model is trained from scratch rather than initialized from a pretrained checkpoint. Hyperparameters for our training can be found in Appendix A.

Standard Fine-Tuning. The model is first trained with a standard causal language modeling objective on the Zebra dataset. For each example, the puzzle is given as input, and the target output consists of the corresponding solution sequence. The loss is computed only over the solution tokens, with puzzle tokens masked out. We apply this procedure to fine-tune our baseline model on randomized solution orders.

GRPO Post-Training. After fine-tuning, we further post-train the random-order model using Group Relative Policy Optimization (GRPO) (Shao et al., 2024), a reinforcement learning algorithm that enables optimizing arbitrary reward functions over model rollouts. During GRPO training, rewards are computed over generated sequences and used to update the model via the GRPO loss. We evaluate several reward designs, including a solved reward, order reward, and their weighted combinations (described in Section 4).

4 REWARD DESIGN

A key focus of this work is the design of reward functions that guide GRPO post-training. Our objective is to study whether scalar hints about the underlying solving order can improve solving accuracy without explicitly training on ordered sequences. We consider two primary reward functions and several fixed weighted combinations of the two. In all reward computations, we score only the completion prefix up to the ground-truth action length to keep indices comparable across rollouts.

4.1 SOLVED REWARD

Let the ground-truth solution be the set of triplets $S^\star = \{(r, c, v)\}$. We define a sparse solved reward that is 1 only when the model produces a fully correct solution and 0 otherwise, capturing overall correctness but not the order of predicted cells. We parse the model completion into triplets, ignore any cell (r, c) assigned multiple distinct values, and mark the rollout as solved only if every ground-truth cell is present with the correct value:

$$R_{\text{solve}} = \mathbb{1}[\forall (r, c, v) \in S^\star : \hat{v}(r, c) = v \ \wedge \ \text{no conflicting values are assigned to } (r, c)]\,. \quad (1)$$

4.2 ORDER REWARD

The order reward measures how closely the model's generation order matches the canonical solver order, independent of whether the predicted values are correct. Let $\pi^\star(r, c)$ denote the index of cell (r, c) in the canonical solver solution, and let $\hat{\pi}(r, c)$ denote the index at which the model first emits cell (r, c) in its completion. We only include cells that the model emits exactly once (to avoid ambiguity from repeated assignments). For eligible cells, we compute

$$r(r, c) \;=\; \frac{1}{1 + |\pi^\star(r, c) - \hat{\pi}(r, c)|}. \tag{2}$$

We then average over eligible cells:

$$R_{\text{order}} \;=\; \frac{1}{|\mathcal{C}|} \sum_{(r,c)\in\mathcal{C}} r(r, c), \tag{3}$$

where $\mathcal{C}$ is the set of ground-truth cells emitted exactly once. If $\mathcal{C}$ is empty, we set $R_{\text{order}} = 0$.

This yields a reward-shaping signal (Ng et al., 1999) that guides toward solver-like rollouts without enforcing exact sequence reproduction.

4.3 COMBINED REWARDS AND BOOTSTRAPPED SCALING

We combine the two rewards via a fixed weighted sum:

$$R_{\text{total}} \;=\; \alpha \cdot R_{\text{solve}} \;+\; (1 - \alpha) \cdot R_{\text{order}}, \qquad \alpha \in [0, 1]. \tag{4}$$

Viewing R_{total} as a weighted composition of objectives follows standard practice in multi-objective RL (Roijers et al., 2013). To avoid manual tuning when the raw magnitudes of R_{solve} and R_{order} differ, we employ a simple bootstrapped reward scaling prior to GRPO. We evaluate the frozen fine-tuned model on the validation split and compute mean rewards $\bar{R}_{\text{solve}}$ and $\bar{R}_{\text{order}}$. For a desired mixture α, we set global scalars

$$\textsc{SolveScale} \;=\; \frac{\alpha}{\bar{R}_{\text{solve}}} \qquad \text{and} \qquad \textsc{OrderScale} \;=\; \frac{1 - \alpha}{\bar{R}_{\text{order}}}, \tag{5}$$

and use $R_{\text{total}} = \textsc{SolveScale} \cdot R_{\text{solve}} + \textsc{OrderScale} \cdot R_{\text{order}}$ throughout training.

This ensures that, at initialization, each component contributes to R_{total} in the target ratio regardless of absolute scale. It simplifies analysis across mixtures by keeping the intended weighting explicit and stable over runs.

In practice, we run a short evaluation pass of the fine-tuned model on the validation split before GRPO, compute the empirical means of each reward term, and set fixed scaling factors once. We then keep these factors constant for the entire post-training phase. Holding the scales fixed aligns initial magnitudes with the chosen mixture and avoids accidental dominance from raw reward-scale differences, making mixture comparisons interpretable. Under this normalization, improvements in either component contribute according to the intended mixture.

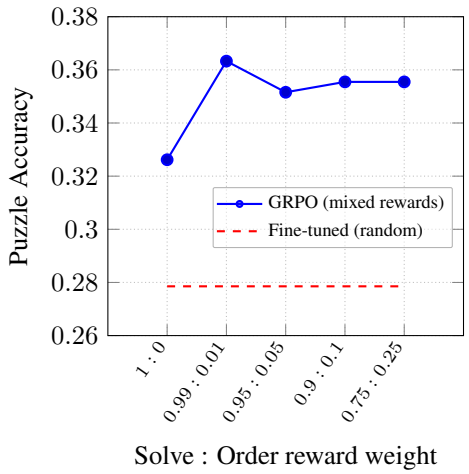
5 RESULTS AND ANALYSIS

We evaluate our models on the held-out test set using puzzle accuracy as the primary metric—the fraction of puzzles fully solved by the model. All results use greedy decoding at inference time. We report metrics for

1. the model fine-tuned on random-order sequences, and
2. models post-trained with GRPO starting from the random-order fine-tuned checkpoint under fixed reward mixtures $\alpha \in \{0.75, 0.9, 0.95, 0.99, 1\}$.

5.1 GRPO WITH DIFFERENT REWARD COMBINATIONS

We first fine-tune a model on random solution order, which achieves a 0.279 puzzle accuracy on the test set. We then post-train this model with GRPO using different reward mixtures. Figure 1 summarizes test puzzle accuracies. Mixtures that include a non-zero ordering component consistently



Solve : Order Weight	Puzzle Accuracy
1 : 0	0.326
0.99 : 0.01	**0.363**
0.95 : 0.05	0.352
0.9 : 0.1	0.355
0.75 : 0.25	0.355
Fine-tuned (random order)	0.279

Figure 1: **Reward mixtures and performance.** Effect of reward mixing on Zebra puzzle accuracy. Each point is GRPO post-trained on the fine-tuned (random order) model at the indicated α. (Note: x-axis positions are categorical and not equidistant in α.)

outperform task-only optimization ($1:0$), with the best result at a $0.99:0.01$ solve-to-order weighting (0.363). Notably, even a very small ordering share yields a clear gain over task-only (0.326), suggesting that the ordering signal is effective as a light shaping term. A broader range of mixtures also improves over the fine-tuned baseline (0.279): $0.75:0.25$ and $0.9:0.1$ both reach 0.355, while $0.95:0.05$ reaches 0.352.

Overall, these results on Zebra puzzles suggest that a coarse scalar ordering hint complements the solved objective: mixed rewards steer the model toward solver-like trajectories at inference time without requiring solver-order sequences during fine-tuning.

5.2 Reward Scaling Effects

The bootstrapped reward scaling makes reward mixing well-defined by aligning component magnitudes to the chosen mixture at initialization. Without normalization, differences in raw scale can cause R_{total} to be dominated by one term regardless of α, making mixture comparisons misleading. By setting fixed scales from validation-set means, we can mix solved and ordering rewards in a controlled ratio and attribute performance changes to the intended weighting.

6 Conclusion

We asked whether a scalar hint about temporal structure (alignment with a canonical solver order) can improve RL post-training without changing supervised data or model architecture. On Zebra puzzles, GRPO with bootstrapped mixtures of solved and ordering rewards yields consistent gains over task-only optimization; even a very small ordering weight improves puzzle accuracy, with the best mixture at $0.99:0.01$.

The ordering reward nudges the policy toward a solver-like progression, producing more canonical rollouts even when the fine-tuning data is randomly ordered. Practically, this provides a cheap, modular post-training knob for injecting structural bias without curating new supervised datasets or training from scratch.

Limitations. This work is ongoing. Our experiments are limited to a single task (Zebra puzzles) and a single model (a GPT-2 style Transformer). We also use fixed bootstrapped scaling factors; because reward components can improve at different rates during training, these fixed scales may become miscalibrated as the policy shifts. A natural next step is to test whether periodically updating the scaling factors improves stability and performance, and whether these findings generalize across additional tasks, scales, and architectures.

REFERENCES

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pp. 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606122.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Int. Res.*, 48(1):67–113, October 2013. ISSN 1076-9757.

Kulin Shah, Nishanth Dikkala, Xin Wang, and Rina Panigrahy. Causal language modeling can elicit search and reasoning capabilities on logic puzzles. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 56674–56702. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/67b31ca159553d5593e62d7b998d63ea-Paper-Conference.pdf.

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.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3008–3021. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1f89885d556929e98d3ef9b86448f951-Paper.pdf.

Parameter	Value
Learning Rate	1e-4
Weight Decay	0.01
Per-device train batch size	32
Per-device eval batch size	32
Validation Size	512
Validation Patience	10
Optimizer	AdamW

Table 1: **Hyperparameters for fine-tuning.**

Parameter	Value
Learning Rate	1e-6
Per-device train batch size	128
Per-device eval batch size	128
Number of rollouts	8
Weight Decay	0.01
KL penalty Beta	0.01
Optimizer	AdamW

Table 2: **Hyperparameters for GRPO.**

A HYPERPARAMETERS

We list the hyperparameters used to train each setup in this section.

A.1 STANDARD FINE-TUNING

We train a 4-layer model with 4 attention heads per layer, and a hidden size of 256. Table 1 lists the training hyperparameters.

A.2 GRPO

Table 2 lists the hyperparameters used for GRPO post-training.

B LLM USAGE

We used an LLM (ChatGPT) to assist with writing edits and LaTeX formatting. All technical content, experiments, and claims were produced and verified by the authors.