CORRECT IS NOT ENOUGH: TRAINING REASONING PLANNERS WITH EXECUTOR-GROUNDED REWARDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards has become a common way to improve explicit reasoning in large language models, but final-answer correctness alone does not reveal whether the reasoning trace is faithful, reliable, or useful to the model that consumes it. This outcome-only signal can reinforce traces that are right for the wrong reasons, overstate reasoning gains by rewarding shortcuts, and propagate flawed intermediate states in multi-step systems. To this end, we propose TraceLift, a planner-executor training framework that treats reasoning as a consumable intermediate artifact. During planner training, the planner emits tagged reasoning. A frozen executor turns this reasoning into the final artifact for verifier feedback, while an executor-grounded reward shapes the intermediate trace. This reward multiplies a rubric-based Reasoning Reward Model (RM) score by measured uplift on the same frozen executor, crediting traces that are both high-quality and useful. To make reasoning quality directly learnable, we introduce TRACELIFT-GROUPS, a rubric-annotated reason-only dataset built from math and code seed problems. Each example is a same-problem group containing a high-quality reference trace and multiple plausible flawed traces with localized perturbations that reduce reasoning quality or solution support while preserving task relevance. Extensive experiments on code and math benchmarks show that this executor-grounded reasoning reward improves the two-stage planner-executor system over execution-only training, suggesting that reasoning supervision should evaluate not only whether a trace looks good, but also whether it helps the model that consumes it.

1 INTRODUCTION

Large language models increasingly use explicit reasoning trajectories especially during post-training with verifiable feedback. Chain-of-thought, bootstrapped rationales, program-aided reasoning, and recent reinforcement learning methods encourage models to produce intermediate text before final answers or programs (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2023; Yao et al., 2023a; Zelikman et al., 2022; Gao et al., 2022; Chen et al., 2023; Shao et al., 2024; DeepSeek-AI, 2025; Han et al., 2025). These trajectories are not merely explain the plan, constraints, intermediate derivations, and implementation choices but help to shape the final artifact. Reinforcement learning with verifiable feedback provides a natural way to improve reasoning-based systems. (Shao et al., 2024; DeepSeek-AI, 2025; Le et al., 2022; Liu et al., 2023a; Shojaee et al., 2023; Su et al., 2025; Pi et al., 2024a)

As reasoning trajectories become part of the computation, reward signals must evaluate not only final correctness but also the quality and downstream effect of the reasoning process itself. Existing reward signals indicate whether a final answer or program succeeds, but they can hide flawed intermediate reasoning behind correct artifacts. This creates a credit-assignment failure in verifiable reinforcement learning: traces that are right for the wrong reasons, rely on shortcuts, or mask early mistakes with later fluent text can still be reinforced as successful behavior. Process supervision and automated process verifiers provide more direct reasoning feedback, yet language-model-based judges can inherit reasoning errors, calibration failures, or hallucinated assessments (Uesato et al., 2022; Gao et al., 2022; Wang et al., 2024; Setlur et al., 2025; Han et al., 2024; Pi et al., 2024b). These limitations leave underexplored how to train reasoning trajectories that are both internally reliable and useful to the model that consumes them.

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

Owing to the modular role that reasoning trajectories already play in reasoning-based systems, they can be viewed as an interface between a trainable planner and a fixed executor. Under this view, the planner should not be rewarded merely for producing coherent-looking text. It should be rewarded for producing an intermediate trajectory that is reliable as a reasoning process and useful to the executor that consumes it. This perspective changes both the training and evaluation targets. Training should assign credit to trajectories according to their intrinsic quality and their measured effect on the executor. Evaluation should hold the executor fixed so that improvements reflect better executor-consumable reasoning rather than a stronger final-answer generator.

To address these limitations, we propose TraceLift as a planner-executor framework that treats reasoning as a consumable intermediate artifact in a controlled planner-executor setting. We introduce a trainable reasoning planner policy that produces an intermediate trajectory consumed by a fixed executor rather than a final artifact for evaluation. The frozen executor uses the problem and trajectory to generate the final artifact, which is then scored by a verifier using task-specific correctness checks. This separation grounds reasoning supervision in the executor that will actually use the trajectory. During planner training, a rubric-based Reasoning Reward Model (RM) scores the trajectory itself and this score is weighted by measured executor uplift over a no-reasoning baseline. The resulting reward gives credit to reasoning only when it is both high quality and useful to the fixed executor. At evaluation time, the Reason RM and measured-uplift computation are removed. Improvements therefore reflect whether the trained planner better guides the same frozen executor under the same two-stage evaluation chain.

Specifically, to overcome the data bottleneck in learning reason-only quality signals, we introduce TRACELIFT-GROUPS as a rubric-annotated reason-only dataset. TRACELIFT-GROUPS contains 6,000 reasoning groups sampled from GSM8K train and OpenCodeReasoning. Each group anchors a high-quality reference trajectory and multiple plausible flawed trajectories to the same problem. These flawed trajectories are produced by targeted perturbations that weaken reasoning quality or solution support while preserving task relevance. An agentic rubric annotation pipeline then assigns multi-dimensional reasoning-quality scores. This grouped reason-only format trains the Reason RM to score a problem-reasoning pair by trajectory quality rather than final-solution correctness.

Our contributions are as follows.

- We introduce TraceLift, an executor-grounded training framework for reasoning planners. It rewards trajectories using both rubric-based reasoning quality and measured uplift on the same frozen executor, making the training signal aligned with the model that will consume the trajectory at test time.
- We introduce TRACELIFT-GROUPS, a rubric-annotated reason-only dataset with 6,000 reasoning groups sampled from GSM8K train and OpenCodeReasoning. Each group pairs a reference trajectory with targeted flawed trajectories and multi-dimensional rubric annotations for reasoning-trajectory quality rather than final-artifact correctness.
- Extensive experiments on code and math benchmarks demonstrate that TraceLift consistently improves fixed two-stage planner-executor systems over execution-only reinforcement learning under the same evaluation chain, highlighting the advantage of rewarding reasoning by both trajectory quality and downstream executor utility.

2 METHOD

TraceLift trains reasoning traces as executor-consumable intermediate artifacts in a fixed planner-executor protocol. In this section, we first describe TRACELIFT-GROUPS, the grouped reason-only supervision data used to learn trace-quality judgments. We then introduce TraceLift training framework, which uses scores problem-reasoning pairs to train the offline Reasoning Reward Model (RM). Finally, we describe executor-grounded planner optimization, where verifier feedback is combined with an uplift-weighted Reasoning RM score to train the planner with GRPO.

2.1 TRACELIFT-GROUPS

TRACELIFT-GROUPS provides reason-only supervision for evaluating reasoning trajectories without using final-artifact correctness as the label. Existing outcome or process labels usually do not

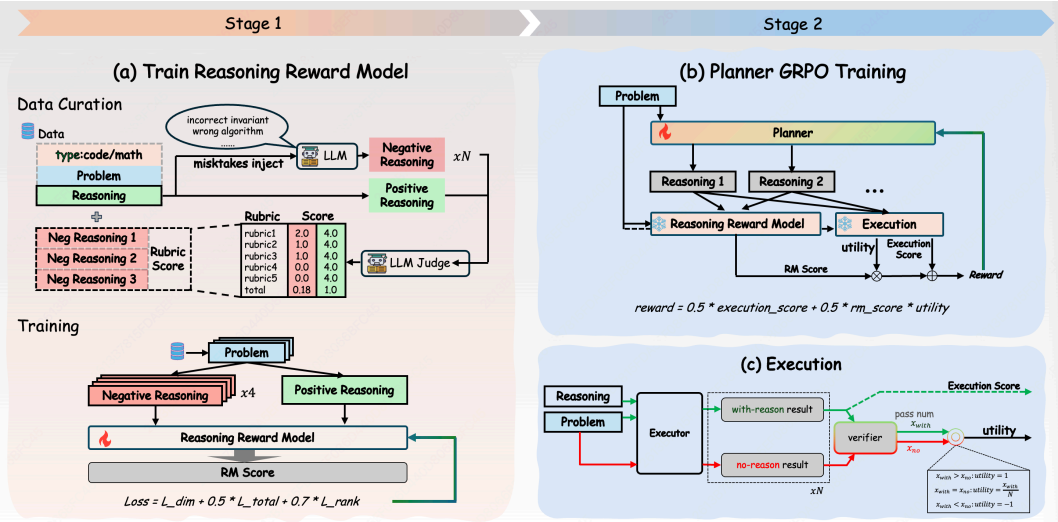


Figure 1: The overall framework of TRACELIFT-GROUPS and TraceLift. **(a)** Data curation pipeline of TRACELIFT-GROUPS. Then we use TRACELIFT-GROUPS to finetune the reward model specialized for reasoning supervising by the designed loss. **(b)** GRPO training process of the planner using previous trained reasoning reward model. **(c)** Details of execution calculation process. The Reasoning RM score is weighted by measured executor uplift before being combined with verifier feedback for planner optimization.

isolate whether an intermediate trajectory is reliable and useful to the executor that consumes it. We therefore introduce TRACELIFT-GROUPS, a rubric-annotated reason-only dataset. Each example is a reasoning group anchored to one problem, containing a high-quality reference trajectory and multiple plausible flawed trajectories. The dataset is constructed in two stages. We first synthesize targeted trajectory perturbations, and then use an agentic rubric annotation pipeline to assign multi-dimensional reasoning-quality scores.

2.1.1 TRAJECTORY PERTURBATION

We construct TRACELIFT-GROUPS from 3,000 seed problems sampled from OpenCodeReasoning and 3,000 seed problems sampled from the GSM8K training set. Each seed provides a problem, a reference reasoning trajectory, and a reference result. The reference trajectory is normalized into a reason-only trajectory that preserves the intermediate reasoning.

As shown in Figure 1, we generate multiple plausible flawed trajectories through targeted error injection from each reference trajectory. The perturbation is local enough to keep the trajectory task-relevant, but strong enough to reduce reasoning reliability or solution support. Code perturbations include wrong algorithm choices, missing edge cases, off-by-one reasoning, incorrect invariants, infeasible complexity, vague pseudo-solutions, and irrelevant explanations. Math perturbations include arithmetic slips, wrong operations, dropped cases, unit mismatches, unsupported jumps, premature answers, and content-free reasoning. Across the 6,000 reasoning groups, these flawed trajectories cover seven perturbation types per domain. This design turns each problem into a controlled comparison over reasoning quality rather than a binary outcome-label example.

2.1.2 AGENTIC RUBRIC ANNOTATION

We annotate each trajectory with an Large-Language-Model-based rubric judge that evaluates the reasoning process rather than the final artifact. The judge receives the problem, the candidate's trajectory, and task-specific rubric instructions. It returns dimension scores, a total score, and a short rationale for the rating. For code, the rubric scores task understanding, plan quality, step coherence, action support, and non-leakage. For math, it scores problem understanding, solution strategy, step coherence, calculation correctness, and answer support. Each dimension is mapped to a five-class label, and the total score is normalized to $[0, 1]$.

The annotation pipeline is designed to prevent two shortcuts in Reasoning RM training. A trajectory should not receive high reward merely because a later artifact happens to pass, and it should not receive high reward merely because it is fluent or verbose. By placing reference and flawed trajectories under the same problem context, TRACELIFT-GROUPS provides both pointwise rubric supervision and within-group ranking supervision for learning trajectory quality.

2.2 TRACELIFT TRAINING FRAMEWORK

As shown in Figure 1, TraceLift has two training stages. The first stage trains a Reasoning RM to score problem-reasoning pairs from rubric-labeled reasoning groups. The second stage uses this score together with verifier feedback and measured executor uplift to train the planner with GRPO. The Reasoning RM and measured uplift are training-time reward components, while the planner remains the only trainable generator in the planner-executor chain.

2.2.1 OFFLINE REASONING RM TRAINING

We first train a Reasoning RM offline on TRACELIFT-GROUPS before planner optimization. Given a problem P and a reasoning trajectory R, the Reasoning RM returns a scalar quality score.

$$\mathrm{RM}_\phi(P, R) \to s_{\mathrm{rm}}, \qquad s_{\mathrm{rm}} \in [0, 1]. \tag{1}$$

The RM input contains the problem context and the trajectory. Here P denotes the static task context available before artifact generation. It may include the problem statement and any public constraints or tests supplied in the prompt. It excludes executor outputs and verifier outcomes as well as final answers or code. This restriction makes the offline training target rubric-labeled reasoning quality rather than outcome correctness.

The Reasoning RM is applied pointwise but trained with group supervision. For each update, we sample one reference trajectory and four flawed trajectories from the same reasoning group. Each trajectory is encoded independently as a problem-reasoning input, while the group structure is used only by the ranking loss. The model uses five rubric heads to predict discrete dimension labels and an aggregate head to predict the normalized total score. For dimension k, the annotated rubric score $\tilde{y}_k^\star$ is rounded and clipped into a five-class label:

$$y_k^\star = \mathrm{clip}\left(\left\lfloor \tilde{y}_k^\star + \frac{1}{2} \right\rfloor, 0, 4\right), \qquad \mathcal{L}_{\mathrm{dim}} = -\frac{1}{5}\sum_{k=1}^{5} \log p_{k, y_k^\star}. \tag{2}$$

Let $p_{k,c}$ be the predicted probability that rubric dimension k receives label $c \in \{0, \ldots, 4\}$. We compute the rubric-head-derived score as

$$s_{\mathrm{rm}}^{\mathrm{dim}} = \sum_{k=1}^{5} w_k \sum_{c=0}^{4} p_{k,c} \frac{c}{4}, \qquad \sum_{k=1}^{5} w_k = 1. \tag{3}$$

The aggregate head outputs $s_{\mathrm{rm}}^{\mathrm{total}} = \sigma(z_{\mathrm{total}})$, and is trained with a Huber loss against the clipped normalized total score:

$$\mathcal{L}_{\mathrm{total}} = \mathrm{Huber}_{\delta=1}\left(s_{\mathrm{rm}}^{\mathrm{total}} - \mathrm{clip}(\tilde{y}_{\mathrm{total}}^\star, 0, 1)\right). \tag{4}$$

The online RM score combines $s_{\mathrm{rm}}^{\mathrm{dim}}$ with the aggregate score:

$$s_{\mathrm{rm}} = 0.5\, s_{\mathrm{rm}}^{\mathrm{dim}} + 0.5\, s_{\mathrm{rm}}^{\mathrm{total}}. \tag{5}$$

We optimize

$$\mathcal{L}_{\mathrm{RM}} = \mathcal{L}_{\mathrm{dim}} + \alpha \mathcal{L}_{\mathrm{total}} + \beta \mathcal{L}_{\mathrm{rank}}, \qquad \mathcal{L}_{\mathrm{rank}} = -\log \sigma(s_{\mathrm{rm}}^{+} - s_{\mathrm{rm}}^{-}). \tag{6}$$

The ranking pairs are drawn from the same reasoning group. In the current implementation, we set $\alpha = 0.5$ and $\beta = 0.7$.

2.2.2 EXECUTOR-GROUNDED PLANNER OPTIMIZATION

We then train the planner to produce trajectories that are both rubric-good and useful to the frozen executor. For each problem, the planner samples reasoning trajectories using the GRPO objective from the preliminary. For each sampled trajectory R_i, the frozen executor produces a final artifact A_i, and the verifier scores that artifact.

$$A_i = E(P, R_i),$$
$$R_{\text{exec}}(P, R_i) = V(A_i), \tag{7}$$

where $R_i \sim \pi_\theta(\cdot \mid P)$, and $R_{\text{exec}}(P, R_i)$ is the executor reward assigned to the sampled trajectory, and V denotes the task verifier. To ground the RM score in executor utility, TraceLift also compares the same frozen executor with and without the planner trajectory. For uplift measurement, $V \in \{0, 1\}$ denotes binary verifier success under matched decoding settings. We estimate the success rate with trajectory R as $\hat{p}(P, R)$ and the no-reasoning baseline success rate as $\hat{p}_0(P)$.

$$\hat{p}(P, R) = \frac{1}{K} \sum_{k=1}^{K} V(E_k(P, R)),$$
$$\hat{p}_0(P) = \frac{1}{K} \sum_{k=1}^{K} V(E_k(P, \varnothing)),$$
$$u_{\text{exec}}(P, R) = \text{clip}\left(\hat{p}(P, R) - \hat{p}_0(P),\ -1,\ 1\right). \tag{8}$$

where the trajectory R is fixed over repeated executor samples during estimation. No-reasoning baseline uses the same executor prompt with the reasoning field omitted. Positive uplift means that R improves verifier success for this executor, while negative uplift means that it hurts the executor.

The scalar reward therefore combines verifier feedback, Reasoning RM quality and executor uplift:

$$\mathcal{R}(P, R) = 0.5\, R_{\text{exec}}(P, R) + 0.5\, \text{RM}_\phi(P, R)\, u_{\text{exec}}(P, R). \tag{9}$$

The verifier term anchors planner training to task success. The uplift-weighted RM term increases the reward when a high-rubric trajectory also improves the frozen executor. This combined scalar reward is assigned at the completion level and used in the GRPO objective. At test time, the Reasoning RM and measured uplift are removed, and Section 3 describes the resulting evaluation protocol.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Models and training objectives. We train TraceLift on Qwen2.5-7B, Llama3.1-8B, and Qwen3-4B model families (Yang et al., 2024; Grattafiori et al., 2024; Yang et al., 2025). For the main policy Group Relative Policy Optimization (GRPO) experiments, all Qwen2.5-7B, Llama3.1-8B, and Qwen3-4B policies are trained for 600 steps in bf16 with learning rate 5×10^{-6}, and we use a temperature of 0.5.

Evaluation metrics. For code, we evaluate TraceLift on HumanEval (Chen et al., 2021), HumanEval+ (Liu et al., 2023b), MBPP-full (Austin et al., 2021) and LiveCodeBench (Jain et al., 2025). For math, we evaluate GSM8K (Cobbe et al., 2021), GSM-Hard (Gao et al., 2022), SVAMP (Patel et al., 2021) and MATH500 (Lightman et al., 2023).

Evaluation protocols. For each problem, the trained policy first generates a reasoning trace with greedy decoding. The frozen executor then consumes the problem and reasoning and generates the final artifact. Code artifacts are judged by executable tests, and math artifacts are judged by answer matching. The Reason RM and uplift estimator are not called during evaluation. For math only, we also report a supplementary direct self-solving protocol in which the policy itself generates both reasoning and final answer; this protocol is not the main evidence for the planner-executor claim.

3.2 MAIN RESULTS

Training details. Compared with TraceLift, Exec-only only trains the policy with end-to-end verifier reward instead of being trained with Eq. 9.

Table 1: Two-stage results on code and math benchmarks. We use pass@1 percentages for code and accuracy for math entries to assess. During evaluation, the executor is initialized from the same model family as the planner. HE and HE+ denote HumanEval and HumanEval+, LCB denotes LiveCodeBench. All results reported in the table are averaged over three runs with different seeds. Micro avg. denotes accuracy pooled over the listed benchmark instances.

Model	Method	Code					Math				
		HE	HE+	MBPP	LCB	Micro avg.	GSM8K	GSM-Hard	SVAMP	MATH500	Micro avg.
Qwen2.5-7B	Base	68.90	60.37	58.00	29.50	50.49	61.79	30.02	58.67	42.40	46.51
	Exec-only	70.12	61.59	59.00	32.75	52.28	87.04	47.84	68.67	48.00	64.72
	TraceLift	**72.56**	**64.02**	**61.20**	**36.00**	**54.89**	**89.16**	**51.78**	**89.67**	**50.40**	**69.23**
Llama3.1-8B	Base	33.54	29.88	34.40	10.25	25.81	16.60	4.93	34.33	10.20	12.74
	Exec-only	37.80	31.10	45.20	15.00	32.49	32.15	9.45	48.67	13.40	22.13
	TraceLift	**39.02**	**32.93**	**46.40**	**18.25**	**34.45**	**35.71**	**9.70**	**49.00**	**14.60**	**23.82**
Qwen3-4B	Base	79.27	75.61	63.00	56.50	64.74	88.48	52.16	86.00	67.00	71.20
	Exec-only	81.09	76.83	63.60	58.00	65.88	89.01	52.01	88.67	66.40	71.50
	TraceLift	**84.15**	**78.66**	**65.80**	**60.75**	**68.32**	**89.23**	**53.22**	**89.00**	**68.20**	**72.34**

Results on Code Benchmarks. Table 1 reports code results under the fixed two-stage protocol. Across all three model families, TraceLift improves over Exec-only on every benchmark. On the micro-average, TraceLift improves over Exec-only by 2.61 percentage points for Qwen2.5-7B, 1.96 points for Llama3.1-8B, and 2.44 points for Qwen3-4B. These gains are especially meaningful because the executor is frozen during evaluation: the only component changed is the planner that produces the reasoning trace.

The code results show two complementary effects. First, execution-only RL already improves over the untrained two-stage baseline, indicating that verifier feedback is a useful signal for planner training. Second, TraceLift consistently improves over Exec-only, supporting the combined training recipe under the reported protocol. The improvement is largest on the harder LiveCodeBench public-test setting for Qwen2.5-7B and Llama3.1-8B, where small changes in the plan often determine whether the executor handles constraints, edge cases, and implementation details correctly. This supports the central claim that reasoning traces should be rewarded not only for downstream correctness, but also for whether they provide executor-consumable guidance.

Results on Math Benchmarks. Table 1 reports math results under the same two-stage planner-executor protocol. TraceLift again improves over Exec-only across all model families and all listed benchmarks. On the micro-average, TraceLift improves over Exec-only by 4.51 percentage points for Qwen2.5-7B, 1.69 points for Llama3.1-8B, and 0.84 points for Qwen3-4B.

The math results are consistent with the code results but reveal a stronger dependence on model strength. For Qwen2.5-7B, TraceLift gives a large micro-average gain over Exec-only, with particularly strong improvement on SVAMP and GSM-Hard. For Qwen3-4B, the base two-stage system is already strong, so the remaining headroom is smaller; nevertheless, TraceLift still improves over Exec-only on every listed benchmark. This pattern suggests that executor-grounded reasoning rewards are most valuable when the planner must supply missing structure to a capable but imperfect executor, while still providing gains for stronger base systems.

3.3 LoRA and Full-Parameter GRPO

Table 4 compares LoRA and full-parameter GRPO for Qwen2.5-7B. On code, full-parameter training improves both objectives, but TraceLift remains stronger than Exec-only under the same parameterization. In the full-parameter setting, TraceLift reaches a code micro-average of 60.26%, compared with 57.90% for Exec-only. On math, full-parameter training is not uniformly better than LoRA, but the objective-level comparison remains favorable: TraceLift improves over Exec-only under both LoRA and full-parameter training.

These results indicate that the gains from TraceLift are not merely a byproduct of parameter budget. On code, increasing trainable capacity raises the ceiling, and the executor-grounded reward continues to provide an additional gain over execution-only training. On math, LoRA TraceLift is the strongest overall configuration, while full-parameter TraceLift still outperforms full-parameter

Table 2: Supplementary math direct self-solving results. All entries are answer accuracy percentages. In this protocol, the policy generates both reasoning and final answer while no frozen executor is used at evaluation time. All results reported in the table are averaged over three runs with different seeds.

Model	Method	GSM8K	GSM-Hard	SVAMP	MATH500	Micro avg.
Qwen2.5-7B	Base	61.79	30.02	58.67	42.40	46.51
	Exec-only	87.26	47.92	67.67	54.00	65.62
	TraceLift	**89.31**	**52.16**	**90.33**	**55.20**	**70.19**
Llama3.1-8B	Base	16.60	4.93	34.33	10.20	12.74
	Exec-only	31.92	9.78	46.33	14.80	22.19
	TraceLift	**32.90**	**10.00**	**47.67**	**15.40**	**22.86**
Qwen3-4B	Base	88.48	52.16	86.00	67.00	71.20
	Exec-only	88.93	52.39	**88.67**	68.60	71.93
	TraceLift	**89.16**	**52.69**	**88.67**	**69.40**	**72.25**

Table 3: Ablation on the number of executor comparisons K used to estimate uplift during Qwen2.5-7B code training. All entries are pass@1 percentages.

Benchmark	$K = 1$	$K = 3$	$K = 5$
HumanEval	68.29	**72.56**	67.07
HumanEval+	63.41	**64.02**	60.37
MBPP-full	60.80	**61.20**	60.40
LiveCodeBench	35.00	**36.00**	35.50
Micro avg.	53.75	**54.89**	53.18

Exec-only. This suggests that the training objective and the adaptation regime play distinct roles: more trainable parameters can help, but they do not replace the need for a reward that distinguishes useful reasoning from reasoning that merely correlates with final correctness.

3.4 Supplementary Math Direct Self-Solving

The main evaluation fixes the executor and measures whether the trained planner produces better executor-consumable reasoning. A possible concern is that such training might overspecialize the policy to the frozen executor and harm its own direct problem-solving ability. To test this, we evaluate math policies in a direct self-solving protocol where the policy generates both the reasoning and final answer. Table 2 reports the results.

The direct self-solving results rule out a simple negative-transfer explanation. TraceLift improves or matches Exec-only on all listed math direct-solving settings, including a 4.57 point micro-average gain for Qwen2.5-7B. Thus, the executor-grounded reward does not appear to train traces that are only useful to an external executor while degrading the policy's own ability to complete the solution. Instead, the learned reasoning remains useful when the same policy must also produce the final answer. We nevertheless treat this as supplementary evidence, since the main claim of this paper concerns the fixed planner-executor setting.

3.5 Ablation Studies

We run ablations on Qwen2.5-7B code tasks, where executable tests provide a high-precision verifier and the planner-executor separation is most direct.

Number of executor comparisons. The uplift estimate in Eq. 8 depends on the number of executor comparisons used during training. Table 3 varies this number. The main setting uses $K = 3$.

The sweep shows that the uplift estimator is useful but not monotonically improved by more comparisons. With $K = 1$, the reward can be too sensitive to a single executor outcome, so the planner may

Table 4: Qwen2.5-7B LoRA and full-parameter GRPO results under the two-stage protocol. All results reported in the table are averaged over three runs with different seeds.

Domain	Benchmark	Base	Exec-only LoRA	Exec-only Full	TraceLift LoRA	TraceLift Full
Code	HumanEval	68.90	70.12	73.78	72.56	**76.83**
	HumanEval+	60.37	61.59	68.29	64.02	**70.12**
	MBPP-full	58.00	59.00	61.00	61.20	**63.60**
	LiveCodeBench	29.50	32.75	43.25	36.00	**45.25**
	Micro avg.	50.49	52.28	57.90	54.89	**60.26**
Math	GSM8K	61.79	87.04	86.66	**89.16**	87.79
	GSM-Hard	30.02	47.84	48.29	**51.78**	49.28
	SVAMP	58.67	68.67	83.67	**89.67**	89.33
	MATH500	42.40	48.00	48.80	50.40	**52.00**
	Micro avg.	46.51	64.72	66.17	**69.23**	67.95

Table 5: Ablation results of Qwen2.5-7B code reward. All entries are pass@1 percentages. Full TraceLift combines verifier reward with uplift-weighted reasoning reward model.

Benchmark	Base	Exec-only	TraceLift	No-uplift	RM-uplift only
HumanEval	68.90	70.12	**72.56**	65.85	71.34
HumanEval+	60.37	61.59	**64.02**	60.37	63.41
MBPP-full	58.00	59.00	**61.20**	58.40	58.60
LiveCodeBench	29.50	32.75	**36.00**	33.50	33.25
Micro avg.	50.49	52.28	**54.89**	51.55	52.69

receive high credit for traces that happen to help one executor comparison but do not robustly guide the executor. Increasing to $K = 3$ stabilizes this marginal-utility signal and gives the best aggregate performance. However, $K = 5$ is weaker in this setting. A likely reason is that a more averaged uplift signal can become conservative: small positive and negative effects cancel, weakening the RM-weighted term that distinguishes actionable traces inside a GRPO group. Since evaluation is pass@1 with deterministic decoding, optimizing an overly smoothed training-time utility estimate is not necessarily aligned with the final evaluation target. We therefore use $K = 3$ as a practical balance between reward stability, cost, and pass@1 alignment.

Reward component ablations. Table 5 isolates the two key components of the TraceLift reward. The No-uplift variant removes executor grounding from the RM term:

$$\mathcal{R}(P, R) = 0.5\, R_{\text{exec}}(P, R) + 0.5\, \text{RM}_\phi(P, R).$$

The RM-uplift only variant removes the end-to-end verifier anchor:

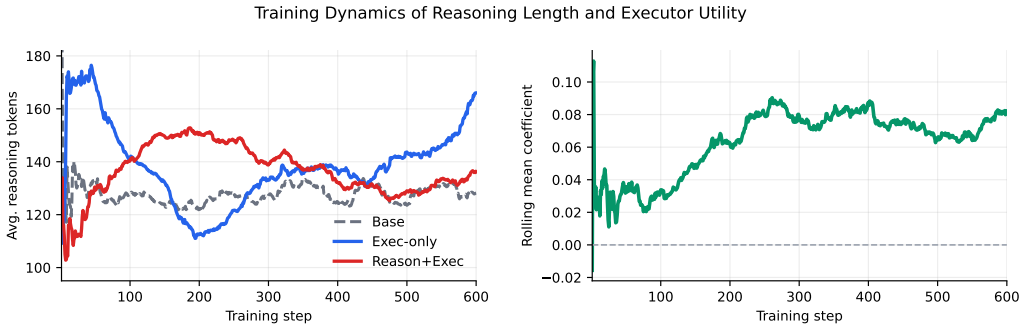
$$\mathcal{R}(P, R) = \text{RM}_\phi(P, R)\, u_{\text{exec}}(P, R).$$

The full reward outperforms the tested ablated variants. Removing uplift turns the RM term into an intrinsic rubric score detached from the executor. This variant is weaker than the full method on every benchmark and even falls below Exec-only on the micro-average. The failure mode is intuitive: a trace can be coherent and rubric-plausible while still being too abstract, misleading, or poorly matched to the executor prompt. Without measured uplift, high-scoring traces that do not improve executor success are not penalized.

The RM-uplift only variant has the opposite failure. It preserves executor grounding but removes the verifier anchor, so it can reward traces that improve over the no-reasoning baseline while still producing incorrect artifacts. Such relative improvement is useful, but not task success. This is visible on MBPP-full and LiveCodeBench, where RM-uplift only improves over Base but remains far below full TraceLift. The full reward achieves the strongest performance among these variants: the verifier term anchors optimization to correct artifacts, while the RM-uplift term filters reasoning credit through rubric quality and executor utility.

3.6 Analysis: Reasoning Length and Executor Utility

A natural alternative explanation is that TraceLift improves performance by simply making the planner produce longer reasoning traces. We test this by measuring reasoning length during training.



(a) Reasoning Length (b) Uplift Coefficient

Figure 2: Training dynamics of reasoning length and executor utility. **Left**: rolling mean of reasoning length for differend methods. The Base is evaluated on the same problem sequence and does not reflect parameter updates. **Right**: rolling mean of the uplift coefficient used in the TraceLift reward.

Table 6: Reasoning length statistics over the final training window. Token counts are computed with the Qwen2.5 tokenizer. The bin columns report percentages over 300 sampled reasoning traces.

Model	Avg. tokens	Median	0–128 tokens	$\geq$ 256 tokens
Base	128.2	104.0	59.00	7.00
Exec-only	166.0	147.5	38.67	12.67
TraceLift	136.3	125.5	53.67	5.00

Table 6 summarizes the last 150 training steps, corresponding to 300 on-policy reasoning samples. The Base row is computed by running the Qwen2.5-7B base planner on the same sequence of training problems, so it reflects problem-sequence variation rather than parameter updates.

The length statistics and training curves reject the verbosity explanation. Exec-only produces longer reasoning than Base, averaging 166.0 tokens with a larger long-tail fraction. In contrast, TraceLift improves task performance while keeping the average length at 136.3 tokens, nearly 30 tokens shorter than Exec-only. Thus, the gain does not come from longer traces or format preference.

Figure 2 further shows that TraceLift increases executor utility without increasing reasoning length. The uplift coefficient gradually becomes more positive during training, while reasoning length remains moderate. This indicates that the planner learns to include information that changes executor behavior, rather than merely expanding the explanation. In other words, the reward shifts the planner toward compact, executor-useful traces.

4 Conclusion

In this paper, we introduced TraceLift, a planner-executor framework that trains reasoning trajectories as executor-consumable intermediate artifacts. TraceLift trains a reasoning planner with verifier feedback on frozen-executor outputs and a Reason RM score weighted by measured executor uplift. We also introduced TraceLift-Groups, a rubric-annotated reason-only dataset that pairs reference trajectories with targeted flawed trajectories for learning intermediate reasoning quality. Experiments on code and math benchmarks show that this training recipe improves fixed two-stage planner-executor systems over execution-only reinforcement learning. Together, these results highlight a practical direction for reasoning supervision: reward trajectories not only by how they read, but by whether they are accurate enough to help the executor produce a correct final artifact.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aieleen Letman, Akhil Mathur, Alan Schelten, Amy Yang, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Tianyang Han, Qing Lian, Rui Pan, Renjie Pi, Jipeng Zhang, Shizhe Diao, Yong Lin, and Tong Zhang. The instinctive bias: Spurious images lead to illusion in mllms. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16163–16177, 2024.

Tianyang Han, Junhao Su, Junjie Hu, Peizhen Yang, Hengyu Shi, Junfeng Luo, and Jialin Gao. Beyond words and pixels: A benchmark for implicit world knowledge reasoning in generative models. *ArXiv*, abs/2511.18271, 2025. URL https://api.semanticscholar.org/CorpusID:283243797.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Yang Wei, and Deheng Ye. RLTF: Reinforcement learning from unit test feedback. *Transactions on Machine Learning Research*, 2023a. URL https://openreview.net/forum?id=hjYmsV6nXZ.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023b.

OpenAI. OpenAI o1 system card. Technical report, OpenAI, 2024. URL https://openai.com/index/openai-o1-system-card/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168.

Renjie Pi, Tianyang Han, Wei Xiong, Jipeng Zhang, Runtao Liu, Rui Pan, and Tong Zhang. Strengthening multimodal large language model with bootstrapped preference optimization. In *European Conference on Computer Vision*, pp. 382–398. Springer, 2024a.

Renjie Pi, Tianyang Han, Jianshu Zhang, Yueqi Xie, Rui Pan, Qing Lian, Hanze Dong, Jipeng Zhang, and Tong Zhang. Mllm-protector: Ensuring mllm's safety without hurting performance. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16012–16027, 2024b.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023.

Amrith Rajagopal Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=A6Y7AqlzLW.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R. Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. Execution-based code generation using deep reinforcement learning. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=0XBuaxqEcG.

Junhao Su, Yuanliang Wan, Junwei Yang, Hengyu Shi, Tianyang Han, Junfeng Luo, and Yurui Qiu. Failure makes the agent stronger: Enhancing accuracy through structured reflection for reliable tool interactions. *arXiv preprint arXiv:2509.18847*, 2025.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 9426–9439, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Baosong Yang, Beichen Ge, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

AI USE STATEMENT

We used generative AI tools to generate perturbed reasoning traces and automatically annotate them with rubric-based quality scores for the construction of TRACELIFT-GROUPS. Details of the generation and annotation procedures are provided in the method section and appendix.

A RELATED WORK

Large Language Models for Reasoning Chain-of-thought prompting shows that explicitly decoding intermediate reasoning can improve multi-step problem solving (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2023). Search and bootstrapping methods further scale reasoning by sampling, refining, or selecting intermediate processes (Yao et al., 2023a; Zelikman et al., 2022). Agentic methods such as ReAct and Reflexion use language-based reasoning to choose subsequent actions or incorporate feedback (Yao et al., 2023b; Shinn et al., 2023). Program-aided methods connect language reasoning with external computation (Gao et al., 2022; Chen et al., 2023). Recent reasoning-oriented models such as OpenAI o1, Qwen, and DeepSeek-R1 further show that explicit reasoning has become central to strong math and code performance (OpenAI, 2024; Yang et al., 2024; 2025; DeepSeek-AI, 2025). These works establish reasoning as more than post-hoc explanation. Our work asks a complementary training question. We study how to reward a reasoning planner so that its intermediate reasoning is not only plausible but also useful to a separate frozen executor under a fixed evaluation protocol.

Agentic reinforcement learning and trajectory-level rewards. Agentic language-model systems increasingly treat problem solving as a trajectory of reasoning, actions, and feedback. ReAct and Reflexion show that language-based reasoning can guide actions and incorporate feedback (Yao et al., 2023b; Shinn et al., 2023). RLHF and preference-optimization methods train policies from learned preferences (Christiano et al., 2017; Ziegler et al., 2019; Ouyang et al., 2022; Rafailov et al., 2023). Verifiable reinforcement learning and execution-based code methods further use answer correctness, compiler signals, or unit tests as rewards (Shao et al., 2024; DeepSeek-AI, 2025; Le et al., 2022; Liu et al., 2023a; Shojaee et al., 2023; Li et al., 2022). Process supervision and reward modeling move beyond final outcomes by assigning credit to intermediate steps or complete solution trajectories (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2023; Wang et al., 2024; Setlur et al., 2025). These works motivate trajectory-level evaluation rather than outcome-only scoring. However, they do not directly evaluate whether the reasoning trajectory is accurate enough to help the downstream executor produce a better artifact. TraceLift instead trains a reasoning planner by combining reason-only quality scoring with measured utility for the frozen executor that consumes the trace.

B LIMITATIONS

The present study focuses on controlled planner-executor evaluations with medium-scale open-weight models, leaving larger model scales as an important direction for future validation. In addition, although TRACELIFT-GROUPS provides targeted reason-only supervision for code and math tasks, future work can further expand the amount and diversity of training data to cover a broader range of reasoning patterns and task domains.

C DATASET CONSTRUCTION PROMPTS

This appendix gives the concrete construction templates used to build the reason-only supervision data. We separate three operations: deterministic reason-only cleaning, localized flawed-trace generation, and rubric annotation. The cleaning stage is rule-based rather than LLM-generated; we therefore present it as an implementation specification instead of a prompt. The flawed-trace and rubric stages use the prompt templates shown below.

C.1 Reason-Only Cleaning Specifications

Code reason-only cleaning specification

Purpose. Convert a raw code reasoning passage into an answer-free planning trace.
Preserve.

- algorithmic route and decomposition;
- data structures, invariants, and state updates;
- boundary cases, type constraints, and complexity considerations;
- executor-actionable implementation constraints.

Remove or truncate.

- complete implementation leakage such as code fences or pasteable code;
- strong code markers such as `def`, `class`, `return`, and `import` when filtering is enabled;
- the exact reference implementation if it appears in the reasoning;
- hardcoded test leakage and post-hoc explanation sections.

Deterministic rules. Cut at `</think>` if present; cut before the exact reference action if it appears; cut before triple-backtick code fences; cut before explanation markers such as `### Explanation`, `## Explanation`, and `**Explanation:**`; strip `<think>` markers and surrounding whitespace.

Math reason-only cleaning specification

Purpose. Convert a raw math solution into an answer-free derivation trace.
Preserve.

- solution strategy and variable definitions;
- key intermediate derivations and equation transformations;
- unit conversions and necessary intermediate quantities.

Remove.

- GSM8K-style final-answer markers such as `#### final_answer`;
- direct final-answer leakage;
- answer-only shortcuts that do not support the derivation.

Deterministic rules. Remove the GSM8K final-answer marker; replace calculator annotations of the form `«expr»value` with `expr`; strip trailing blank lines. Synthetic flawed math traces are loaded from the cleaned negative field and then stripped.

C.2 Localized Flawed-Trace Generation

The flawed-trace generator receives a cleaned reference reasoning passage and injects one localized perturbation. The goal is to produce a plausible, task-relevant trace whose reasoning quality or solution support is weakened without turning it into random text.

C.2.1 Code Flawed-Trace Generation

Code flawed-trace generation: system prompt

```
You will be given one passage.

You need to modify only a small part of this passage and inject an error of the specified
    type.

You should keep all other parts unchanged as much as possible.

Important requirements:
- Do not rewrite the whole passage.
- Only make small local changes.
- Preserve most of the original wording, order, structure, and reasoning style.
```

Table 7: Code perturbation types used for flawed reasoning generation.

Perturbation type	Mutation target
`wrong_algorithm_choice`	Mutate the high-level algorithm or problem decomposition.
`missing_edge_case`	Remove or weaken boundary handling and case coverage.
`off_by_one`	Corrupt index arithmetic or inclusive/exclusive range reasoning.
`incorrect_invariant`	Corrupt a maintained state assumption, loop invariant, or proof invariant.
`complexity_unaware_plan`	Ignore input-scale constraints or propose an infeasible plan.
`pseudo_solution_without_executable_detail`	Replace actionable details with vague pseudo-solution text.
`verbose_irrelevant_explanation`	Replace useful reasoning with verbose off-target discussion.

```
- Keep the same format as the input passage.
- Change only the part needed to inject the requested error type.
- Keep all other parts unchanged whenever possible.
- Do not add meta commentary.
- Do not explain what error you introduced.
- Do not mention that the reasoning is wrong.
- Do not point out contradictions, inconsistencies, or failures in the modified reasoning.
- Write the modified passage as if it were a normal reasoning attempt.
- Do not output code blocks.
- Do not add code or final answers.
- Directly output the modified reasoning passage only.
```

Code flawed-trace generation: user template

```
The error type is: {display_name}
This error type means: {definition}

The passage is:
{raw_reason}
```

Code flawed-trace parsing and filtering

The returned text is wrapped as the mutated reasoning. We reject empty generations and generations containing leakage markers such as triple-backtick code fences, `def`, `class`, `<call>`, or `final code`. During group construction, we additionally drop placeholder negatives, near-duplicates, too-short negatives, and examples without a recognized perturbation type.

C.2.2 Math Flawed-Trace Generation

Table 8: Math perturbation types used for flawed reasoning generation.

Perturbation type	Mutation target
`arithmetic_slip`	Corrupt a small but consequential numeric calculation.
`wrong_operation`	Use the wrong operation, formula, or equation setup.
`dropped_case`	Omit a required condition, case, entity, or final step.
`unit_mismatch`	Confuse units, rates, time spans, or quantities.
`unsupported_jump`	Jump to a conclusion without the needed intermediate support.
`premature_answer`	Stop at an intermediate value and treat it as the final result.
`verbose_content_free`	Add generic filler instead of mathematically useful content.

Math flawed-trace generation: system prompt

```
You will be given one math reasoning passage.

Create an internally coherent but mathematically incorrect reasoning trace.
Inject the specified mathematical reasoning error in 1-4 places in this passage.

Important requirements:
- Do not rewrite the whole passage.
- Preserve most of the original wording, order, structure, and reasoning style.
- Keep the same format as the input passage.
- Change only the part needed to inject the requested error type.
- After introducing the error, update any downstream calculations so the reasoning remains
    self-consistent.
- The final answer implied by the reasoning must be different from the original correct
    answer.
- Keep all unrelated parts unchanged whenever possible.
- Avoid isolated contradictions such as writing 7 * 3 = 22 unless the requested error type
    is specifically arithmetic_slip.
- Do not add meta commentary.
- Do not explain what error you introduced.
- Do not mention that the reasoning is wrong.
- Do not point out contradictions, inconsistencies, or failures in the modified reasoning.
- Write the modified passage as if it were a normal math reasoning attempt.
- Do not output code blocks.
- Do not add a separate final answer line.
- Directly output the modified reasoning passage only.
```

Math flawed-trace generation: user template

```
The math error type is: {display_name}
This error type means: {definition}
Target negative kind slug: {negative_kind}
Mutation target: {mutation_target}
The final answer field is provided only as context and should not be copied as a separate
    final answer: {action_gt}

The math reasoning passage is:
{raw_reason}
```

Math flawed-trace parsing and filtering

The returned text is stored as a cleaned negative reasoning trace. Group construction deduplicates repeated negatives and keeps only groups with at least four valid flawed traces. Positive GSM8K traces are cleaned before perturbation by removing final-answer markers and calculator annotations.

C.3 Rubric Annotation Prompts

The rubric judge is used only to annotate the offline training data. It may receive the final code or answer to assess whether the reasoning supports the final artifact. The trained Reason RM and the online planner reward do not receive final answers, generated code, executor outputs, or verifier outcomes as input.

C.3.1 Code Rubric Judge

Code rubric judge prompt

```
You are judging the quality of reasoning for a code task.

You will be given:
- a coding problem
- an answer-free reasoning text
- the final code / result

Score the reasoning itself instead of the final answer alone.
```

```
Rubric dimensions (0-10, where 10 is best):
1. task_understanding: Does the reasoning correctly understand the task and constraints?
2. plan_quality: Is the proposed approach sensible and appropriate?
3. step_coherence: Are the reasoning steps logically connected and internally consistent?
4. action_support: Does the reasoning actually support the final code/result?
5. non_leakage: Does the reasoning avoid simply dumping the final implementation or over-
     leaking the final action?

Scoring rules:
- Each dimension must be scored from 0 to 10.
- Decimal scores are allowed.
- Use at most one decimal place when needed.
- 10 means excellent reasoning quality on that dimension.
- 0 means the reasoning completely fails on that dimension.

Then produce:
- rubric_score: overall score from 0 to 10 (decimal allowed)
- rubric_label: one of [strong, acceptable, weak, bad]
- rubric_reason: short explanation in 1-3 sentences

Return STRICT JSON only, with this exact schema:
{
  "task_understanding": 0.0,
  "plan_quality": 0.0,
  "step_coherence": 0.0,
  "action_support": 0.0,
  "non_leakage": 0.0,
  "rubric_score": 0.0,
  "rubric_label": "bad",
  "rubric_reason": "..."
}

Problem:
{problem}

Reasoning:
{reason_clean}

Final code/result:
{action_gt}
```

C.3.2 MATH RUBRIC JUDGE

Math rubric judge: system prompt

```
You are an expert math reasoning rubric judge. Evaluate the quality of the reasoning
     process for solving math problems, not just whether the final answer string is present.
```

Math rubric judge: user prompt

```
You are judging the quality of reasoning for a math problem.

You will be given:
- a math problem
- a mathematical reasoning text
- the final answer

Score the reasoning itself instead of the final answer alone. A reasoning trace can deserve
      a low score even when it states the correct answer if the derivation is unsupported,
     incoherent, or mathematically invalid.

Rubric dimensions (0-10, where 10 is best):
1. problem_understanding: Does the reasoning correctly identify the quantities, conditions,
      and goal?
2. solution_strategy: Is the mathematical approach appropriate and efficient for the
     problem?
3. step_coherence: Are the reasoning steps logically connected and internally consistent?
4. calculation_correctness: Are computations, algebraic manipulations, and unit conversions
      correct?
```

```
5. answer_support: Does the reasoning actually justify the final answer without unsupported
      jumps?

Scoring rules:
- Each dimension must be scored from 0 to 10.
- Decimal scores are allowed.
- Use at most one decimal place when needed.
- 10 means excellent reasoning quality on that dimension.
- 0 means the reasoning completely fails on that dimension.

Then produce:
- rubric_score: overall score from 0 to 10 (decimal allowed)
- rubric_label: one of [strong, acceptable, weak, bad]
- rubric_reason: short explanation in 1-3 sentences

Return STRICT JSON only, with this exact schema:
{
  "problem_understanding": 0.0,
  "solution_strategy": 0.0,
  "step_coherence": 0.0,
  "calculation_correctness": 0.0,
  "answer_support": 0.0,
  "rubric_score": 0.0,
  "rubric_label": "bad",
  "rubric_reason": "..."
}

Problem:
{problem}

Reasoning:
{reason_clean}

Final answer:
{action_gt}
```

Table 9: Conversion from raw judge scores to Reason RM training labels. Dimension labels are used by the rubric heads, while the normalized total score is used by the aggregate head.

RM label	Raw judge-score interval
0	$[0, 2)$
1	$[2, 4)$
2	$[4, 6)$
3	$[6, 8)$
4	$[8, 10]$

The exact conversion is

$$y_k = \min(4, \max(0, \lfloor s_k/2 \rfloor)), \qquad s_{\text{total}} = \text{clip}(s_{\text{rubric}}/10, 0, 1),$$

where s_k is a raw dimension score and s_{rubric} is the raw total rubric score.

C.4 Group Schema and Reason RM Input Rendering

Reasoning-group schema

```
{
  "problem_id": "code_000000 | math_gsm8k_000000",
  "source": "code | gsm8k",
  "task_type": "math optional",
  "problem": "...",
  "reference_solution": "...",
  "positive_pool": [
    {
      "reasoning": "...",
      "label": "positive",
      "rubric": {
        "dimension_name": 0,
        "total": 0.0
      },
```

```
      "rubric_label": "positive | strong | acceptable | weak | bad",
      "rubric_score_raw": 10.0,
      "raw_reason_length": 0,
      "clean_reason_length": 0
    }
  ],
  "negative_bank": [
    {
      "reasoning": "...",
      "negative_kind": "...",
      "negative_index": 0,
      "label": "negative",
      "rubric": {
        "dimension_name": 0,
        "total": 0.0
      },
      "rubric_label": "bad",
      "rubric_score_raw": 0.0,
      "rubric_reason": "...",
      "raw_reason_length": 0,
      "clean_reason_length": 0,
      "metadata": {}
    }
  ],
  "metadata": {
    "source_dataset": "...",
    "source_row_index": 0,
    "negative_count": 0,
    "dimension_names": []
  }
}
```


Reason RM input rendering

```
<task>
...
</task>
<type>code|math</type>
<problem>
...
</problem>
<reasoning>
...
</reasoning>
```

The Reason RM input contains only the task type, problem text, and candidate reasoning. It excludes the reference solution, final answer, generated code, executor output, and verifier result. This separation prevents the Reason RM from learning a shortcut based on final-artifact correctness rather than reasoning quality.

D TRACELIFT-GROUPS STATISTICS AND REASON RM VALIDATION

This appendix audits the grouped supervision data used to train the Reason RM and validates whether the trained RM learns the intended reason-quality ordering. The main paper describes the full code-and-math construction. Here we report the detailed exported audit for the GSM8K-derived math split, because this split contains complete per-perturbation metadata and all-negative held-out RM predictions. The goal is not to claim broad reward-model generalization beyond the construction distribution, but to verify that the learned RM reliably distinguishes high-quality reference reasoning from localized flawed traces under the same grouped supervision format used for training.

D.1 GROUPED DATA STATISTICS

Table 10 summarizes the retained math reasoning groups. Starting from 3,000 GSM8K seed problems, the construction keeps 2,989 groups after filtering, corresponding to a retention rate of 99.63%. Each retained group contains one reference trace and at least four flawed traces, with an average of

6.28 flawed traces per problem. Lengths in this appendix are whitespace-token counts computed from the stored reasoning text.

Table 10: Statistics of the audited GSM8K-derived TRACELIFT-GROUPS split.

Split	Seed problems	Kept groups	Reference traces	Flawed traces	Avg. ref len.	Avg. flawed len.
Math	3000	2989	2989	18764	49.95	78.85

The high retention rate indicates that the perturbation pipeline rarely fails to produce enough valid local negatives. The flawed traces are longer on average than the references, mainly because the perturbation set includes verbose content-free reasoning. This length asymmetry is useful for evaluation: a reward model that simply favors longer reasoning would be pulled toward some flawed candidates, whereas a valid reason-quality model should still rank compact reference reasoning above longer but less supportive traces.

D.2 PERTURBATION COVERAGE

Table 11 reports the perturbation distribution and average rubric score of flawed traces. The seven perturbation types are all represented at nontrivial frequency, with the largest type accounting for 15.90% of negatives and the smallest accounting for 10.89%. The average normalized judge scores are low across all perturbation types, confirming that the perturbations reduce reasoning quality while preserving task relevance.

Table 11: Perturbation statistics for flawed math reasoning traces. Judge scores are normalized to $[0, 1]$.

Perturbation type	Count	Share	Avg. judge score	Avg. length
`arithmetic_slip`	2628	14.01%	0.36	49.92
`dropped_case`	2044	10.89%	0.32	55.57
`premature_answer`	2781	14.82%	0.37	44.17
`unit_mismatch`	2616	13.94%	0.25	56.91
`unsupported_jump`	2853	15.21%	0.32	49.19
`verbose_content_free`	2858	15.23%	0.29	235.44
`wrong_operation`	2984	15.90%	0.23	50.22

Two patterns are important. First, the perturbation set is not dominated by a single easy flaw type: arithmetic errors, wrong operations, unit mistakes, dropped cases, unsupported jumps, premature stopping, and verbose non-supportive traces all appear frequently. This matters because the Reason RM should learn a general notion of reasoning support rather than a detector for one synthetic artifact. Second, `verbose_content_free` traces are by far the longest but still receive low judge scores. This directly supports the design goal of discouraging verbosity as a shortcut for reasoning quality.

D.3 RUBRIC LABEL DISTRIBUTION

The rubric labels are produced by discretizing raw judge scores into five classes, from 0 to 4. Table 12 groups these labels into low, middle, good, and strong regions. The distribution shows that the perturbations are local rather than destructive. Many flawed traces still receive high problem-understanding labels, because they often preserve the original problem setup. However, dimensions that depend on actually supporting the solution, especially calculation correctness and answer support, shift strongly toward low labels.

This distribution is desirable for grouped reason-only supervision. If all flawed traces simply misunderstood the problem, the RM could solve the task by detecting superficial topic mismatch. Instead, many flawed traces still look relevant at the problem-understanding level, while failing in coherence, calculation, or answer support. The grouped comparison therefore forces the RM to attend to whether the trace actually sustains a valid solution path.

Table 12: Rubric label distribution over all annotated math candidates. Low combines labels 0 and 1.

Dimension	Low: 0–1	Mid: 2	Good: 3	Strong: 4
Problem understanding	11.5%	13.8%	36.2%	38.5%
Solution strategy	20.8%	20.4%	23.2%	35.7%
Step coherence	52.5%	21.8%	10.1%	15.6%
Calculation correctness	72.1%	8.7%	3.5%	15.7%
Answer support	83.6%	2.4%	0.2%	13.8%

D.4 Held-Out Reason RM Validation

We validate the Reason RM on held-out GSM8K perturbation groups. The validation uses an all-negative protocol: for each held-out problem, the reference trace is compared against all available flawed traces in the group, rather than against a single sampled negative. This directly tests the use case needed by the online reward: the RM should assign higher scores to reliable reasoning than to plausible local perturbations.

Table 13: Held-out Reason RM validation on GSM8K perturbation groups. Pairwise accuracy measures whether the reference trace scores above a flawed trace from the same problem. Group accuracy measures whether the reference trace is the top-scoring candidate in the group.

RM	Groups	Candidates	Neg. pairs	Pairwise acc.	Group acc.	Total Spearman	Total MAE	Total RMSE	Dim. acc.
Math LoRA-CE step504	299	2186	1887	99.15%	96.32%	0.791	0.105	0.156	66.68%

The RM shows strong separation between reference and flawed reasoning. Its mean score for reference traces is 0.933, compared with 0.273 for flawed traces, giving a mean pairwise margin of 0.659. The high pairwise accuracy shows that the RM reliably rejects localized perturbations, while the high group accuracy shows that this ranking remains robust when all flawed candidates compete against the reference simultaneously. The score correlation and low total-score error further indicate that the RM is not only learning a binary reference-versus-flaw separator, but also preserving a useful ordering of reasoning quality.

D.5 Validation by Perturbation Type

Table 14 breaks down pairwise ranking accuracy by perturbation type. The RM performs well across all seven flaw categories, including subtle arithmetic slips and more structural wrong-operation or unit-mismatch errors.

Table 14: Held-out Reason RM ranking accuracy by perturbation type. Mean margin is the average RM score difference between the reference trace and the flawed trace.

Perturbation type	Pairs	Pairwise acc.	Mean margin
`arithmetic_slip`	271	97.79%	0.520
`dropped_case`	195	98.46%	0.614
`premature_answer`	281	99.64%	0.650
`unit_mismatch`	264	99.62%	0.730
`unsupported_jump`	286	99.30%	0.648
`verbose_content_free`	292	99.32%	0.697
`wrong_operation`	298	99.66%	0.733

The lowest accuracy appears on `arithmetic_slip`, which is expected because these perturbations can be extremely local and may leave most of the surrounding derivation intact. Even there, the RM reaches 97.79% pairwise accuracy. The largest margins appear for `wrong_operation` and `unit_mismatch`, where the perturbation changes the mathematical structure or quantity semantics in a way that should strongly reduce executor usefulness. This breakdown is consistent with

the intended role of the RM: it is sensitive not only to fluent formatting, but also to whether the reasoning trace preserves the calculation and support needed for a correct downstream artifact.

D.6 Takeaways

The audit supports three conclusions. First, the grouped construction creates dense same-problem comparisons: almost every seed yields a valid group, and each group contains multiple localized flawed traces. Second, the evaluation covers multiple perturbation types, although it does not rule out shortcuts specific to the construction distribution. Third, the trained Reason RM accurately recovers the intended within-group ordering on held-out perturbation groups, with high pairwise and group-level ranking accuracy. These properties justify using the RM as the intrinsic reasoning-quality component in TraceLift, while the executor-uplift term in the main reward further filters this quality signal by downstream utility.

E Additional Ablations and Training Dynamics

This appendix provides diagnostic views of the ablation and training-dynamics results. The main text reports the absolute benchmark numbers. Here we focus on deltas, reward failure modes, and temporal behavior, which better expose why each component of TraceLift is needed. All code ablations in this section use the Qwen2.5-7B two-stage code setting. During training-time executor comparisons, the with-reasoning and no-reasoning branches use matched sampling settings with temperature 0.5; final evaluation uses greedy decoding with temperature 0.

E.1 Uplift Rollout Count

The uplift term estimates whether a candidate trace changes the frozen executor's success probability relative to a no-reasoning baseline. Table 15 reports drops relative to the main $K = 3$ setting. This delta view makes clear that $K = 3$ is not only best on the aggregate score, but also consistently stronger than both $K = 1$ and $K = 5$ across all four code benchmarks.

Table 15: Effect of the number of executor comparisons used for uplift estimation. Entries are percentage-point changes relative to the main $K = 3$ setting. Negative values indicate a drop from $K = 3$.

Benchmark	$K = 1$ minus $K = 3$	$K = 5$ minus $K = 3$
HumanEval	-4.27	-5.49
HumanEval+	-0.61	-3.65
MBPP-full	-0.40	-0.80
LiveCodeBench	-1.00	-0.50
Micro avg.	-1.14	-1.71

For fixed P and R, the uplift estimator is unbiased for all tested K, and its variance decreases as K increases. Table 15 shows that $K = 3$ performs best in this sweep, but it does not identify the cause of the lower score at $K = 5$. In particular, these results do not establish a systematic bias or attenuation of the expected uplift at larger K.

E.2 Reward Component Diagnostics

Table 16 reports how much each ablated reward drops relative to the full TraceLift reward. Unlike the main ablation table, this view isolates the magnitude and location of each failure mode.

The three ablations fail for different reasons. The No-uplift reward removes the executor-consumption check from the reasoning score. Its large drop on HumanEval indicates that a rubric-good trace is not always an executor-useful trace: it may be coherent and fluent while still omitting the exact edge condition or implementation constraint that the executor needs. The RM-uplift-only reward keeps executor grounding, but removes the verifier anchor. It is closer to full TraceLift on HumanEval and HumanEval+, where relative executor improvement often aligns with pass/fail success, but it loses more on MBPP-full and LiveCodeBench, where a trace can improve the executor

Table 16: Reward-component diagnostics. Entries are percentage-point drops relative to full TraceLift.

Benchmark	No-uplift	RM-uplift only	LLM-as-judge
HumanEval	-6.71	-1.22	-5.49
HumanEval+	-3.65	-0.61	-1.82
MBPP-full	-2.80	-2.60	-2.40
LiveCodeBench	-2.50	-2.75	-4.25
Micro avg.	-3.34	-2.20	-3.34

relative to the no-reasoning branch without making the final artifact correct. This shows that uplift is a useful marginal-utility signal, but it should not replace the task verifier. Finally, the LLM-as-judge variant underperforms because the judge score is not calibrated enough to serve as a dense on-policy reward.

Table 17 summarizes the same point at the aggregate level by comparing each reward with Exec-only and full TraceLift.

Table 17: Aggregate reward diagnostics on Qwen2.5-7B code. Deltas are percentage points on the code micro-average.

Reward	Micro avg.	Δ vs. Exec-only	Δ vs. TraceLift
Exec-only	52.28	0.00	-2.61
No-uplift	51.55	-0.73	-3.34
RM-uplift only	52.69	$+0.41$	-2.20
LLM-as-judge	51.55	-0.73	-3.34
TraceLift	54.89	$+2.61$	0.00

This aggregate view supports the design of Eq. 9. Reason-quality supervision alone is insufficient if it is detached from the executor. Executor uplift alone is insufficient if it is detached from final task success. A generic LLM judge is insufficient if its scores are saturated or poorly calibrated. The full reward combines all three requirements: task success, intrinsic reasoning quality, and measured executor utility.

E.3 Why Direct LLM Judging Is Not Enough

The LLM-as-judge replacement uses the same rubric text as the Reason RM, but queries Qwen2.5-7B-Instruct directly during reward computation. The final score is normalized as $s_{\text{judge}} = \text{rubric_score}/10$ and used in

$$\mathcal{R}(P, R) = 0.5\, R_{\text{exec}}(P, R) + 0.5\, s_{\text{judge}}(P, R)\, u_{\text{exec}}(P, R).$$

The ablation is weaker than the trained Reason RM, and the logged judge scores explain why. Table 18 reports the score saturation diagnostic from on-policy samples.

Table 18: LLM-as-judge score saturation diagnostic from logged on-policy samples.

Diagnostic	Value
Logged samples	600
Samples with $s_{\text{judge}} = 1.0$	573
Saturation rate	95.50%
Mean judge score	0.990

This saturation makes the judge almost non-discriminative inside a GRPO group. When most candidate traces receive near-perfect rubric scores, the reward term $s_{\text{judge}} \cdot u_{\text{exec}}$ collapses toward uplift-only behavior and loses the ability to prefer traces that are cleaner, more complete, or less misleading. In contrast, the trained Reason RM is optimized on same-problem reference-versus-perturbation groups, so its score is shaped to resolve exactly the local distinctions that arise during

Table 19: Ablation results of Qwen2.5-7B LLM-as-judge replacement. All entries are pass@1 percentages.

Benchmark	Base	TraceLift	LLM-as-judge
HumanEval	68.90	**72.56**	67.07
HumanEval+	60.37	**64.02**	62.20
MBPP-full	58.00	**61.20**	58.80
LiveCodeBench	29.50	**36.00**	31.75
Micro avg.	50.49	**54.89**	51.54

planner training. This explains why merely prompting an instruction-tuned model with the rubric does not substitute for a calibrated Reason RM.

E.4 LLM-AS-JUDGE REPLACEMENT.

Table 19 replaces the trained Reason RM with a direct LLM-as-judge score from Qwen2.5-7B-Instruct under the same rubric. The reward is

$$\mathcal{R}(P, R) = 0.5\, R_{\text{exec}}(P, R) + 0.5\, s_{\text{judge}}(P, R)\, u_{\text{exec}}(P, R),$$

where s_{judge} is the normalized rubric score returned by the judge model.

The tested Qwen2.5-7B-Instruct judge configuration achieves a micro-average of 51.54%, compared with 54.89% for TraceLift. Its score saturation, reported in Table 18, limits the quality-ranking information it supplies within a group. The trained Reason RM yields better downstream performance in this comparison; the experiment does not establish lower reward variance or superior calibration on natural on-policy errors, nor does it rule out stronger prompted judges.

E.5 TRAINING-DYNAMICS DIAGNOSTICS

The main text shows the full reasoning-length and uplift curves. Here we summarize the same logs with rolling-window statistics. The rolling mean window is 30 steps. The Base curve is obtained by re-running the Qwen2.5-7B base planner on the same training problem sequence, so it reflects problem-sequence variation rather than parameter updates. Exec-only and TraceLift are on-policy training samples.

Table 20: Rolling-window training dynamics. Token values are Qwen2.5-tokenizer counts. The last column is the rolling mean uplift coefficient used in the TraceLift reward.

Window	Base len.	Exec-only len.	TraceLift len.	TraceLift − Exec-only	Uplift coeff.
Steps 1–100	130.40	158.62	126.01	−32.60	0.0306
Steps 251–350	128.01	131.99	142.18	+10.19	0.0791
Steps 501–600	129.08	148.87	131.64	−17.23	0.0711
Final step	128.15	166.02	136.25	−29.77	0.0820

The dynamics show that TraceLift does not follow a monotonic length-increase path. In the early window, it is shorter than both Base and Exec-only while the uplift coefficient is already positive. In the middle window, length increases temporarily as the planner explores more detailed traces, and the uplift coefficient rises. By the late window and final step, the planner retains the positive uplift signal while returning to much shorter traces than Exec-only. This pattern is consistent with a refinement process: the reward initially permits exploration of useful details, then favors traces that keep the executor-relevant information without accumulating unnecessary explanation.

The final-step contrast is especially informative. Exec-only reaches a rolling length of 166.02 tokens, while TraceLift is 136.25 tokens, nearly 30 tokens shorter, despite achieving better downstream performance. Thus, the additional reward term is not functioning as a verbosity bonus. It acts as a filter that rewards details only when they are both rubric-supported and executor-useful.

F QUALITATIVE CASE STUDIES

This appendix provides qualitative diagnostics for representative code and math examples. The goal is not to repeat the aggregate benchmark results, but to inspect how different training objectives change the information carried by the reasoning trace. Each case is summarized from evaluation artifacts and focuses on the decisive reasoning signal rather than reproducing full model completions. Across cases, the recurring pattern is that TraceLift does not merely produce more text; it tends to preserve the specific constraint, invariant, unit relation, or algebraic dependency that changes the frozen executor's final behavior.

F.1 CODE CASE STUDIES

HumanEval/64: `vowels_count`

Failure type	edge-case handling	duplicate counting
Task signal	Count ordinary vowels, with a special rule for terminal `y`.	
Base behavior	The reasoning covers ordinary vowels but tends to omit the terminal-`y` rule. This gives the executor an underspecified plan.	
Exec-only behavior	The reasoning expands the plan, but can place `y` inside the vowel set and also describe a separate terminal-`y` adjustment. This creates a duplicate-counting path.	
TraceLift behavior	The reasoning keeps the ordinary vowel set as `aeiou` and handles only word-final `y` as a separate boundary case.	

Analysis. This case illustrates why execution-only feedback can be too coarse. A longer trace may mention the missing edge case, but if it encodes the edge case in the wrong form, the executor can implement a systematically wrong counting rule. TraceLift improves the trace by separating the default rule from the exceptional rule. The key gain is therefore not verbosity, but a cleaner partition of cases: ordinary vowels are counted uniformly, and terminal `y` is treated exactly once. This is the kind of local constraint that a rubric-only score may not fully identify unless it is grounded in whether the executor's behavior actually changes.

HumanEvalPlus/55: `fib`

Failure type	infeasible complexity	stronger-test timeout
Task signal	Compute Fibonacci values in a way that remains feasible under stronger tests.	
Base behavior	The reasoning can select the mathematically natural recursive definition, which is correct as a recurrence but inefficient as an implementation plan.	
Exec-only behavior	The trace remains compatible with naive recursion and therefore may still lead the executor to redundant exponential computation.	
TraceLift behavior	The trace specifies an iterative plan, maintaining the two most recent Fibonacci values and updating them in a loop.	

Analysis. This case separates mathematical correctness from executor-useful planning. The recursive definition of Fibonacci is semantically correct, so a surface-level reasoning judge may view it as plausible. However, for code generation, the reasoning trace must also communicate an executable route that satisfies the test regime. TraceLift shifts the trace from a definition-level explanation to an implementation-level invariant: keep two previous values and iterate. This supports the paper's central claim that reasoning quality should be evaluated as an intermediate artifact consumed by an executor, not only as a fluent explanation of the task.

MBPP/16: `text_lowercase_underscore`

Failure type	missing full-string constraint · regex anchoring
Task signal	Check whether a string consists of lowercase letters, one underscore, and lowercase letters.
Base behavior	The trace tends to describe a pattern similar to `[a-z]+_[a-z]+`, which can match a substring instead of the whole input.
Exec-only behavior	The reasoning remains close to the same unanchored pattern and does not reliably force full-string matching.
TraceLift behavior	The trace explicitly specifies the anchored pattern `^[a-z]+_[a-z]+$`.

Analysis. The decisive difference is a small symbolic constraint. Adding anchors does not make the reasoning much longer, but it changes the executor's implementation path from substring search to full-string validation. This is a useful example of executor-consumable reasoning: a compact trace carries exactly the information needed to avoid a common false-positive pattern. It also explains why length alone is a poor proxy for reasoning quality; the important unit of supervision is whether the trace contains the constraint that affects execution.

HumanEval/95: `check_dict_case`

Failure type	missing type guard · invalid-key behavior
Task signal	Determine whether all dictionary keys are consistently lowercase or uppercase, while handling non-string keys correctly.
Base behavior	The trace focuses on checking uppercase/lowercase consistency but can omit the requirement that all keys must first be strings.
Exec-only behavior	The trace may become more compact, but the decisive improvement depends on whether it explicitly includes the non-string guard.
TraceLift behavior	The trace states that every key must be a string before applying lower/upper consistency checks.

Analysis. This case demonstrates the value of type-level constraints. Without the string-key guard, the executor may write a solution that works for ordinary dictionaries but fails on adversarial or edge inputs. TraceLift improves the trace by making the guard a prerequisite rather than an afterthought. The reasoning is not valuable because it explains the task at length; it is valuable because it orders the checks correctly: validate key types first, then check case consistency.

F.2 Math Case Studies

GSM8K `gsm8k_test_0038`: weekly running hours

Failure type	unit aggregation · rate denominator error
Task signal	Aggregate the total weekly running time before dividing total distance by time.
Base behavior	The executor predicts a value corresponding to an incorrect denominator, indicating that the weekly hours are not aggregated correctly.
Exec-only behavior	The reasoning still fails to supply the correct total-time structure and can lead to another wrong denominator.
TraceLift behavior	The trace computes the weekly total as $3 + 1.5 + 1.5 = 6$ hours and then evaluates $60/6 = 10$.

Analysis. The important reasoning signal is the construction of the denominator. The problem is not solved by recognizing a generic distance-rate-time template; the executor needs the correct unit aggregation before applying the division. TraceLift provides this intermediate quantity explicitly, which makes the final operation unambiguous. This is a typical executor-grounding benefit: the trace does not merely state a formula, but supplies the quantity that the executor would otherwise infer incorrectly.

GSM-Hard `gsm_hard_0065`: total pets

Failure type	dropped entity	large-number stress
Task signal	Compute the number of pets for Cindy, Marcia, and Jan, then sum all three quantities.	
Base behavior	The trace can omit one participant from the final aggregation, producing a total that excludes Cindy.	
Exec-only behavior	The same dropped-entity pattern remains possible under execution-only training.	
TraceLift behavior	The trace separately computes Cindy, Marcia, and Jan, and then sums the three values to obtain the final total.	

Analysis. This case shows a failure that final-answer reward alone may not localize: the arithmetic can be internally consistent after one entity has been dropped. The trace appears procedural, but it no longer supports the full problem. TraceLift improves the reasoning by preserving the entity inventory throughout the derivation. This is exactly the kind of support relation targeted by the Reason RM rubric: a trace should justify the answer with all required quantities, not merely perform plausible arithmetic on a subset of them.

GSM-Hard `gsm_hard_0225`: bakery afternoon sales

Failure type	dropped condition	intermediate-state tracking
Task signal	Track morning sales, remaining loaves, and the specified afternoon share.	
Base behavior	The reasoning can jump to an answer after using only part of the sales process.	
Exec-only behavior	The executor may still confuse the leftover quantity with the afternoon-sold quantity.	
TraceLift behavior	The trace maintains the intermediate state: first compute what remains after morning sales, then apply the afternoon fraction to the remaining loaves.	

Analysis. This example highlights why step support matters. The final answer depends on applying an operation to the correct intermediate state, not merely on choosing the right operation somewhere in the solution. TraceLift makes the state transition explicit. This helps the executor avoid a common shortcut: treating the remaining amount and the sold amount as interchangeable. The trace is therefore useful because it constrains the order and target of operations.

SVAMP `svamp_test_0034`: Jake's balloons

Failure type	target-entity scope	incomplete accumulation
Task signal	Combine Jake's original balloons with the additional balloons bought later.	
Base behavior	The trace may fail to preserve the accumulation target across the story.	
Exec-only behavior	The executor answers using only the newly bought balloons, focusing on the local event rather than Jake's total.	
TraceLift behavior	The trace keeps Jake as the target entity and adds the original 6 balloons to the 3 bought later, yielding 9.	

Analysis. This is a scope-tracking failure. The local action "bought 3 balloons" is salient, but the question asks for Jake's total. TraceLift improves the trace by anchoring the computation to the target entity and maintaining the cumulative quantity. The example is useful because the corrected reasoning is simple and compact; the gain comes from selecting the right referent and preserving it across the calculation.

MATH500 `math500_test_0058`: evaluating $99^2 + 99 + 1$

Failure type	arithmetic shortcut	incorrect expansion
Task signal	Evaluate the expression exactly without rounding or approximation.	
Base behavior	The executor rounds or approximates the square, leading to an answer near 10000.	
Exec-only behavior	The trace still permits an arithmetic slip, producing 9902.	
TraceLift behavior	The trace expands $(100 - 1)^2 = 9801$, then computes $9801 + 99 + 1 = 9901$.	

Analysis. This case is small but diagnostic. The executor does not need a long derivation; it needs the exact decomposition that prevents approximation. TraceLift supplies a stable arithmetic route by rewriting 99^2 around 100. The improvement illustrates a general math pattern: useful reasoning often consists of a compact intermediate representation that removes ambiguity from the final computation.

MATH500 `math500_test_0131`: polynomial symmetry

Failure type	false symmetry lost odd term
Task signal	Account for both even and odd components of the polynomial.
Base behavior	The reasoning can treat the polynomial as if it were even, leading to a symmetric conclusion.
Exec-only behavior	The same shortcut remains: the executor predicts the value implied by false symmetry.
TraceLift behavior	The trace separates the odd linear term from the even components and derives $f(3) = 8$.

Analysis. The key error is structural rather than numeric. Treating the polynomial as even discards the contribution of the odd term, so subsequent arithmetic can be consistent but based on the wrong invariant. TraceLift prevents this by making the asymmetry explicit. This is a strong example for executor-grounded supervision: a fluent explanation of symmetry is actively harmful if the executor consumes it as a plan, whereas a short trace that identifies the non-even component changes the solution path.

F.3 Cross-Case Patterns

Table 21 summarizes the mechanism-level patterns across the case studies. These patterns are more informative than raw length or fluency because they identify what the executor needs from the trace.

Table 21: Mechanism-level patterns observed in qualitative cases.

Pattern	Typical failure under weaker traces	TraceLift trace behavior
Boundary condition	The trace mentions the common case but misses or misrepresents the exception.	Keeps the default rule and exception as separate executor-actionable cases.
Type or format guard	The trace describes the main operation but omits validation constraints.	Places validation before the main operation, guiding the executor's control flow.
Complexity constraint	The trace gives a mathematically valid but computationally infeasible plan.	Converts the idea into an efficient implementation invariant.
Entity or unit tracking	The trace uses the right operation on the wrong quantity or subset.	Preserves the target entity, unit, or intermediate state until the final operation.
Algebraic structure	The trace applies a tempting shortcut that loses an asymmetric term or exact value.	Names the structural dependency that determines the final answer.

The case studies support three qualitative conclusions. First, TraceLift traces are often valuable because they remove a specific ambiguity that the executor would otherwise resolve incorrectly. Second, the improvements are local and actionable: a regex anchor, a string-key guard, an iterative invariant, a unit denominator, or an odd polynomial term. Third, execution-only training can improve final accuracy but does not explicitly distinguish useful detail from misleading or redundant detail. The executor-grounded reward addresses this gap by crediting reasoning that is both rubric-supported and empirically useful to the frozen executor.

G Theoretical Analysis

This appendix analyzes the reward used by TraceLift in the fixed planner-executor setting. The goal is not to prove global convergence of GRPO or to claim that the training-time stochastic executor

distribution exactly equals the deterministic evaluation distribution. Instead, we analyze the credit-assignment structure induced by Eq. 9. The analysis shows that TraceLift can be interpreted as a quality-weighted conditional executor-utility objective: the verifier term anchors training to task success, while the Reason RM term modulates the marginal utility of the reasoning trace for the frozen executor.

G.1 Formal Setting

Let $P \sim \mathcal{D}$ be a problem and let a planner policy sample a reasoning trace $R \sim \pi_\theta(\cdot \mid P)$. The executor is frozen. For a decoding temperature τ, denote by

$$A_\tau(P, R; \xi) \tag{10}$$

the final artifact produced by the executor from problem P and reasoning trace R under executor randomness ξ. The no-reasoning input is denoted by $\varnothing$. The verifier is deterministic given the artifact:

$$V(A) \in \{0, 1\}. \tag{11}$$

For code, V is executable-test success; for math, V is answer matching. Define the executor success probability under the training-time executor distribution as

$$q_{\mathrm{tr}}(P, R) = \mathbb{P}_\xi[V(A_{\tau_{\mathrm{tr}}}(P, R; \xi)) = 1]\,, \tag{12}$$

where $\tau_{\mathrm{tr}} = 0.5$ in our training-time rollout and uplift estimation. The no-reasoning success probability is

$$q_{\mathrm{tr}}(P, \varnothing) = \mathbb{P}_\xi[V(A_{\tau_{\mathrm{tr}}}(P, \varnothing; \xi)) = 1]\,. \tag{13}$$

The training-time executor uplift of a trace is

$$u_{\mathrm{tr}}(P, R) = q_{\mathrm{tr}}(P, R) - q_{\mathrm{tr}}(P, \varnothing). \tag{14}$$

Since both terms are probabilities, $u_{\mathrm{tr}}(P, R) \in [-1, 1]$.

Let

$$m_\phi(P, R) = \mathrm{RM}_\phi(P, R) \in [0, 1] \tag{15}$$

be the Reason RM score. The RM is fixed during planner optimization and receives only the problem and reasoning trace, not executor outputs or verifier outcomes.

Assumption 1 (Fixed-executor reward analysis)**.** The following analysis assumes:

1. the executor parameters and executor prompt are fixed during planner training;
2. with-reasoning and no-reasoning executor branches use matched training-time decoding settings;
3. verifier outputs are binary and deterministic given the final artifact;
4. the Reason RM score lies in $[0, 1]$ and is fixed during each planner update;
5. the reward analyzed is Eq. 9:

$$\mathcal{R}(P, R) = 0.5\, R_{\mathrm{exec}}(P, R) + 0.5\, \mathrm{RM}_\phi(P, R)\, u_{\mathrm{tr}}(P, R). \tag{16}$$

The analysis is therefore about the training-time reward distribution. Final evaluation uses temperature 0 greedy decoding and is assessed empirically in the main experiments.

G.2 Executor Uplift as a Conditional Treatment Effect

The reasoning trace can be viewed as a treatment applied to a frozen executor. For the same problem P and the same executor, we compare the potential outcome with trace R against the potential outcome with the reasoning field omitted. Under matched decoding settings, the only intervention is whether the executor receives the trace.

Definition 1 (Executor treatment effect)**.** For a fixed problem P and trace R, the training-time conditional executor treatment effect is

$$u_{\mathrm{tr}}(P, R) = q_{\mathrm{tr}}(P, R) - q_{\mathrm{tr}}(P, \varnothing). \tag{17}$$

A positive value means that the trace improves the frozen executor's success probability relative to the no-reasoning prompt. A negative value means that the trace harms the executor.

During training, TraceLift estimates this quantity by repeated executor samples:

$$\widehat{u}_K(P, R) = \frac{1}{K}\sum_{k=1}^{K} Y_k^R - \frac{1}{K}\sum_{k=1}^{K} Y_k^0, \tag{18}$$

where

$$\begin{aligned} Y_k^R &= V(A_{\tau_{\mathrm{tr}}}(P, R; \xi_k^R)), \\ Y_k^0 &= V(A_{\tau_{\mathrm{tr}}}(P, \varnothing; \xi_k^0)). \end{aligned} \tag{19}$$

Proposition 1 (Unbiasedness and variance of the uplift estimator)**.** Assume that $\{Y_k^R\}_{k=1}^K$ are i.i.d. Bernoulli with mean $q_{\mathrm{tr}}(P, R)$ and $\{Y_k^0\}_{k=1}^K$ are i.i.d. Bernoulli with mean $q_{\mathrm{tr}}(P, \varnothing)$. If the two branches are sampled independently, then

$$\mathbb{E}[\widehat{u}_K(P, R)] = u_{\mathrm{tr}}(P, R), \tag{20}$$

and

$$\mathrm{Var}[\widehat{u}_K(P, R)] = \frac{q_{\mathrm{tr}}(P, R)(1 - q_{\mathrm{tr}}(P, R)) + q_{\mathrm{tr}}(P, \varnothing)(1 - q_{\mathrm{tr}}(P, \varnothing))}{K} \le \frac{1}{2K}. \tag{21}$$

Proof. By linearity of expectation,

$$\begin{aligned} \mathbb{E}[\widehat{u}_K(P, R)] &= \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}[Y_k^R] - \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}[Y_k^0] \\ &= q_{\mathrm{tr}}(P, R) - q_{\mathrm{tr}}(P, \varnothing) \\ &= u_{\mathrm{tr}}(P, R). \end{aligned} \tag{22}$$

For independent branches,

$$\mathrm{Var}[\widehat{u}_K(P, R)] = \mathrm{Var}\left[\frac{1}{K}\sum_{k=1}^{K} Y_k^R\right] + \mathrm{Var}\left[\frac{1}{K}\sum_{k=1}^{K} Y_k^0\right]. \tag{23}$$

Since the samples are i.i.d.,

$$\mathrm{Var}\left[\frac{1}{K}\sum_{k=1}^{K} Y_k^R\right] = \frac{1}{K} q_{\mathrm{tr}}(P, R)(1 - q_{\mathrm{tr}}(P, R)), \tag{24}$$

and similarly for the no-reasoning branch. Therefore,

$$\mathrm{Var}[\widehat{u}_K(P, R)] = \frac{q_{\mathrm{tr}}(P, R)(1 - q_{\mathrm{tr}}(P, R)) + q_{\mathrm{tr}}(P, \varnothing)(1 - q_{\mathrm{tr}}(P, \varnothing))}{K}. \tag{25}$$

For any Bernoulli mean $p \in [0, 1]$, $p(1 - p) \le 1/4$, so

$$\mathrm{Var}[\widehat{u}_K(P, R)] \le \frac{1/4 + 1/4}{K} = \frac{1}{2K}. \tag{26}$$

□

Remark 1 (Effect of clipping)**.** Because both empirical success rates lie in $[0, 1]$, their difference lies in $[-1, 1]$. Thus, for binary verifiers and success-rate estimates, the clipping in Eq. 8 is inactive mathematically. It is an implementation safeguard and does not change the estimator under the assumptions above.

G.3 TRACELIFT AS A QUALITY-WEIGHTED UPLIFT OBJECTIVE

Let the single execution score used in the reward be

$$X(P, R) = V(A_{\tau_{\mathrm{tr}}}(P, R; \xi)), \tag{27}$$

so that

$$\mathbb{E}[X(P, R) \mid P, R] = q_{\mathrm{tr}}(P, R). \tag{28}$$

The population version of the TraceLift reward is obtained by taking expectation over executor randomness.

Proposition 2 (Expected TraceLift reward)**.** Under Assumption 1,

$$\mathbb{E}[\mathcal{R}(P, R) \mid P, R] = 0.5\, q_{\mathrm{tr}}(P, \varnothing) + 0.5\,(1 + m_\phi(P, R))\, u_{\mathrm{tr}}(P, R). \tag{29}$$

Proof. By Eq. 9,

$$\mathcal{R}(P, R) = 0.5\, X(P, R) + 0.5\, m_\phi(P, R)\, \widehat{u}_K(P, R). \tag{30}$$

Taking conditional expectation and using Proposition 1,

$$\mathbb{E}[\mathcal{R}(P, R) \mid P, R] = 0.5\, q_{\mathrm{tr}}(P, R) + 0.5\, m_\phi(P, R)\, u_{\mathrm{tr}}(P, R). \tag{31}$$

Since

$$q_{\mathrm{tr}}(P, R) = q_{\mathrm{tr}}(P, \varnothing) + u_{\mathrm{tr}}(P, R), \tag{32}$$

we substitute:

$$\begin{aligned} \mathbb{E}[\mathcal{R}(P, R) \mid P, R] &= 0.5\,\big(q_{\mathrm{tr}}(P, \varnothing) + u_{\mathrm{tr}}(P, R)\big) + 0.5\, m_\phi(P, R)\, u_{\mathrm{tr}}(P, R) \\ &= 0.5\, q_{\mathrm{tr}}(P, \varnothing) + 0.5\,(1 + m_\phi(P, R))\, u_{\mathrm{tr}}(P, R). \end{aligned} \tag{33}$$

□

This identity is the central theoretical interpretation of TraceLift. The no-reasoning executor success $q_{\mathrm{tr}}(P, \varnothing)$ is a problem-level constant. The trace-dependent part is

$$0.5\,(1 + m_\phi(P, R))\, u_{\mathrm{tr}}(P, R). \tag{34}$$

Thus, TraceLift does not add an arbitrary preference bonus. It modulates the marginal executor utility of the trace by a bounded reasoning-quality score.

G.4 EFFECT UNDER GRPO GROUP NORMALIZATION

GRPO normalizes rewards within a group of traces sampled for the same problem. Therefore, problem-level constants do not affect the relative advantage.

Lemma 1 (Problem-level constants cancel under group normalization)**.** For a fixed problem P, let a group of rewards be $r_i = c(P) + s_i$ for $i = 1, \ldots, G$, where $c(P)$ is independent of i. Let

$$\hat{A}_i(r) = \frac{r_i - \mathrm{mean}_j(r_j)}{\mathrm{std}_j(r_j) + \delta}. \tag{35}$$

Then

$$\hat{A}_i(r) = \hat{A}_i(s). \tag{36}$$

Proof. The group mean satisfies

$$\begin{aligned} \mathrm{mean}_j(r_j) &= \mathrm{mean}_j(c(P) + s_j) \\ &= c(P) + \mathrm{mean}_j(s_j). \end{aligned} \tag{37}$$

Thus,

$$\begin{aligned} r_i - \mathrm{mean}_j(r_j) &= c(P) + s_i - \big(c(P) + \mathrm{mean}_j(s_j)\big) \\ &= s_i - \mathrm{mean}_j(s_j). \end{aligned} \tag{38}$$

The standard deviation is unchanged by adding a constant:

$$\mathrm{std}_j(r_j) = \mathrm{std}_j(s_j). \tag{39}$$

Substituting into the definition of $\hat{A}_i$ gives the result. □

Applying Lemma 1 to the population rewards in Proposition 2, the problem-level constant cancels when these population rewards are group-normalized. Their trace-dependent component is proportional to

$$S_{\mathrm{TL}}(P, R) = (1 + m_\phi(P, R))\, u_{\mathrm{tr}}(P, R), \tag{40}$$

with proportionality factor 0.5. This statement concerns population rewards; it does not imply an equivalence for the expected finite-sample GRPO advantages, because group normalization is nonlinear.

Corollary 1 (Quality-gated executor utility)**.** For a fixed problem P, the trace-dependent component of the expected TraceLift reward is proportional to a quality-weighted executor utility term:

$$(1 + m_\phi(P, R))\, u_{\mathrm{tr}}(P, R). \tag{41}$$

Moreover,

$$\frac{\partial}{\partial m_\phi}\mathbb{E}[\mathcal{R}(P, R) \mid P, R] = 0.5\, u_{\mathrm{tr}}(P, R). \tag{42}$$

Therefore:

1. if $u_{\mathrm{tr}}(P, R) > 0$, a higher RM score increases the trace's expected reward;
2. if $u_{\mathrm{tr}}(P, R) = 0$, the RM score alone creates no trace-specific expected reward;
3. if $u_{\mathrm{tr}}(P, R) < 0$, a higher RM score decreases the trace's expected reward.

Proof. The derivative follows directly from Proposition 2:

$$\mathbb{E}[\mathcal{R}(P, R) \mid P, R] = 0.5\, q_{\mathrm{tr}}(P, \varnothing) + 0.5\, (1 + m_\phi(P, R))\, u_{\mathrm{tr}}(P, R). \tag{43}$$

Holding P and R fixed except for the RM score,

$$\frac{\partial}{\partial m_\phi}\mathbb{E}[\mathcal{R}(P, R) \mid P, R] = 0.5\, u_{\mathrm{tr}}(P, R). \tag{44}$$

The three cases follow from the sign of $u_{\mathrm{tr}}(P, R)$. □

This corollary formalizes the intended behavior. A fluent trace with no measured effect on the executor does not receive extra trace-specific credit from the RM score. A harmful trace receives negative uplift, so multiplying by the RM score does not hide the harm; it strengthens the penalty for traces that look high-quality but actually mislead the executor.

G.5 Pairwise Credit Assignment

Consider two candidate traces R_a and R_b for the same problem P. Let

$$u_a = u_{\mathrm{tr}}(P, R_a), \qquad u_b = u_{\mathrm{tr}}(P, R_b), \tag{45}$$

and

$$m_a = m_\phi(P, R_a), \qquad m_b = m_\phi(P, R_b). \tag{46}$$

By Proposition 2 and Lemma 1, the expected trace-specific reward difference is

$$\Delta_{\mathrm{TL}} = 0.5\, [(1 + m_a)u_a - (1 + m_b)u_b]\,. \tag{47}$$

For execution-only training, the expected trace-specific difference is

$$\Delta_{\mathrm{exec}} = u_a - u_b, \tag{48}$$

because the no-reasoning baseline $q_{\mathrm{tr}}(P, \varnothing)$ is constant for the problem.

Proposition 3 (Preference under aligned quality and utility)**.** If

$$u_a \geq u_b \geq 0 \qquad \text{and} \qquad m_a \geq m_b, \tag{49}$$

then

$$\Delta_{\mathrm{TL}} \geq 0. \tag{50}$$

If at least one of the inequalities is strict and either $u_b > 0$ or $u_a > u_b$, then $\Delta_{\mathrm{TL}} > 0$.

Proof. We expand:

$$(1+m_a)u_a - (1+m_b)u_b = u_a - u_b + m_a u_a - m_b u_b. \tag{51}$$

Rewrite the second difference:

$$m_a u_a - m_b u_b = m_a(u_a - u_b) + (m_a - m_b)u_b. \tag{52}$$

Therefore,

$$(1+m_a)u_a - (1+m_b)u_b = (u_a - u_b) + m_a(u_a - u_b) + (m_a - m_b)u_b. \tag{53}$$

Under the assumptions $u_a \geq u_b \geq 0$ and $m_a \geq m_b$, all three terms are nonnegative. Thus the whole expression is nonnegative, and so $\Delta_{\mathrm{TL}} \geq 0$. If $u_a > u_b$, then the first term is positive. If $m_a > m_b$ and $u_b > 0$, then the third term is positive. Hence the strict case follows. □

Corollary 2 (Resolution when outcome utility is tied)**.** If $u_a = u_b = u > 0$, then

$$\Delta_{\mathrm{TL}} = 0.5\, u\, (m_a - m_b). \tag{54}$$

Thus, among traces with the same positive executor uplift, TraceLift prefers the one with higher reasoning-quality score.

Proof. Substitute $u_a = u_b = u$:

$$\begin{aligned} \Delta_{\mathrm{TL}} &= 0.5\, [(1+m_a)u - (1+m_b)u] \\ &= 0.5\, u\, (m_a - m_b). \end{aligned} \tag{55}$$

□

Corollary 3 (Separation of useful and harmful traces)**.** If $u_a > 0$ and $u_b \leq 0$, then

$$\Delta_{\mathrm{TL}} > 0 \tag{56}$$

for all $m_a, m_b \in [0,1]$.

Proof. Since $m_a \in [0,1]$, $(1+m_a)u_a > 0$. Since $m_b \in [0,1]$ and $u_b \leq 0$, $(1+m_b)u_b \leq 0$. Hence

$$(1+m_a)u_a - (1+m_b)u_b > 0, \tag{57}$$

and therefore $\Delta_{\mathrm{TL}} > 0$. □

These pairwise results show what the product term contributes. It does not replace executor utility. Instead, it refines executor utility by preferring traces that are both useful and reason-quality aligned, while still separating positive-uplift traces from harmful traces.

G.6 Why the Ablated Rewards Are Weaker

The main ablations can be understood by comparing the population reward decompositions.

G.6.1 No-uplift reward

The No-uplift ablation uses

$$\mathcal{R}_{\text{no-uplift}}(P,R) = 0.5\, X(P,R) + 0.5\, m_\phi(P,R). \tag{58}$$

Taking expectation,

$$\mathbb{E}[\mathcal{R}_{\text{no-uplift}}(P,R) \mid P,R] = 0.5\, q_{\mathrm{tr}}(P,\varnothing) + 0.5\, (u_{\mathrm{tr}}(P,R) + m_\phi(P,R))\,. \tag{59}$$

After removing the problem-level constant, the trace-specific score is

$$S_{\text{no-uplift}}(P,R) = u_{\mathrm{tr}}(P,R) + m_\phi(P,R). \tag{60}$$

Proposition 4 (No-uplift can reward executor-useless traces)**.** For two traces R_a, R_b for the same problem, the No-uplift score prefers R_a over R_b whenever

$$m_a - m_b > u_b - u_a. \tag{61}$$

Therefore, a trace with lower executor uplift can be preferred solely because it has a higher RM score.

Proof. No-uplift prefers R_a over R_b when

$$u_a + m_a > u_b + m_b. \tag{62}$$

Rearranging gives

$$m_a - m_b > u_b - u_a. \tag{63}$$

This condition can hold even when $u_a < u_b$, so the reward can prefer a lower-utility trace if its RM score is sufficiently higher. □

This formalizes the observed weakness of removing uplift. The RM score is no longer grounded in whether the frozen executor actually benefits from the trace. A fluent or rubric-plausible trace can receive credit even when its executor treatment effect is zero or negative.

G.6.2 RM-UPLIFT-ONLY REWARD

The RM-uplift-only ablation uses

$$\mathcal{R}_{\text{RM-uplift}}(P, R) = m_\phi(P, R)\, u_{\text{tr}}(P, R). \tag{64}$$

This preserves executor grounding, but removes the direct verifier anchor. The trace-specific score is

$$S_{\text{RM-uplift}}(P, R) = m_\phi(P, R)\, u_{\text{tr}}(P, R). \tag{65}$$

Proposition 5 (Verifier-anchor robustness of the full reward)**.** For the full TraceLift reward, the derivative of the expected trace-specific score with respect to executor uplift is

$$\frac{\partial}{\partial u_{\text{tr}}} [0.5(1 + m_\phi)u_{\text{tr}}] = 0.5(1 + m_\phi) \in [0.5, 1]. \tag{66}$$

For RM-uplift-only, the corresponding derivative is

$$\frac{\partial}{\partial u_{\text{tr}}} [m_\phi u_{\text{tr}}] = m_\phi \in [0, 1]. \tag{67}$$

Thus, the full reward always retains at least half-strength direct sensitivity to executor utility, even when the RM score is underestimated, whereas RM-uplift-only can suppress the utility signal entirely when m_ϕ is near zero.

Proof. Both derivatives follow directly from the displayed scores. Since $m_\phi \in [0, 1]$,

$$0.5(1 + m_\phi) \in [0.5, 1], \tag{68}$$

while

$$m_\phi \in [0, 1]. \tag{69}$$

If $m_\phi = 0$, the RM-uplift-only derivative is zero, whereas the full reward derivative is 0.5. □

This explains why the verifier term is not redundant. The RM-uplift product is useful as a reasoning-quality filter, but the direct verifier term ensures that training remains anchored to final task success even when the RM score is imperfect.

G.6.3 LLM-AS-JUDGE REPLACEMENT

A direct LLM-as-judge replacement uses the same form as TraceLift but substitutes a prompted judge score $s_{\text{judge}}(P, R)$ for the trained RM score. The ablation is weak when the judge score saturates, because a nearly constant score cannot supply within-group quality information.

Lemma 2 (Effect of a constant judge score)**.** Suppose that, for all traces in a GRPO group for the same problem P, the judge score is constant:

$$s_{\text{judge}}(P, R_i) = c \qquad \text{for all } i, \tag{70}$$

where $c \in [0, 1]$. Then the expected judge-based reward is a positive affine transformation of the execution success probability $q_{\text{tr}}(P, R_i)$ within that group:

$$\mathbb{E}[\mathcal{R}_{\text{judge}}(P, R_i) \mid P, R_i] = 0.5(1 + c)\, q_{\text{tr}}(P, R_i) - 0.5c\, q_{\text{tr}}(P, \varnothing). \tag{71}$$

Consequently, the judge score contributes no additional trace-quality ranking inside the group.

Proof. The judge-based reward expectation is

$$0.5\, q_{\mathrm{tr}}(P, R_i) + 0.5\, c\, \big(q_{\mathrm{tr}}(P, R_i) - q_{\mathrm{tr}}(P, \varnothing)\big). \tag{72}$$

Expanding,

$$0.5\, q_{\mathrm{tr}}(P, R_i) + 0.5c\, q_{\mathrm{tr}}(P, R_i) - 0.5c\, q_{\mathrm{tr}}(P, \varnothing) = 0.5(1 + c)\, q_{\mathrm{tr}}(P, R_i) - 0.5c\, q_{\mathrm{tr}}(P, \varnothing). \tag{73}$$

For fixed P, the second term is constant across i, and $0.5(1 + c) > 0$. Therefore the reward ranks traces exactly as $q_{\mathrm{tr}}(P, R_i)$ ranks them and adds no independent quality signal. □

Corollary 4 (Saturated judges reduce to execution-only ranking)**.** If a prompted judge returns nearly the same score for most on-policy traces, then the judge-based reward approaches a positively scaled execution-only reward under GRPO group normalization. It cannot reliably distinguish fluent but flawed traces from genuinely executor-useful reasoning.

Proof. If $s_{\mathrm{judge}}(P, R_i) = c$ exactly, Lemma 2 gives a positive affine transformation of $q_{\mathrm{tr}}(P, R_i)$ within the group. Adding a group-level constant has no effect by Lemma 1. Positive scaling changes the magnitude of normalized advantages by a group-wide factor but does not introduce any new ranking signal. If the judge scores are nearly constant, the same conclusion holds approximately. □

This analysis explains why a trained Reason RM can be more useful than direct rubric prompting. The RM is trained on same-problem reference-versus-perturbation groups, which makes its score discriminative in precisely the local comparisons used by GRPO. A saturated judge score collapses this extra signal.

G.7 Finite-Sample Reward Noise

The reward used during GRPO is estimated from finite executor samples. The uplift estimator is unbiased, but its variance affects how reliably the policy can identify better traces inside a group.

Assume that the single execution score $X(P, R)$ and the K samples used for $\widehat{u}_K(P, R)$ are independent. This independence assumption is made only for the variance calculation below; the expectation analysis above does not require it.

Proposition 6 (Variance bound for the sampled TraceLift reward)**.** Let

$$\widehat{\mathcal{R}}(P, R) = 0.5\, X(P, R) + 0.5\, m_\phi(P, R)\, \widehat{u}_K(P, R). \tag{74}$$

Under the independence assumptions of Proposition 1,

$$\mathrm{Var}[\widehat{\mathcal{R}}(P, R) \mid P, R] \le \frac{1}{16} + \frac{m_\phi(P, R)^2}{8K} \le \frac{1}{16} + \frac{1}{8K}. \tag{75}$$

Proof. Using independence between $X(P, R)$ and $\widehat{u}_K(P, R)$,

$$\mathrm{Var}[\widehat{\mathcal{R}}(P, R) \mid P, R] = 0.25\ \mathrm{Var}[X(P, R)] + 0.25\, m_\phi(P, R)^2\ \mathrm{Var}[\widehat{u}_K(P, R)]. \tag{76}$$

Since $X(P, R)$ is Bernoulli with mean $q_{\mathrm{tr}}(P, R)$,

$$\mathrm{Var}[X(P, R)] = q_{\mathrm{tr}}(P, R)(1 - q_{\mathrm{tr}}(P, R)) \le \frac{1}{4}. \tag{77}$$

By Proposition 1,

$$\mathrm{Var}[\widehat{u}_K(P, R)] \le \frac{1}{2K}. \tag{78}$$

Therefore,

$$\begin{aligned} \mathrm{Var}[\widehat{\mathcal{R}}(P, R) \mid P, R] &\le 0.25 \cdot \frac{1}{4} + 0.25\, m_\phi(P, R)^2 \cdot \frac{1}{2K} \\ &= \frac{1}{16} + \frac{m_\phi(P, R)^2}{8K}. \end{aligned} \tag{79}$$

Because $m_\phi(P, R) \in [0, 1]$,

$$\frac{1}{16} + \frac{m_\phi(P, R)^2}{8K} \le \frac{1}{16} + \frac{1}{8K}. \tag{80}$$

□

The bound shows the role of the rollout count K: increasing K reduces the variance of the uplift component, but it does not remove the variance of the single verifier outcome. Therefore, beyond a moderate K, the marginal benefit of additional executor comparisons can be limited, especially when training optimizes a stochastic temperature-0.5 executor utility while evaluation measures deterministic temperature-0 pass@1.

Corollary 5 (Finite-sample sign reliability)**.** Consider two traces R_a, R_b for the same problem with population reward gap

$$\Delta = \mathbb{E}[\widehat{\mathcal{R}}(P, R_a) - \widehat{\mathcal{R}}(P, R_b) \mid P, R_a, R_b] > 0. \tag{81}$$

Let the empirical gap be $\widehat{\Delta}$. If the two reward estimates are independent, then

$$\mathbb{P}[\widehat{\Delta} \le 0] \le \frac{\mathrm{Var}[\widehat{\mathcal{R}}(P, R_a)] + \mathrm{Var}[\widehat{\mathcal{R}}(P, R_b)]}{\Delta^2}. \tag{82}$$

Proof. Let

$$\varepsilon = \widehat{\Delta} - \Delta. \tag{83}$$

Then $\mathbb{E}[\varepsilon] = 0$. If $\widehat{\Delta} \le 0$, then

$$\varepsilon \le -\Delta, \tag{84}$$

which implies

$$|\varepsilon| \ge \Delta. \tag{85}$$

Therefore, by Chebyshev's inequality,

$$\mathbb{P}[\widehat{\Delta} \le 0] \le \mathbb{P}[|\varepsilon| \ge \Delta] \le \frac{\mathrm{Var}[\varepsilon]}{\Delta^2}. \tag{86}$$

Under independence of the two reward estimates,

$$\mathrm{Var}[\varepsilon] = \mathrm{Var}[\widehat{\mathcal{R}}(P, R_a)] + \mathrm{Var}[\widehat{\mathcal{R}}(P, R_b)]. \tag{87}$$

Substituting gives the result. □

This corollary is not a convergence theorem for GRPO. It only states that larger reward margins and lower reward variance make pairwise credit assignment more reliable. The product term helps by increasing the margin between traces whose executor utility is supported by high reasoning quality and traces whose utility is weak, noisy, or unsupported.

G.8 Robustness to RM Score Error

The theory above does not require the RM to be perfect. Since the RM score is bounded and multiplied by uplift, the effect of RM score error is also bounded.

Let $m_\phi(P, R)$ be an ideal bounded score used for analysis and let $\widetilde{m}(P, R)$ be the score actually used in the reward. Suppose

$$|\widetilde{m}(P, R) - m_\phi(P, R)| \le \eta \tag{88}$$

for a given trace.

Proposition 7 (Bounded effect of RM score error)**.** For any fixed P, R,

$$\left|\mathbb{E}[\widetilde{\mathcal{R}}(P, R) - \mathcal{R}(P, R) \mid P, R]\right| \le \frac{\eta}{2}, \tag{89}$$

where $\widetilde{\mathcal{R}}$ denotes the reward computed with $\widetilde{m}$ and $\mathcal{R}$ denotes the reward computed with m_ϕ.

Proof. The only changed term is the RM-uplift term:

$$\mathbb{E}[\widetilde{\mathcal{R}} - \mathcal{R} \mid P, R] = 0.5\,(\widetilde{m}(P, R) - m_\phi(P, R))\,u_{\mathrm{tr}}(P, R). \tag{90}$$

Taking absolute values,

$$\left|\mathbb{E}[\widetilde{\mathcal{R}} - \mathcal{R} \mid P, R]\right| = 0.5\,|\widetilde{m}(P, R) - m_\phi(P, R)|\,|u_{\mathrm{tr}}(P, R)|. \tag{91}$$

Since $|\widetilde{m} - m_\phi| \le \eta$ and $|u_{\mathrm{tr}}(P, R)| \le 1$,

$$\left|\mathbb{E}[\widetilde{\mathcal{R}} - \mathcal{R} \mid P, R]\right| \le 0.5\,\eta. \tag{92}$$

□

For pairwise comparisons, if two traces each have RM score error at most η, then the error in the expected reward gap is at most η, because each trace contributes at most $\eta/2$. Therefore, a population reward margin larger than η is stable to such bounded RM score perturbations. This motivates training a calibrated Reason RM: the smaller and less saturated the score error, the more reliably the reward preserves the intended within-group ordering.

G.9 Scope of the Analysis

The analysis establishes properties of the training reward under a fixed executor and matched training-time sampling. It does not prove that every improvement in the training-time objective must transfer to deterministic evaluation. Such a statement would require additional assumptions relating $q_{\mathrm{tr}}(P, R)$ at temperature 0.5 to $q_{\mathrm{eval}}(P, R)$ at temperature 0, as well as assumptions about policy optimization and generalization. The main experiments provide this empirical link: the Reason RM and uplift estimator are removed at evaluation time, and improvements are measured only through the fixed planner-executor chain.

Within its stated scope, the analysis gives the following conclusions:

1. Executor uplift is an unbiased estimate of the conditional effect of supplying trace R to the frozen executor.
2. The expected TraceLift reward reduces, up to a problem-level constant, to a quality-weighted executor-uplift objective.
3. The product term prevents high RM scores from creating trace-specific reward when the trace has zero executor utility.
4. The verifier term provides a direct task-success anchor and prevents the reward from depending only on RM-weighted relative uplift.
5. A saturated LLM-as-judge score collapses to an execution-only ranking and therefore cannot substitute for a calibrated Reason RM.

H Additional Evaluation

H.1 Constant-RM Control

We compare Exec-only, Constant-RM, and TraceLift on HumanEval+ using the Qwen2.5-7B planner and the same frozen executor. Constant-RM sets the RM score to one, giving $r = 0.5R_{\text{exec}} + 0.5\hat{u}_{\text{exec}}$, while preserving TraceLift's $K = 3$ uplift sampling and sample-reuse rules. Exec-only uses R_{exec}; TraceLift uses its full reward. All variants use the same initialization, training data, optimizer settings, and 600 training steps. We evaluate the final checkpoints with greedy decoding and report mean pass@1 over three matched training seeds. This comparison tests whether learned RM weighting adds value beyond the matched executor sampling used by Constant-RM.

Table 22: HumanEval+ pass@1 (%).

Constant-RM	Exec-only	TraceLift
61.32	61.59	64.02

H.2 Human Preference Evaluation

We randomly select 20 HumanEval+ problems before inspecting model outputs and compare one greedy reasoning trace from Constant-RM with one from TraceLift for each problem, using a training seed fixed in advance. Three experts independently review the problem and both traces, with method names hidden and presentation order randomized. They prefer the trace with more correct, coherent, and executor-actionable reasoning, allowing ties; executor outputs and RM scores are hidden. Across the 60 judgments, TraceLift's tie-adjusted preference win rate is $100(W + 0.5T)/60$, where W and T count TraceLift preferences and ties, respectively. With 45 preferences for TraceLift, 9 ties, and 6 preferences for Constant-RM, the resulting win rate is **82.5%**. This measures human preference on the sampled traces, not reasoning faithfulness.