LEARNING AFTER EXECUTION WITH ACTION-SPECIFIC CREDIT

Anonymous authors
Paper under double-blind review

ABSTRACT

Execution is often used as terminal feedback: it scores a generated program, but the resulting state is not used for another learning action. Learning after execution instead uses the execution result as an intermediate state that supports an additional learning interaction for the same policy. This creates a credit-assignment problem because the two actions share the same final objective but play different roles in the trajectory. Accordingly, action-specific credit assigns the initial action credit from the final trajectory outcome and the corrective action credit for improvement over the intermediate state. Image-to-TikZ serves as an executable visual testbed for this setting. On 945 held-out examples, learning after execution strengthens the shared policy under single-call inference and also produces a post-execution corrective capability. Single-call render success reaches 92.70%, compared with 88.25% for continued single-stage GRPO after the same number of additional training iterations. This corrective capability is not reproduced by continued single-stage training and depends on the intermediate state. Alternative credit definitions also induce different group-relative preferences over the same trajectories. Together, these results suggest that executable tasks contain useful learning structure beyond a terminal reward.

1 INTRODUCTION

Executing a generated program exposes the result of the model's prediction. This result can be scored to train the policy, but it also provides a state on which the policy can act again (Zheng et al., 2024; Li et al., 2025; Bi et al., 2024). In a single-stage interaction, execution marks the end of the trajectory. The executed outcome provides a training signal, but the state produced by execution is not used for another learning action. Rather than ending the interaction at this point, we ask whether the execution state can support an additional learning interaction, and whether learning from this interaction strengthens the shared policy itself.

This setting introduces a credit-assignment problem because the two actions share the same final objective but serve different roles in the trajectory (Qu et al., 2026; Du et al., 2026). The initial action generates the intermediate output and determines the state from which the corrective action begins, while the corrective action inherits that state and seeks to improve it. The final trajectory outcome is relevant to both actions, but the corrective action must also be evaluated relative to the state it receives. Training should therefore account for both where the trajectory ends and what changes after execution.

We study learning after execution by using the execution state of the policy's own prediction to create an additional learning interaction from the same training target. For each target, the shared policy first generates and executes an initial program, then revises or repairs it from the resulting state. We assign action-specific credit to the two actions. The initial action receives credit from the final trajectory outcome, while the corrective action receives credit for improvement over the intermediate result. Both actions jointly update the same policy.

We use Image-to-TikZ as an executable visual testbed for this problem (Belouadi et al., 2024b; Saito et al., 2025). Image-to-TikZ reconstructs a target image as an executable TikZ program and belongs to the broader family of image-to-code (Zhou et al., 2026; Yang et al., 2025; Wu et al., 2025; Si et al., 2025; Roberts et al., 2024). A generated program can be executed to reveal either a rendered image or an execution failure. A valid render makes the visual difference from the target observable, while

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

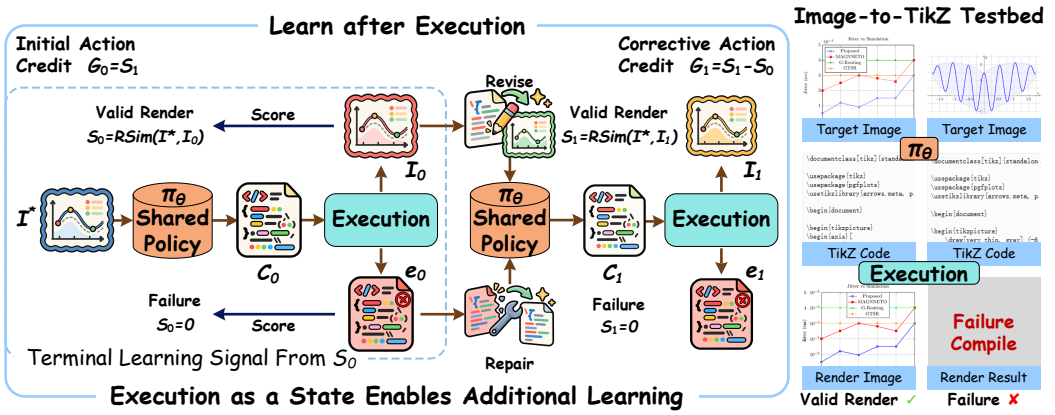


Figure 1: Overview of learning after execution with action-specific credit. Execution first provides a score for the initial action; learning after execution additionally uses the resulting render or failure state for a corrective action by the same policy. The initial action receives final-outcome credit, while the corrective action receives improvement-relative credit. The right panel illustrates the Image-to-TikZ instantiation with valid and failed executions.

a failed execution identifies a program that still requires repair. The result before and after correction can therefore be evaluated under the same execution and visual criteria. This makes Image-to-TikZ suitable for studying both how a policy learns from an execution state and what changes when it acts on that state.

Using Image-to-TikZ, we study learning after execution through a series of controlled comparisons. We separate improvement of the shared policy from the effect of invoking a corrective action at inference time, and distinguish learned post-execution correction from behavior that may arise from continued single-stage training alone. We further examine how alternative credit definitions shape the group-relative learning signals assigned to the two actions. Together, these analyses characterize whether learning after execution strengthens the shared policy, produces a distinct corrective capability, and how its learning is shaped by action-specific credit. Our main contributions are:

- We introduce learning after execution as a training setting in which execution is used not only to score a prediction, but also to produce an intermediate state that provides additional learning experience for the same policy.
- We formulate action-specific credit for this setting, preserving a shared final-quality objective while matching credit to the different roles of initial generation and correction.
- We show that learning from post-execution interaction strengthens the shared policy even under single-call inference, while also producing a corrective capability that is not reproduced by continued single-stage training alone.

2 Preliminaries: Code Supervision and Execution Feedback

Before introducing learning after execution, we define the program generation, execution score, and single-stage training setting used throughout the paper. In our Image-to-TikZ instantiation, a target image I^* is mapped to an executable TikZ program C. Executing C either produces a rendered image or results in execution failure. These outcomes define the execution score used by Direct-GRPO and by our method in Section 3.1.

2.1 Supervised Image-to-TikZ

Given paired training data $\mathcal{D} = \{(I_i^*, C_i^*)\}_{i=1}^N$, where I_i^* is a target image and C_i^* is its reference TikZ program, supervised fine-tuning trains the model to generate C conditioned on I_i^*. The supervision is defined on the reference program rather than on the executed outcome of a program sampled from the policy. Different TikZ programs can produce similar rendered figures, while small

code changes can alter the visual result or break execution. The rendered consequence of the model's own prediction therefore does not directly participate in the supervised training objective.

2.2 Direct Reinforcement Learning with Execution Feedback

Execution exposes the consequence of a generated program. For each target image, we sample a group of K programs $C_i \sim \pi_{\theta_{\text{old}}}(\cdot \mid I^*)$, for $i = 1, \ldots, K$, and evaluate their executed outputs. We define the execution score

$$S(C; I^*) = \begin{cases} \text{RSim}(\mathcal{E}(C), I^*), & \text{if } C \text{ produces a valid render,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

$\mathcal{E}$ denotes the LaTeX execution and rasterization pipeline. Following TikZilla, we use RSim as the visual similarity reward, which compares patch-level representations from the DeTikZify-V2 image encoder using Earth Mover's Distance (Kusner et al., 2015; Belouadi et al., 2024b; Greisinger & Eger, 2026). The score combines executability and visual fidelity: failed executions receive zero, while valid renders are scored by their similarity to the target.

For Direct-GRPO, each sampled program receives $S_i = S(C_i; I^*)$. Within each target-image group, we mean-center these scores to obtain group-relative advantages and optimize them with the clipped GRPO objective (Shao et al., 2024). Implementation details are given in Appendix B. We refer to this single-stage setting as Direct-GRPO.

Direct-GRPO learns from the executed outcomes of programs sampled from the current policy. Its optimization signal reflects both executability and rendered visual fidelity. Execution, however, still marks the end of the interaction: the result is used to score the sampled program, but the policy does not observe the rendered output or failure state as input for another action. The next section uses this execution result as an intermediate state on which the same policy can act again.

3 Learning after Execution with Action-Specific Credit

Direct-GRPO uses execution as terminal feedback. We instead use the execution result as an intermediate state on which the same policy can act again. The policy first generates and executes a program, then revises or repairs it based on the observed execution result. The initial and corrective actions are jointly trained with action-specific credit.

3.1 Execution-Conditioned Correction

Let $I^\star$ denote the target image. At training iteration k, the same policy π_{θ_k} is called twice within each trajectory. The first call directly generates an initial TikZ program,

$$C_{0,i} \sim \pi_{\theta_k}(\cdot \mid I^\star), \tag{2}$$

where i indexes one sampled trajectory. We execute $C_{0,i}$ using the same LaTeX rendering pipeline introduced in Section 2.2 and compute

$$S_{t,i} = S(C_{t,i}; I^\star), \qquad t \in \{0, 1\}, \tag{3}$$

using the execution score defined in Section 2.2. Thus, $S_{0,i}$ evaluates the intermediate program produced by the first call. The score is used for credit computation; it is not itself the observation provided to the second call.

The execution result determines the context for the corrective action. If $C_{0,i}$ produces a valid rendered image $I_{0,i}$, the model receives the target image, the current render, and the current TikZ program. It can therefore compare the target with what its own program actually produced. If no valid render is produced, the model instead receives the target image, the current TikZ attempt, and a compact execution status $e_{0,i}$. The corrective action is sampled as

$$C_{1,i} \sim \begin{cases} \pi_{\theta_k}\left(\cdot \mid I^\star, I_{0,i}, C_{0,i}\right), & \text{valid Stage 0 render,} \\ \pi_{\theta_k}\left(\cdot \mid I^\star, C_{0,i}, e_{0,i}\right), & \text{otherwise.} \end{cases} \tag{4}$$

The first branch performs visual revision. The second branch repairs, completes, or replaces an unsuccessful program. We then execute $C_{1,i}$ and obtain the final score $S_{1,i}$ using Eq. 3.

Stage 0 and Stage 1 are two calls to the same policy π_{θ_k} under different contexts. The first call observes only the target image. The second call observes the execution state produced by the first action. No parameter update occurs between the two calls. Both actions are sampled before the shared policy is updated.

3.2 Action-Specific Credit from Incremental Rewards

Both actions serve the same final-quality objective, $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[S_1]$. With two actions and no discounting, we decompose the execution score into incremental rewards and use the corresponding returns-to-go as action-specific credit:

$$\begin{aligned} r_{0,i} &= S_{0,i}, & r_{1,i} &= S_{1,i} - S_{0,i}, \\ G_{0,i} &= r_{0,i} + r_{1,i} = S_{1,i}, & G_{1,i} &= r_{1,i} = S_{1,i} - S_{0,i}. \end{aligned} \tag{5}$$

The incremental rewards telescope to the final score, $r_{0,i} + r_{1,i} = S_{1,i}$. The initial action therefore receives the return of the complete trajectory, while the corrective action receives the improvement relative to the intermediate state it inherits.

For the initial action, assigning only $S_{1,i} - S_{0,i}$ would omit its immediate reward $S_{0,i}$ and, for a fixed final score, favor a lower intermediate score. We therefore use the complete return $G_{0,i} = S_{1,i}$ and evaluate the initial action by the final quality reached by the trajectory.

For the corrective action, $G_{1,i} = S_{1,i} - S_{0,i}$ directly measures the score change from the state it receives. Its raw credit is positive when the execution score increases and negative when the score decreases. Because unsuccessful executions receive zero, the same definition also covers execution repair and failure: if $S_{0,i} = 0$, then $G_{1,i} = S_{1,i} \geq 0$; if $S_{1,i} = 0$, then $G_{1,i} = -S_{0,i} \leq 0$. Visual revision and execution repair are therefore handled by the same credit definition.

3.3 Group-Relative Optimization

For each target image, we sample K complete trajectories, with $K = 8$ in all experiments using the corrective action. Each trajectory $\tau_i = (C_{0,i}, S_{0,i}, C_{1,i}, S_{1,i})$ contains an independently sampled initial action followed by a corrective action conditioned on its own execution state. Thus, a group contains K independently sampled trajectories rather than one shared Stage 0 program followed by multiple Stage 1 candidates.

The action-specific returns in Eq. 5 are centered separately within the same target-image group. With the implementation scale $\lambda = 10$,

$$A_{t,i} = \lambda \left(G_{t,i} - \frac{1}{K} \sum_{j=1}^{K} G_{t,j} \right), \qquad t \in \{0, 1\}. \tag{6}$$

We subtract only the group mean and do not normalize by the group standard deviation. $A_{0,i}$ is assigned to the completion tokens of $C_{0,i}$, while $A_{1,i}$ is assigned to those of $C_{1,i}$.

The distinction between raw credit and group-relative advantage is important. A positive $G_{1,i}$ means that the corrective action increases the execution score, but it does not necessarily imply $A_{1,i} > 0$; the latter also depends on the mean improvement within the sampled group.

For the corrective action, $S_{0,i}$ is determined before $C_{1,i}$ is sampled and acts as a state-dependent baseline. Appendix E further compares improvement-relative and absolute final-score credit.

The group-relative objective is applied separately to the two action sets,

$$\mathcal{L} = 0.5\mathcal{L}_{\text{GRPO}}(C_0, A_0) + 0.5\mathcal{L}_{\text{GRPO}}(C_1, A_1). \tag{7}$$

Both terms optimize the same policy parameters. All Stage 0 and Stage 1 actions in a training iteration are sampled from the same policy π_{θ_k} before any parameter update. The corresponding losses for the two stages are optimized sequentially over action microbatches using the shared policy.

4 EXPERIMENTS

The experiments examine how using execution during training affects both initial generation and subsequent correction. We first evaluate overall performance under direct and post-execution inference. We then study the corrective action, the role of action-specific credit, and how correction varies across executed intermediate states.

4.1 EXPERIMENTAL SETUP

Models and Training Data. Qwen3-VL-8B-Instruct (Bai et al., 2025) is used as the base model throughout the experiments. All supervised and reinforcement learning data are drawn from DaTikZ-V4 (Greisinger & Eger, 2026). Direct-SFT is trained on approximately 4K Image-to-TikZ pairs. Direct-GRPO is initialized from the final SFT checkpoint and trained on 1K samples for 125 training iterations. Our method is initialized from the final Direct-GRPO checkpoint and trained on the same 1K samples for 125 training iterations.

Evaluation Data. A temporally held-out evaluation set is constructed using the same source collection procedure as DaTikZ-V4, restricted to data available from 2026 onward. Samples are collected from arXiv, GitHub, and TeX StackExchange. The 1,095 examples are divided into two non-overlapping subsets. **Dev-150** contains 150 examples for checkpoint selection, ablations, and behavioral analysis. **Final-945** contains the 945 examples and is reserved for final evaluation.

Evaluation Metrics. Render rate measures whether the generated TikZ program executes successfully. Visual fidelity is evaluated with RSim (Greisinger & Eger, 2026), DINO (Caron et al., 2021), DreamSim (Fu et al., 2023), and CLIPImg (Radford et al., 2021). RSim is also used as the reinforcement learning reward. Unless stated otherwise, visual metrics use the *fixed-zero* convention, where failed executions receive a score of zero.

Inference Protocol. All models are evaluated with temperature 1.0, top-p 0.9, repetition penalty 1.0, and at most 8192 generated tokens. Generation stops at the first complete `\end{document}`. Stage 0 denotes the initial output produced from the target image alone, while Stage 1 denotes the raw output produced after observing the Stage 0 execution state. We additionally evaluate an execution-aware selection rule over the two outputs already produced. If one stage fails to render, the valid output is retained. If both render, the output with the higher RSim score is selected, with ties assigned to Stage 0. No additional model generation is required by the selection rule.

4.2 OVERALL RESULTS

We first compare the main models on Final-945. Our initializer is the final Direct-GRPO checkpoint, evaluated with the same Stage 0/Stage 1 interaction before further training. Because decoding is stochastic, its Stage 0 result and the separately sampled Direct-GRPO baseline can differ slightly. As an additional control, we continue Direct-GRPO from the same checkpoint on the same training set for another 100 training iterations.

Table 1 shows that SFT does not improve performance on this held-out set, whereas Direct-GRPO reaches a render rate of 81.90%, together with consistent improvements across all four visual metrics. Optimizing executed outcomes provides a substantially stronger signal than reference code supervision alone.

Continuing Direct-GRPO for another 100 training iterations provides further gains, raising render success to 88.25%. Starting from the same Direct-GRPO checkpoint and after the same number

Table 1: Main results on Final-945. Visual metrics use the fixed-zero convention. *Direct* denotes single-call generation. *Stage 0* is the initial single-call output of our policy, while *Stage 1* is produced after execution-conditioned correction. *Selected* chooses between the two outputs using execution success and RSim. Direct-GRPO (+100) continues single-stage GRPO for 100 additional training iterations from the same Direct-GRPO checkpoint. Best results are shown in bold and second-best results are underlined.

Method	Output	Render Success ↑	RSim ↑	DINO ↑	DreamSim ↑	CLIPImg ↑
Base model	Direct	71.64%	0.4033	0.6025	0.5545	0.6435
SFT	Direct	64.66%	0.3554	0.5427	0.4960	0.5781
Direct-GRPO	Direct	81.90%	0.4709	0.7046	0.6468	0.7410
Direct-GRPO (+100)	Direct	88.25%	0.5135	0.7629	0.7019	0.7996
Ours (initializer)	Stage 0	83.07%	0.4766	0.7129	0.6544	0.7489
Ours (initializer)	Stage 1	83.17%	0.4781	0.7142	0.6556	0.7501
Ours	Stage 0	92.70%	0.5382	0.8014	0.7330	0.8372
Ours	Stage 1	<u>92.80%</u>	<u>0.5446</u>	<u>0.8046</u>	<u>0.7401</u>	<u>0.8409</u>
Ours	Selected	**94.18%**	**0.5649**	**0.8221**	**0.7593**	**0.8583**

of additional training iterations, our Stage 0 output reaches 92.70% and is higher under all four visual metrics. Continued single-stage training therefore does not reproduce the gain obtained with our training. Importantly, this gain is already present in Stage 0, which uses the same single-call generation setting as the direct baselines.

Simply adding a second execution-conditioned call is insufficient. At initialization, Stage 0 and Stage 1 differ little. After training, raw Stage 1 reaches 92.80% render success and achieves slightly higher visual similarity than Stage 0 across all four metrics. Execution-aware selection further raises render success to 94.18% and improves all four visual metrics.

These results separate two effects. Training strengthens the shared policy itself, as shown by the improved Stage 0 output, while the second action provides an additional opportunity to act on the execution state. We examine the latter behavior next.

4.3 Learning Dynamics of Generation and Correction

Since Stage 0 already improves substantially, we isolate the corrective action itself. For visual correction, we consider only examples where both stages render successfully, termed the *paired success* subset. For a visual similarity metric M, we define

$$\Delta M = M(I_1, I^\star) - M(I_0, I^\star), \tag{8}$$

where I_0 and I_1 are the rendered outputs before and after correction, and $I^\star$ is the target image. This comparison excludes gains from execution recovery and isolates the visual change produced by Stage 1 on already executable outputs. For the correction-outcome composition, we use a small neutral region: $\Delta\mathrm{RSim} > 0.005$ is counted as improvement, $|\Delta\mathrm{RSim}| \leq 0.005$ as neutral, and $\Delta\mathrm{RSim} < -0.005$ as degradation.

Figure 2 shows that the initializer does not yet act as a reliable visual corrector. On paired-success examples, its average Stage 1 change is negative under all four visual metrics. Strong direct generation therefore does not automatically provide the ability to revise an executed result. Further single-stage optimization shows the same pattern: when Continued Direct-GRPO is evaluated with the same execution-conditioned second call, its paired visual changes remain close to zero.

This behavior changes during training. At step 75, the average Stage 1 change is positive under RSim, DINO, DreamSim, and CLIPImg. RSim is used for training. The concurrent gains under the other three metrics show that the visual improvement is not confined to the reward metric. Correction quality is not monotonic across training, even though render rate improves much more consistently.

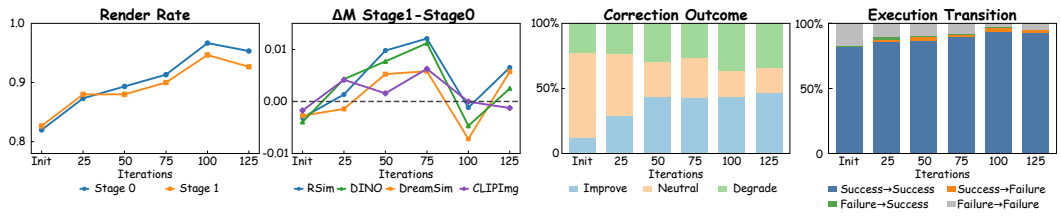


Figure 2: Learning dynamics of our method on Dev-150. The figure tracks Stage 0 and Stage 1 render rates, paired visual changes ΔM, the distribution of correction outcomes, and execution transitions across training checkpoints.

This shows that initial generation quality and the effect of the corrective action can evolve differently. We return to this difference in Section 4.5.

The correction-outcome distribution also changes substantially. The fraction of paired-success examples improved by Stage 1 rises from 12.2% for the initializer to 42.5% at step 75 and 46.8% at step 125. Most successful Stage 0 executions remain executable after Stage 1, while failed-to-successful recovery remains relatively rare on Dev-150. Together, these results show that training changes both the initial action and the behavior of the subsequent corrective action. Checkpoint 100 is selected on Dev-150 for subsequent behavioral analyses and final evaluation on Final-945, while training dynamics are examined across the full checkpoint trajectory.

4.4 Credit Assignment in Post-Execution Learning

We factorize the action-specific credit design into a 2×2 ablation. Stage 0 credit is either the final trajectory score S_1 or the local intermediate score S_0, while Stage 1 credit is either the improvement $S_1 - S_0$ or the absolute final score S_1. This yields the four variants in Table 2, with A corresponding to our method.

Table 2: Factorized ablation of action-specific credit assignment. Rows vary Stage 0 credit and columns vary Stage 1 credit.

	Stage 1 credit	
Stage 0 credit	$\lambda(S_1 - S_0)$	λS_1
λS_1	**A (Ours)**	C
λS_0	B	D

Downstream versus local credit for the initial action. The A–B comparison isolates Stage 0 credit. At step 50, B already achieves a slightly higher Stage 0 render rate than A, yet its subsequent Stage 1 correction is nearly neutral. A produces a substantially larger correction at the same checkpoint and remains positive at step 75, whereas B becomes slightly negative. Thus, optimizing the intermediate output itself does not produce the same subsequent correction behavior. Using downstream credit S_1 instead favors initial actions according to the quality reached after the corrective action.

Improvement-relative versus absolute credit for the corrective action. The A–C comparison isolates Stage 1 credit. We recompute absolute S_1-based Stage 1 advantages on the 8,000 trajectories collected from the completed A run. The difference is substantial at both trajectory and group levels: replacing $S_1 - S_0$ with S_1 flips the sign of the group-relative advantage for 37.2% of trajectories and changes the highest-advantage trajectory in 74.5% of target-image groups. The two credit definitions therefore induce substantially different group-relative preferences over corrective actions.

Optimization behavior of the credit variants. Variants C and D share absolute Stage 1 credit and enter the same failure regime from step 26 despite using different Stage 0 credits. Their Stage 1 generations become substantially longer and normal document termination deteriorates, causing most outputs to fail execution. For D, the average Stage 1 completion approaches 7K tokens and 86% of completions reach the generation limit.

Once both stage scores collapse to zero within a group, reward variation disappears and the group-relative advantages, policy loss, and gradient norm become zero. The zero learning signal therefore follows the generation failure rather than preceding it. In contrast, A and B remain executable and

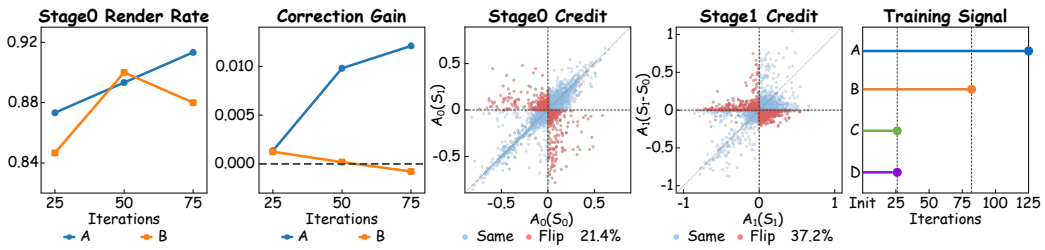


Figure 3: Effect of action-specific credit assignment. The figure compares Stage 0 quality and correction gain under alternative credit definitions, measures how the resulting group-relative preferences differ, and tracks the training signal of all four ablation variants.

retain nonzero training signals on the same groups. C and D differ in Stage 0 credit but share the same absolute Stage 1 credit, making this the common factor associated with their early failure. Together, the fixed-trajectory preference analysis and observed optimization behavior show that S_1 and $S_1 - S_0$ are not interchangeable credit signals for the corrective action.

4.5 Correction Outcomes Across Intermediate States

The previous analysis shows that the effect of the corrective action is not constant across training. We therefore examine how Stage 1 outcomes vary with the quality of the executed Stage 0 state. Using the Dev-selected checkpoint at step 100, Figure 4 shows a clear pattern across intermediate-state quality. Stage 1 provides the largest average gain when the intermediate render is weak. For Stage 0 RSim below 0.4, it improves over Stage 0 in 76% of paired examples. As Stage 0 quality increases, both the average correction gain and the probability of improvement generally decrease. In the strongest group, the average change becomes negative and Stage 1 improves over Stage 0 in only 23% of cases. Additional qualitative examples covering successful revision, execution repair, over-revision, preservation, and unresolved cases are provided in Appendix A.

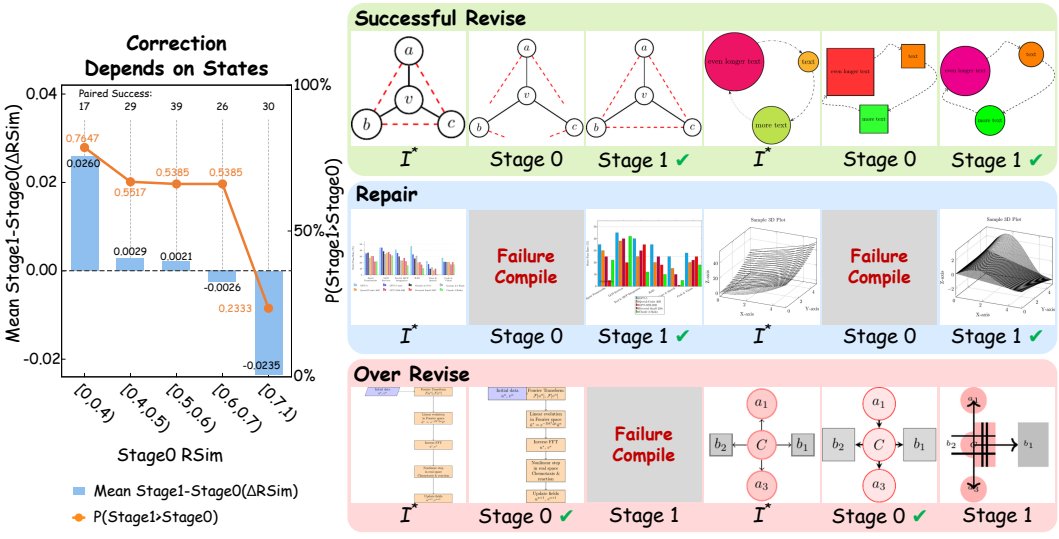


Figure 4: Correction behavior as a function of Stage 0 quality. We report the mean Stage 1 change in RSim and the frequency with which Stage 1 achieves higher RSim than Stage 0 across Stage 0 quality ranges, together with qualitative examples of beneficial and harmful corrections.

This pattern helps explain the non-monotonic correction curves in Figure 2. Weak intermediate renders leave larger discrepancies for Stage 1 to address. As Stage 0 becomes stronger, the remaining errors become smaller and further modification carries a greater risk of degrading an already good reconstruction. Thus, a stronger Stage 0 policy can reduce the average marginal value of correction even when Stage 1 remains capable of improving weak states.

Execution-Aware Selection. The asymmetric effect of correction motivates selecting between the two outputs already produced by the policy. We use the execution-aware rule defined in Section 4.1, which requires no additional model generation. On Dev-150, checkpoint 100 gives the strongest selected result. Selection raises render success to 97.3%, compared with 96.7% for Stage 0 and 94.7% for raw Stage 1, while RSim-FZ reaches 0.5646. DINO, DreamSim, and CLIPImg also improve although they are not used by the selector.

The checkpoint and selection rule are then frozen before evaluation on Final-945. As shown in Table 1, selection raises render success from 92.80% for raw Stage 1 to 94.18% and improves all four fixed-zero visual metrics. Stage 0 can therefore be used directly as the single-call output, while the post-execution action and selection provide an optional additional refinement mode.

5 Related Work

Rendering- and execution-aware reinforcement learning. RLRF (Rodríguez et al., 2025b) optimizes vector graphics with rewards from rendered SVG outputs, while TikZilla (Greisinger & Eger, 2026) applies GRPO to Text-to-TikZ using rendered visual similarity. In code synthesis, RLEF (Gehring et al., 2025) and related work (Le et al., 2022; Liu et al., 2023) use execution feedback to improve generated programs. These methods establish execution as an effective learning signal. We focus on the additional learning and credit-assignment problem that arises when the execution result becomes an intermediate state for another action.

Iterative correction and multi-turn policy learning. Iterative improvement has been explored through search, self-correction, and multi-turn reinforcement learning (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024; Yao et al., 2023; Chen et al., 2024). DeTikZify refines programs with MCTS, while Sketch2Diagram selects among independently generated candidates. Recent agentic RL methods study finer-grained credit across interaction steps, including GiGPO (Feng et al., 2025) and turn-level policy optimization (Ding et al., 2026). Our setting instead uses the execution state produced by the policy's own initial program as the context for a subsequent action, and assigns credit to the initial and corrective actions according to their different roles in the trajectory.

Graphics program synthesis. Recent work increasingly treats scientific figure reconstruction as graphics program synthesis rather than raster image generation (Belouadi et al., 2025; Rodríguez et al., 2025a; Belouadi et al., 2024a; Wu et al., 2023). DeTikZify (Belouadi et al., 2024b) generates TikZ programs from figures and sketches with inference-time refinement, while Sketch2Diagram (Saito et al., 2025) studies sketch-to-TikZ generation with data augmentation and multi-candidate inference. Omni-I2C (Zhou et al., 2026) further highlights the difficulty of high-fidelity image-to-code generation across diverse domains. We use Image-to-TikZ as an executable visual testbed for studying learning after execution.

6 Conclusion and Limitations

The study shows that execution can provide more than a terminal learning signal. It can expose an intermediate state from which additional learning becomes possible. Using the same training targets, shared policy, and execution score, our method turns this state into a second learning interaction and assigns credit according to the distinct roles of initial generation and correction. This simple change strengthens the policy even under single-call inference, while also producing an additional post-execution corrective capability. More broadly, our results suggest that executable tasks may contain useful learning structure beyond the scalar reward obtained at the end of generation, and that carefully assigning credit can expose this structure without requiring a more complex reward design. At the same time, our experiments provide an initial demonstration of this principle rather than a complete characterization of its scope. Our study is limited to one 8B backbone, one executable visual program domain, and a relatively small training set. Correction is not always beneficial for strong intermediate states, motivating learned stopping or routing, while the instability of absolute corrective credit remains specific to our current RSim-based setting.

AI USE STATEMENT

Generative AI tools were used during the development of this work to assist in research brainstorming, experimental design discussions, code development and debugging, literature search and summarization, figure preparation and drafting and editing parts of the manuscript. AI-assisted suggestions related to methodology, experiments, and interpretation were independently reviewed by the authors, and all reported experimental results were obtained from the pipelines implemented by the authors and verified against the corresponding source data and logs. AI-generated or AI-modified code used in the experiments was reviewed and tested by the authors. The authors assume full responsibility for the final content of the article, including all claims, results, code, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the complete training and evaluation protocol in Appendices A–F, including prompts, action-specific credit definitions, optimization details, data splits, checkpoint selection, decoding parameters, and metric conventions. An anonymized implementation, training configurations, and evaluation scripts are provided in the supplementary material.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *CoRR*, abs/2511.21631, 2025. doi: 10.48550/ARXIV.2511.21631. URL https://doi.org/10.48550/arXiv.2511.21631.

Jonas Belouadi, Anne Lauscher, and Steffen Eger. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=v3K5TVP8kZ.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b. URL http://papers.nips.cc/paper_files/paper/2024/hash/9a8d52eb05eb7b13f54b3d9eada667b7-Abstract-Conference.html.

Jonas Belouadi, Eddy Ilg, Margret Keuper, Hideki Tanaka, Masao Utiyama, Raj Dabre, Steffen Eger, and Simone Paolo Ponzetto. Tikzero: Zero-shot text-guided graphics program synthesis. In *IEEE/CVF International Conference on Computer Vision, ICCV 2025, Honolulu, HI, USA, October 19-25, 2025*, pp. 17793–17806. IEEE, 2025. doi: 10.1109/ICCV51701.2025.01653. URL https://doi.org/10.1109/ICCV51701.2025.01653.

Zhangqian Bi, Yao Wan, Zheng Wang, Hongyu Zhang, Batu Guan, Fangxin Lu, Zili Zhang, Yulei Sui, Hai Jin, and Xuanhua Shi. Iterative refinement of project-level code context for precise code generation with compiler feedback. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, volume ACL 2024 of *Findings of ACL*, pp. 2336–2353. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.138. URL https://doi.org/10.18653/v1/2024.findings-acl.138.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pp. 9630–9640. IEEE, 2021. doi: 10.1109/ICCV48922.2021.00951. URL https://doi.org/10.1109/ICCV48922.2021.00951.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=KuPixIqPiq.

Yifeng Ding, Hung Le, Songyang Han, Kangrui Ruan, Zhenghui Jin, Varun Kumar, Zijian Wang, and Anoop Deoras. Empowering multi-turn tool-integrated agentic reasoning with group turn policy optimization. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 42409–42423. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.1962. URL https://doi.org/10.18653/v1/2026.acl-long.1962.

Linxuan Du, Guangquan Xue, Xiaobo Liang, Qipeng Huang, Yuyang Ding, Xinyu Shi, Zhang Yijun, Ji Qi, Wenpeng Zhu, Juntao Li, and Min Zhang. Escaping the echo trap: On credit assignment failure in multi-turn LLM self-reflection. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 35393–35405. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.1636. URL https://doi.org/10.18653/v1/2026.acl-long.1636.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Stephanie Fu, Netanel Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. Dreamsim: Learning new dimensions of human visual similarity using synthetic data. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/9f09f316a3eaf59d9ced5ffaefe97e0f-Abstract-Conference.html.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: grounding code llms in execution feedback with reinforcement learning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/gehring25a.html.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

Christian Greisinger and Steffen Eger. Tikzilla: Scaling text-to-tikz with high-quality data and reinforcement learning. *CoRR*, abs/2603.03072, 2026. doi: 10.48550/ARXIV.2603.03072. URL https://doi.org/10.48550/arXiv.2603.03072.

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 957–966. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/kusnerb15.html.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu-Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/8636419dea1aa9fbd25fc4248e702da4-Abstract-Conference.html.

Qingyao Li, Xinyi Dai, Xiangyang Li, Weinan Zhang, Yasheng Wang, Ruiming Tang, and Yong Yu. Codeprm: Execution feedback-enhanced process reward model for code generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pp. 8169–8182. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-ACL.428. URL https://doi.org/10.18653/v1/2025.findings-acl.428.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. RLTF: reinforcement learning from unit test feedback. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=hjYmsV6nXZ.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Changle Qu, Sunhao Dai, Hengyi Cai, Jun Xu, Shuaiqiang Wang, and Dawei Yin. Matchtir: Fine-grained supervision for tool-integrated reasoning via bipartite matching. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 11953–11968. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.549. URL https://doi.org/10.18653/v1/2026.acl-long.549.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 2021. URL http://proceedings.mlr.press/v139/radford21a.html.

Josselin Somerville Roberts, Tony Lee, Chi Heem Wong, Michihiro Yasunaga, Yifan Mai, and Percy Liang. Image2struct: Benchmarking structure extraction for vision-language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/d0718553fd6b227a353c6432cf893285-Abstract-Datasets_and_Benchmarks_Track.html.

Juan A. Rodríguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Sai Rajeswar, David Vázquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images and text. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pp. 29691–29693. AAAI Press, 2025a. doi: 10.1609/AAAI.V39I28.35369. URL https://doi.org/10.1609/aaai.v39i28.35369.

Juan A. Rodríguez, Haotian Zhang, Abhay Puri, Rishav Pramanik, Aarash Feizi, Pascal Wichmann, Arnab Kumar Mondal, Mohammad Reza Samsami, Rabiul Awal, Perouz Taslakian, Spandana Gella, Sai Rajeswar Mudumba, David Vázquez, Chris Pal, and Marco Pedersoli. Rendering-aware reinforcement learning for vector graphics generation. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/57126328c3b40cf618a34f1c5df24d8a-Abstract-Conference.html.

Itsumi Saito, Haruto Yoshida, and Keisuke Sakaguchi. Sketch2diagram: Generating vector diagrams from hand-drawn sketches. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=KvaDHPhhir.

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. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 3956–3974. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.199. URL https://doi.org/10.18653/v1/2025.naacl-long.199.

Chengyue Wu, Zhixuan Liang, Yixiao Ge, Qiushan Guo, Zeyu Lu, Jiahao Wang, Ying Shan, and Ping Luo. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, volume NAACL 2025 of *Findings of ACL*, pp. 3006–3028. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-NAACL.164. URL https://doi.org/10.18653/v1/2025.findings-naacl.164.

Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. Iconshop: Text-guided vector icon synthesis with autoregressive transformers. *ACM Trans. Graph.*, 42(6):230:1–230:14, 2023. doi: 10.1145/3618364. URL https://doi.org/10.1145/3618364.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, Gongye Liu, Xiaomei Nie, Deng Cai, and Yujiu Yang. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=sGpCzsfd1K.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. Opencodeinterpreter: Integrating code generation with execution and refinement. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, volume ACL 2024 of *Findings of ACL*, pp. 12834–12859. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.762. URL https://doi.org/10.18653/v1/2024.findings-acl.762.

Jiawei Zhou, Chi Zhang, Xiang Feng, Qiming Zhang, Haibo Qiu, Lihuo He, Dengpan Ye, Xinbo Gao, and Jing Zhang. Omni-i2c: A holistic benchmark for high-fidelity image-to-code generation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 37761–37798. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.1753. URL https://doi.org/10.18653/v1/2026.acl-long.1753.

A Additional Qualitative Results and Failure Cases

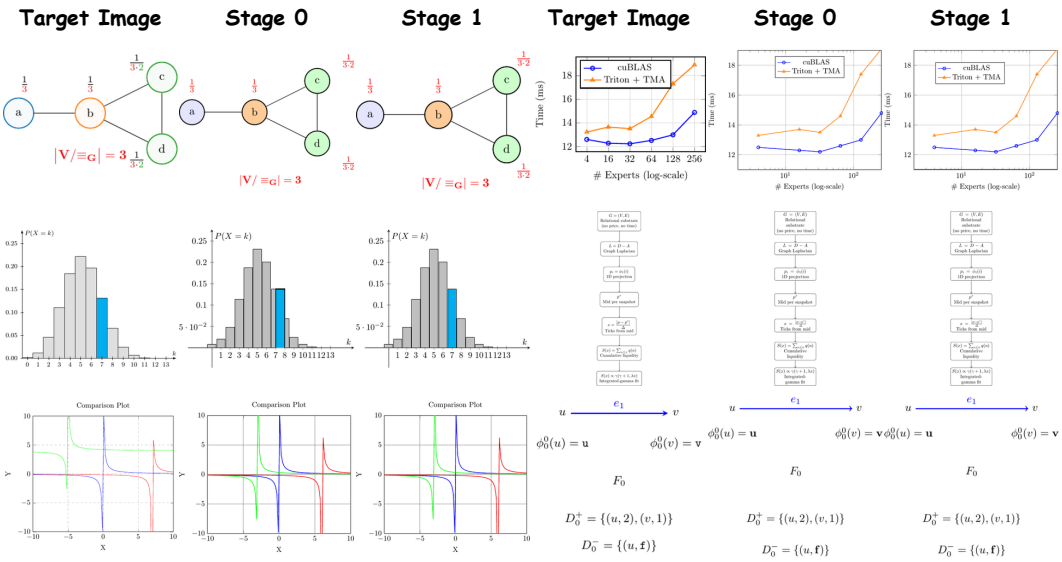


Figure 5: Additional qualitative examples of our method from Final-945.

This section provides additional qualitative examples from Final-945. All examples from our method use the checkpoint selected on Dev-150 at step 100 and the same inference protocol as the quantitative evaluation. The examples complement the quantitative analyses by illustrating changes across learning settings, successful visual revision, execution repair, over-revision, preservation, and unresolved cases.

A.1 Qualitative Progression across Learning Settings

Figure 6 compares outputs across the main learning settings studied in the paper. For each target image, we show predictions from the base model, Direct-SFT, Direct-GRPO, and Stage 0 and Stage 1 of our trained policy. The examples complement the aggregate results in Table 1. Execution-feedback training can substantially change the direct prediction relative to supervised code learning, while our Stage 0 output shows that the shared policy itself improves further. Stage 1 then provides an additional opportunity to act on discrepancies revealed by execution.

Stage 1 does not necessarily introduce a visible change. When Stage 0 already closely matches the target, the Stage 1 render can remain effectively unchanged. These cases are consistent with the state-dependent analysis in Appendix F.1, where the marginal value of correction decreases as the intermediate state becomes stronger.

A.2 Successful Visual Revision

Figure 7 shows cases in which Stage 0 already produces a valid program but its rendered result still differs visibly from the target. Stage 1 receives the target image, the Stage 0 render, and the corresponding TikZ program, and revises the program from this executed state.

The examples show that execution-conditioned correction is not restricted to small cosmetic edits. In some cases, Stage 1 substantially reorganizes an incorrect visual configuration, for example by changing geometric structure or restoring a missing layout component. In others, the correction is more localized, such as modifying the shape, placement, or appearance of a visual element.

The second action is conditioned on the specific program and render produced by Stage 0 rather than independently resampling from the target image. The examples therefore illustrate how the policy can use the observed execution result to modify its own previous program. The cross-metric

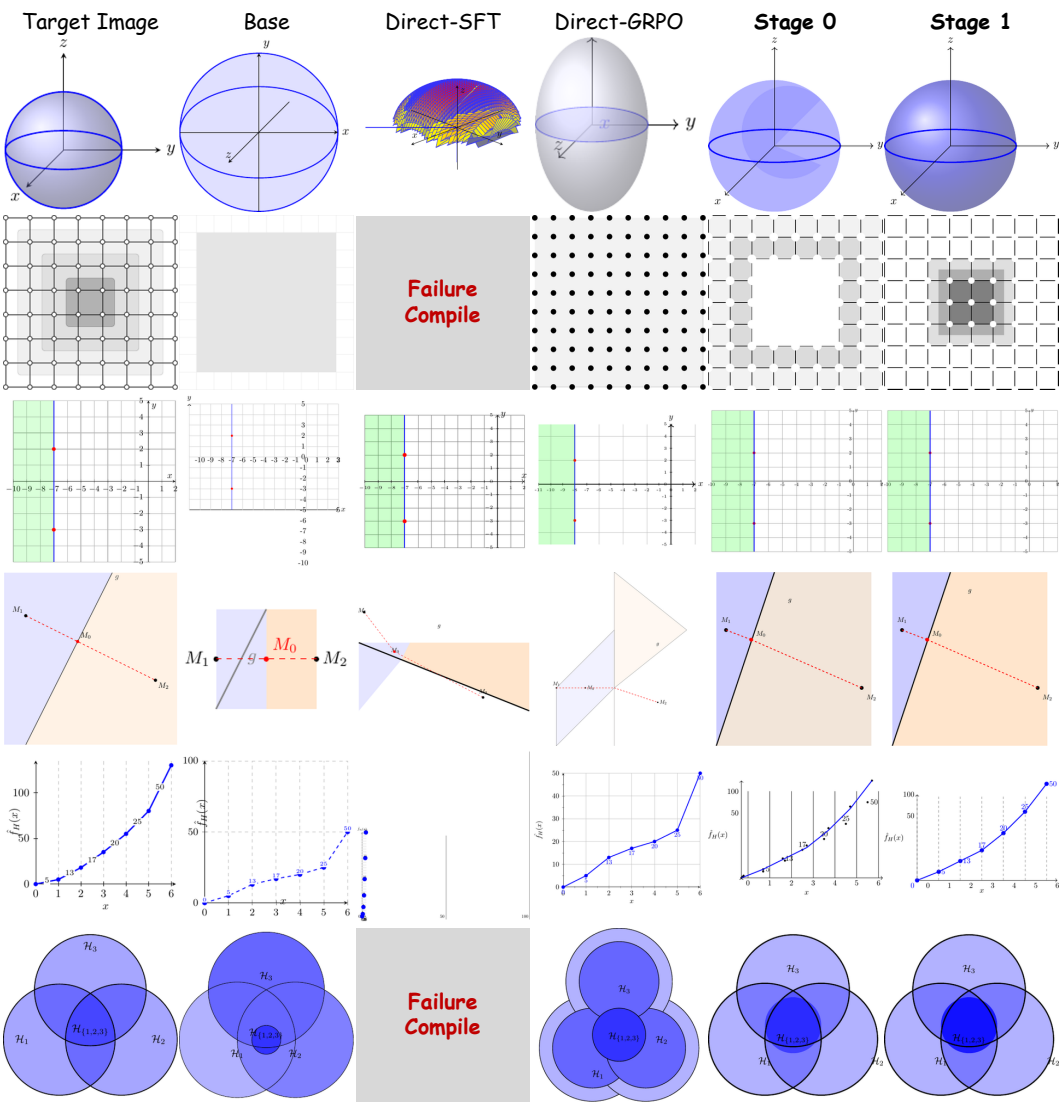


Figure 6: Qualitative progression across the main learning settings on Final-945. Base model, Direct-SFT, and Direct-GRPO show single-call predictions. Ours Stage 0 is also a single-call prediction from the trained policy, while Stage 1 is produced after observing the Stage 0 execution state. The examples include visible corrective changes and cases where an already strong Stage 0 result is largely preserved.

improvements reported in Section 4.3 and Appendix F.1 show that these changes are not confined to RSim.

A.3 Execution Repair

The corrective action is also used for execution repair. When Stage 0 fails, no rendered image is available. Stage 1 instead receives the target image, the unsuccessful TikZ program, and a compact execution status. Figure 8 shows examples in which this route turns an invalid Stage 0 attempt into a valid program.

The selected examples include complete Stage 0 generations that fail during compilation, after which Stage 1 modifies the existing program into an executable one. The corrective action therefore operates on an unsuccessful program state rather than simply appending missing tokens. Repair and visual revision receive different observations but use the same improvement-relative credit. For an

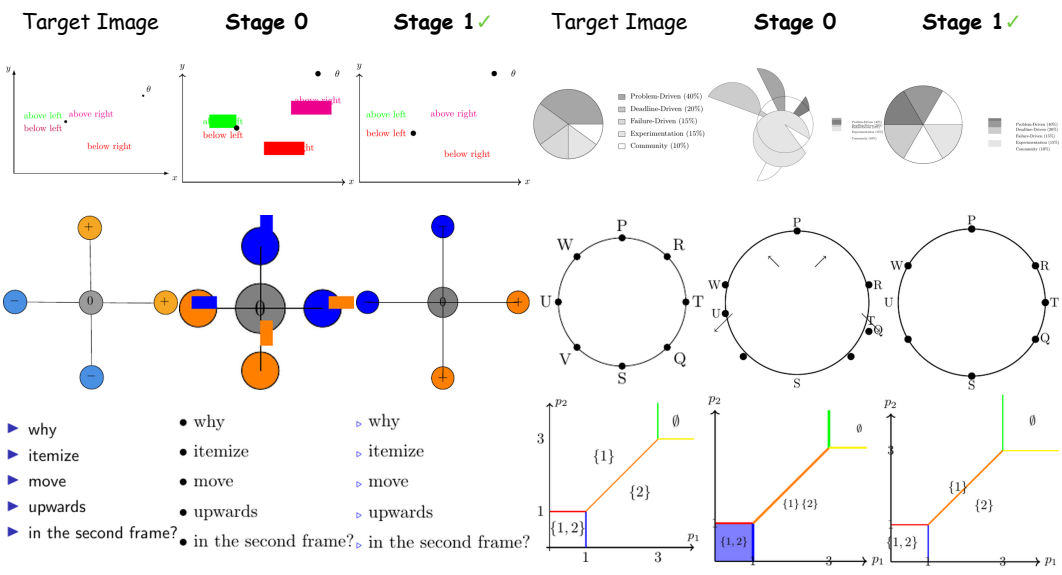


Figure 7: Successful visual revisions from Final-945 at checkpoint 100. Stage 0 already renders successfully, and Stage 1 revises the executed state to better match the target. The examples illustrate both large structural corrections and more localized changes to geometry, visual elements, or layout.

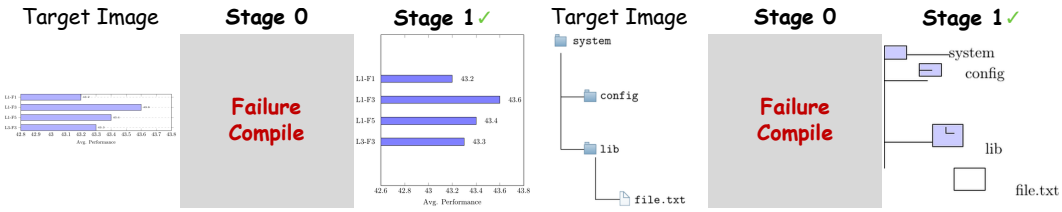


Figure 8: Successful execution repair from Final-945. Stage 0 does not produce a valid render and therefore receives zero execution score. Stage 1 conditions on the failed program and its execution status and produces a valid TikZ program. The illustrated repairs include cases in which Stage 0 has already produced a complete document that nevertheless fails compilation.

invalid Stage 0 execution, $S_0 = 0$, so a successful repair with positive visual similarity receives positive corrective credit. Repair trajectories are less frequent than visual-revision trajectories, as quantified in Appendix F.3, while their execution outcomes are analyzed in Appendix F.4.

A.4 Over-Revision of Strong Intermediate States

Correction is not always beneficial. Figure 9 shows examples in which Stage 0 already provides a strong reconstruction and Stage 1 modifies the program in a way that reduces visual fidelity.

Some Stage 1 outputs introduce visible changes that are unsupported by the target, such as additional graphical elements, distorted annotations, or large rendering artifacts. These examples provide a qualitative counterpart to the negative average correction observed for the strongest Stage 0 bucket in Appendix F.1. Together with the preservation cases in Figure 6, they illustrate the asymmetric value of the corrective action: substantial modification can help when the executed state contains clear discrepancies, while an already strong state may be better left largely unchanged.

When correction-enabled inference is used, execution-aware selection can retain Stage 0 when the subsequent correction is worse. The rule does not generate a new candidate; it only chooses between the two outputs already produced.

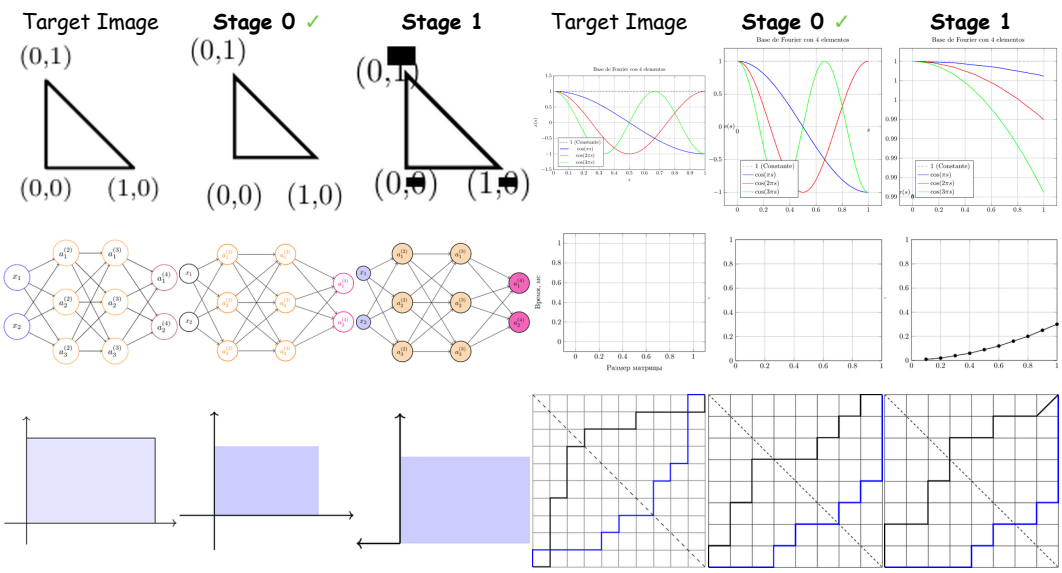


Figure 9: Examples of over-revision at checkpoint 100. Stage 0 already provides a useful reconstruction, while the Stage 1 modification reduces RSim or introduces visible artifacts. The execution-aware selection rule retains Stage 0 in these cases.

A.5 Preservation and Unresolved Cases

Stage 1 does not always substantially change the state produced by Stage 0. Figure 10 shows both useful preservation and cases where important errors remain unresolved. When Stage 0 already provides a strong reconstruction, preserving the existing result can avoid unnecessary modification. In other cases, however, Stage 0 still contains visible errors and Stage 1 changes little.

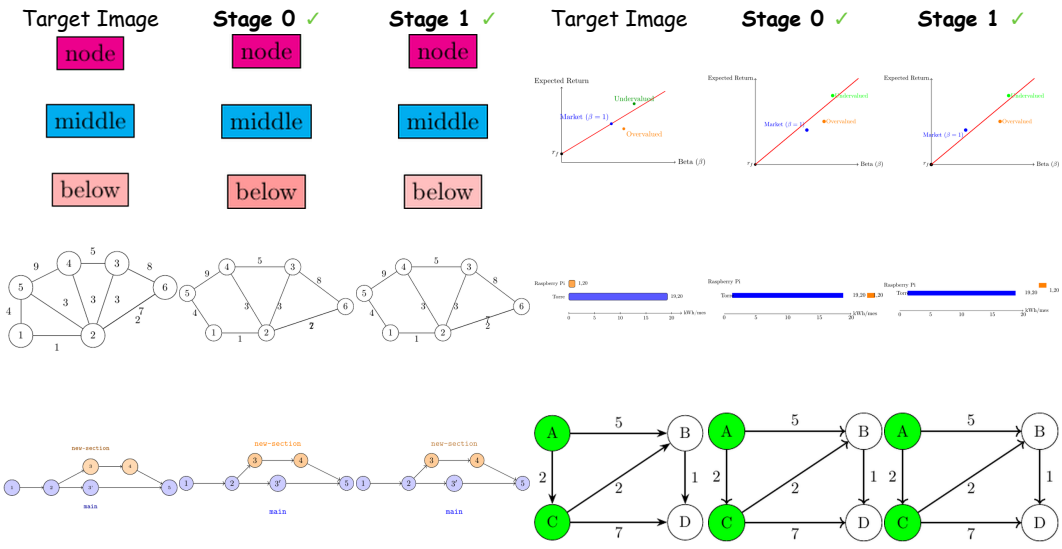


Figure 10: Examples of preservation and unresolved cases on Final-945. Stage 1 may preserve an already strong Stage 0 result, or leave an imperfect reconstruction largely unchanged.

Some unsuccessful Stage 0 programs also remain non-executable after the repair action. Together with the successful revisions, execution repairs, and over-revisions shown above, these examples illustrate the range of behaviors of the corrective action: it can improve, degrade, or largely preserve the execution state it receives.

B Implementation Details

This section provides implementation details omitted from the main text. We first give the complete optimization procedure, followed by the prompt templates and execution-dependent routing used for the two actions. We then summarize the training configuration and execution environment.

B.1 Full Optimization Algorithm

Algorithm 1 summarizes one training iteration of our method. For each target image, we sample $K = 8$ independent complete trajectories. Each trajectory contains an initial action followed by a corrective action conditioned on the execution state produced by that initial action. Both actions are sampled from the same pre-update policy, and no parameter update occurs during rollout collection.

Algorithm 1 One training iteration of our method

Require: Policy π_{θ_k}, target-image batch $\mathcal{B}$, group size $K = 8$, credit scale $\lambda = 10$

for each target image $I^\star \in \mathcal{B}$ **do**
  for $i = 1, \dots, K$ **do**
    Sample initial program $C_{0,i} \sim \pi_{\theta_k}(\cdot \mid I^\star)$
    Execute $C_{0,i}$ and obtain Stage 0 score

$$S_{0,i} = \begin{cases} \mathrm{RSim}(I_{0,i}, I^\star), & \text{if execution succeeds,} \\ 0, & \text{otherwise} \end{cases}$$

    if $C_{0,i}$ produces a valid render $I_{0,i}$ **then**
      Construct a visual-revision state $(I^\star, I_{0,i}, C_{0,i})$
      Sample $C_{1,i} \sim \pi_{\theta_k}(\cdot \mid I^\star, I_{0,i}, C_{0,i})$
    else
      Construct a repair state $(I^\star, C_{0,i}, e_{0,i})$
      Sample $C_{1,i} \sim \pi_{\theta_k}(\cdot \mid I^\star, C_{0,i}, e_{0,i})$
    end if
    Execute $C_{1,i}$ and obtain

$$S_{1,i} = \begin{cases} \mathrm{RSim}(I_{1,i}, I^\star), & \text{if execution succeeds,} \\ 0, & \text{otherwise} \end{cases}$$

    Compute action-specific returns

$$G_{0,i} = S_{1,i}, \qquad G_{1,i} = S_{1,i} - S_{0,i}$$

  end for
  Mean-center and scale the two return sets independently

$$A_{0,i} = \lambda \left(G_{0,i} - \frac{1}{K} \sum_{j=1}^{K} G_{0,j} \right), \qquad A_{1,i} = \lambda \left(G_{1,i} - \frac{1}{K} \sum_{j=1}^{K} G_{1,j} \right)$$

end for
Optimize the Stage 0 loss over action microbatches with weight 0.5
Optimize the Stage 1 loss over action microbatches with weight 0.5

Equation 7 in the main text summarizes the relative weighting of the two action losses. In the implementation, the two losses are optimized sequentially over action microbatches rather than accumulated into one optimizer update. All Stage 0 and Stage 1 trajectories in an iteration are generated before this optimization begins.

Both Direct-GRPO and our method use the same clipped token-level loss form and Dr.GRPO-style fixed-length normalization. For our method, let $\mathcal{M}_{a,b}$ denote action microbatch b of stage $a \in \{0, 1\}$, and let $\theta_{a,b}$ denote the current policy parameters when that microbatch is processed. For each completion token, the implementation first computes a detached reference log probability from

the current policy,

$$q^{\mathrm{ref}}_{a,i,t} = \mathrm{sg}\big[\log \pi_{\theta_{a,b}}(c_{a,i,t} \mid x_{a,i}, c_{a,i,<t})\big], \tag{9}$$

where $\mathrm{sg}[\cdot]$ denotes stop-gradient and $x_{a,i}$ is the action-specific context. A differentiable forward pass of the same current policy is then used to form

$$r_{a,i,t}(\theta) = \exp\left(\log \pi_\theta(c_{a,i,t} \mid x_{a,i}, c_{a,i,<t}) - q^{\mathrm{ref}}_{a,i,t}\right), \tag{10}$$

with the differentiable forward evaluated at the current parameters $\theta = \theta_{a,b}$ before the corresponding update.

The clipped token-level surrogate is

$$\ell_{a,i,t}(\theta) = -\min\left[r_{a,i,t}(\theta)A_{a,i},\ \mathrm{clip}\left(r_{a,i,t}(\theta), 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}}\right)A_{a,i}\right]. \tag{11}$$

For an action microbatch $\mathcal{M}_{a,b}$, the optimized loss is

$$\mathcal{L}_{a,b} = \frac{w_a}{N_a T_{\max}} \sum_{i \in \mathcal{M}_{a,b}} \sum_{t=1}^{T_{\max}} m_{a,i,t}\, \ell_{a,i,t}(\theta), \qquad w_0 = w_1 = 0.5, \tag{12}$$

where N_a is the number of local actions for stage a in the training iteration, $T_{\max}$ is the maximum completion length, and $m_{a,i,t}$ masks padding and non-completion positions. The denominator remains fixed at the stage level when the actions are split into microbatches. Completions that reach the generation limit are masked from the policy loss.

We use $\epsilon_{\mathrm{low}} = 0.20$ and $\epsilon_{\mathrm{high}} = 0.28$. No group-standard-deviation normalization is applied, and no KL term is added ($\beta = 0$). Only completion tokens contribute to the policy loss. Because the detached reference log probabilities are recomputed from the current policy immediately before each differentiable forward pass, they are not a frozen copy of the policy that generated the rollout. Consistent with this implementation, the recorded clipping ratio is zero throughout the formal 8B run, so the clipping branch is inactive; the advantage-weighted token loss and its gradients remain active.

For the formal 8B run, training uses four ranks with two target images per rank, yielding eight target-image groups per training iteration. Each group contains eight complete trajectories, and each rank receives 16 actions per stage for optimization. With an action microbatch size of two, each stage is processed in eight action microbatches. Under the executed ZeRO-2 configuration, each such microbatch reaches an AdamW update boundary, giving eight updates for Stage 0 followed by eight updates for Stage 1, or 16 AdamW updates per training iteration. Checkpoint labels such as 25, 50, 75, 100, and 125 therefore refer to training iterations rather than individual optimizer updates.

B.2 Prompt and Routing Contract

Three prompt templates are used: direct generation, visual revision, and execution repair. The direct-generation prompt is shared with Direct-SFT and Direct-GRPO and is used for Stage 0 of our method. Stage 1 uses either the revision or repair prompt according to the Stage 0 execution result. The templates below reproduce the prompts used in the formal experiments; bracketed placeholders denote dynamically inserted state information.

B.2.1 Direct Generation Prompt

The direct prompt receives only the target image.

B.2.2 Visual Revision Prompt

When Stage 0 renders successfully, Stage 1 receives the target image, the current Stage 0 render, and the current TikZ source. The source is inserted at `<CURRENT TIKZ PROGRAM>`.

B.2.3 Execution Repair Prompt

When Stage 0 does not produce a valid render, Stage 1 receives the target image, the unsuccessful TikZ attempt, and a compact execution status instead of a current render.

Direct Generation Prompt

You are an expert Image-to-TikZ model and LaTeX/TikZ developer. Study the provided target image carefully and reconstruct it as a high-quality TikZ figure. Focus on the visible objects, geometry, topology, text and mathematical labels, colors, line and arrow styles, relative sizes, alignment, and overall layout. Choose the native TikZ, PGFPlots, circuitikz, tikz-cd, or other LaTeX drawing constructs that best reproduce the image. Do not invent or omit important visible content when it can be represented faithfully.

OUTPUT REQUIREMENTS

- Return only raw LaTeX source code. Do not output Markdown fences, explanations, review prose, or any text outside the document.
- Output exactly one complete standalone document. Start with \documentclass[tikz]{standalone}; only when a small margin is needed to prevent visible clipping, use border=1pt, border=2pt, or border=3pt in the same document class.
- Include every package and TikZ library required by the generated code in the preamble. The document must be self-contained, use native TikZ/PGF drawing, and compile successfully with pdflatex with shell escape disabled. Do not load external files or assets.
- Reconstruct the target image as faithfully as possible while preserving compilability and document completeness.
- Include exactly one \begin{document} and one matching \end{document}. The final non-whitespace content must be \end{document}; stop immediately after it.

Figure 11: Direct generation prompt.

B.3 Execution Routing and State Construction

Routing is determined solely by the execution result of Stage 0. A successful render routes the trajectory to visual revision; otherwise it is routed to repair or completion. The routing decision does not use RSim or any learned threshold.

For the revision route, the Stage 1 multimodal context is ordered as

$$\text{target image} \rightarrow \text{current rendered image} \rightarrow \text{current TikZ source}.$$

The rendered image and TikZ source are produced by the same Stage 0 action, so the second call observes the visual consequence of the exact program it is asked to modify.

For the repair route, the context contains

$$\text{target image} \rightarrow \text{compact execution status} \rightarrow \text{current TikZ attempt}.$$

The status is intentionally compact rather than a full compiler log. Depending on the failure, it can include the tags `hit_max_tokens`, `document_incomplete`, and the leading category of the renderer/compiler error; `render_failed` is used as a fallback.

A generated program is considered complete at the first `\end{document}`. Generation terminates independently for each sequence once this string is produced, subject to a maximum of 8192 completion tokens for either action.

B.4 Training Configuration

The training subsets and initialization sequence are summarized in Appendix C.1. Here we report the optimization and runtime configuration used for Direct-SFT, Direct-GRPO, and our method.

Direct-SFT uses a global batch size of 32 (two GPUs, eight examples per GPU, and two gradient-accumulation steps). Direct-GRPO uses two target prompts per device with gradient accumulation of eight and $K = 8$ sampled completions per target. Across four devices, this corresponds to eight target-image groups per training iteration and one AdamW update per iteration.

Our method also uses two target images per rank on four ranks, yielding eight target-image groups per training iteration; each group contains eight independent complete trajectories. The policy loss

Visual Revision Prompt

You are an expert Image-to-TikZ revision model and LaTeX/TikZ developer.

You are given three aligned inputs describing one revision state:
1. TARGET / REFERENCE IMAGE: the first image is the visual ground truth that the final TikZ should reproduce.
2. CURRENT RENDERED IMAGE: the second image is the rendering produced by the current TikZ program below.
3. CURRENT TIKZ PROGRAM: the editable source code that produced the current rendered image.

The current rendered image and current TikZ program describe the same current state. Compare that state against the target image and revise the current TikZ program so that the new rendered result matches the target image as faithfully as possible. Use the target image as the authority for visible content. Use the current rendered image as execution evidence for what the current program actually draws, and use the current TikZ as the program to edit.

Focus on visible objects, geometry, topology, text and mathematical labels, colors, line and arrow styles, relative sizes, alignment, spacing, and overall layout. Preserve correct parts when appropriate, while fixing missing, extra, misplaced, malformed, or visually incorrect elements. Do not add details that are not visually supported by the target image merely because they appear in the current code.

CURRENT TIKZ PROGRAM
{CURRENT_TIKZ_PLACEHOLDER}

OUTPUT REQUIREMENTS

- Return only the complete revised raw LaTeX source code. Do not output Markdown fences, explanations, review prose, analysis, or any text outside the document.
- Output exactly one complete standalone document. Start with \documentclass[tikz]{standalone}; only when a small margin is needed to prevent visible clipping, use border=1pt, border=2pt, or border=3pt in the same document class.
- Include every package and TikZ library required by the revised code in the preamble. The document must be self-contained, use native TikZ/PGF drawing, and compile successfully with pdflatex with shell escape disabled. Do not load external files or assets.
- The revised program may keep, modify, reorganize, or replace parts of the current program as needed to improve visual fidelity to the target image. Return the full document, not a patch or diff.
- Include exactly one \begin{document} and one matching \end{document}. The final non-whitespace content must be \end{document}; stop immediately after it.

Figure 12: Visual revision prompt.

is evaluated in action microbatches of two. Each rank therefore processes 16 actions per stage as eight action microbatches, which gives 16 AdamW updates per training iteration under the executed ZeRO-2 configuration, as detailed in Appendix B.1. The main run uses random seed 20260823.

For our method, rollout generation uses temperature 1.0, top-p 0.9, repetition penalty 1.0, and a maximum of 8192 tokens for each action. Stage 0 and Stage 1 generation use microbatch sizes of six and three, respectively. Training uses AdamW with weight decay 10^{-2}, a constant learning-rate schedule, no warmup, and gradient clipping at norm 1.0.

B.5 Execution and Reward Environment

Direct-GRPO and our method use the RSim-based execution score defined in Eq. 1 during reinforcement learning. For our method, execution uses TeX Live 2026 and attempts `pdflatex`, `lualatex`, and `xelatex`, with shell escape disabled. Compilation has a 30-second timeout, and rasterization uses 150 DPI before conversion to the 448×448 model input resolution. Failed

Execution Repair Prompt

You are an expert Image-to-TikZ model and LaTeX/TikZ developer.

The first-stage TikZ attempt did not produce a valid rendered image. Use the target image as visual ground truth and the current TikZ attempt as editable state. Complete or repair the program into one complete, self-contained, compilable TikZ document that matches the target image as faithfully as possible.

STAGE-0 STATUS
{FAILURE_STATUS_PLACEHOLDER}

CURRENT TIKZ ATTEMPT
{CURRENT_TIKZ_PLACEHOLDER}

OUTPUT REQUIREMENTS

- Return only raw LaTeX source code. Do not output Markdown fences, explanations, review prose, analysis, or text outside the document.
- Return one complete standalone document, not a patch or diff.
- Preserve useful existing content when appropriate, but freely repair, complete, reorganize, or replace the current attempt when needed.
- The document must compile with shell escape disabled and must not depend on external assets.
- Include exactly one \begin{document} and one matching \end{document}. The final non-whitespace content must be \end{document}; stop immediately after it.

Figure 13: Execution repair prompt.

Table 3: Optimization and runtime configuration for Direct-SFT, Direct-GRPO, and our method. The final column reports the configuration used by our method in the main experiments.

	Direct-SFT	Direct-GRPO	Ours
Training duration	3 epochs	1 epoch / 125 iterations	1 epoch / 125 iterations
Learning rate	1×10^{-4}	5×10^{-6}	5×10^{-6}
Weight decay	–	10^{-2}	10^{-2}
LoRA	$r = 256,\ \alpha = 512$	continued	continued
Group size K	–	8	8
GRPO loss	–	Dr.GRPO	Dr.GRPO
β	–	0	0
Clip range	–	$[0.80, 1.28]$	$[0.80, 1.28]$
Reward normalization	–	none	none
Credit scale λ	–	–	10
Action loss weights	–	–	$0.5/0.5$
Sampling temperature	–	1.0	1.0
Top-p	–	0.9	0.9
Maximum completion	2,048	8,192	8,192 / action
Image resolution	448×448	448×448	448×448
Precision	BF16	BF16	BF16
Gradient checkpointing	✓	✓	✓
Distributed training	DDP	ZeRO-2	ZeRO-2
A100-80GB GPUs	2	4	4

compilation or rendering receives $S_t = 0$. The same execution score is used for both actions; their learning signals differ through the action-specific returns defined in Section 3.2.

C Data and Evaluation Protocol

This section provides additional details on the training subsets, held-out evaluation data, evaluation metrics, inference protocol, and checkpoint selection used throughout the experiments.

C.1 Training Data

All supervised and reinforcement-learning data are drawn from DaTikZ-V4 (Greisinger & Eger, 2026). Direct-SFT is trained on 3,851 eligible image–TikZ pairs obtained from an approximately 4K candidate subset after preprocessing. Direct-GRPO uses 1,000 examples sampled from this supervised-training pool. Our method is initialized from the final Direct-GRPO policy and reuses the same 1,000 target images. The corrective action is generated from execution states produced by the policy's own predictions, so no additional training examples or paired program annotations are introduced.

Table 4: Training data used by the main learning settings. The Direct-GRPO subset is contained in the supervised-training pool, and the same 1K subset is reused by our method.

Method	Training Examples	Initialization	Training Signal
Direct-SFT	3,851	Base	Reference TikZ code
Direct-GRPO	1,000	Direct-SFT	Executed outcome S
Ours	1,000	Direct-GRPO	S_1 / $S_1 - S_0$

For the continued Direct-GRPO controls in Section 4.2, we continue Direct-GRPO from the same final Direct-GRPO checkpoint on the same 1K target images while retaining the single-stage Direct-GRPO objective. We evaluate controls after 100 additional training iterations, matching the training-iteration budget and target-image exposure at the Dev-selected checkpoint of our method, and after 125 additional training iterations as an endpoint comparison. The controls do not match the number of low-level optimizer updates performed by our implementation; the corresponding update schedule is described in Appendix B.

C.2 Held-Out Evaluation Data

We construct the evaluation set independently from the training subsets using the same broad source families as DaTikZ-V4: arXiv, GitHub, and TeX StackExchange. Evaluation examples are restricted to source records from 2026 onward; the examples used in this work span January to mid July 2026. Each retained item contains an executable reference TikZ program and its corresponding rendered target image, and all examples undergo manual review during dataset construction.

The paper uses 1,095 held-out examples. We fix 150 examples as **Dev-150** for checkpoint selection, ablations, and behavioral analysis, and reserve the remaining 945 examples as **Final-945** for final evaluation. Final-945 is not used for checkpoint selection or method development.

Table 5: Composition of the held-out evaluation set. All samples are drawn from source records dated in 2026.

Split	arXiv	GitHub	TeX.SE	Total
Dev-150	82	43	25	150
Final-945	458	309	178	945
All	540	352	203	1,095

The two splits contain distinct evaluation examples and serve different purposes throughout the paper: all checkpoint-dependent analysis is performed on Dev-150, while Final-945 is evaluated only after the checkpoint and inference procedure have been fixed.

C.3 Evaluation Metrics

We evaluate both executability and visual fidelity. *Render success* indicates whether a generated program can be successfully compiled and rasterized. For a valid render I, visual fidelity to the target image $I^\star$ is measured using four complementary image-similarity metrics: RSim (Greisinger & Eger, 2026), DINO (Caron et al., 2021), DreamSim (Fu et al., 2023), and CLIPImg (Radford et al., 2021).

RSim uses the DeTikZify-V2 vision encoder and serves as the only visual optimization reward in Direct-GRPO and our method. DINO, DreamSim, and CLIPImg are used only for evaluation and do not participate in training or execution-aware selection. We report DreamSim similarity as one minus the perceptual distance returned by the official DreamSim implementation.

Table 6: Evaluation metrics and their roles. All main visual results use the fixed-zero convention.

Metric	Role in Training	Main Evaluation Convention
Render success	Execution signal	Success / failure
RSim	RL reward + evaluation	Failure $\rightarrow 0$
DINO	Evaluation only	Failure $\rightarrow 0$
DreamSim	Evaluation only	Failure $\rightarrow 0$
CLIPImg	Evaluation only	Failure $\rightarrow 0$

Under the *fixed-zero* convention, unsuccessful executions receive zero for every visual metric and successful executions retain their measured similarity. This convention is used for the main aggregate results because it reflects both executability and visual quality. When analyzing the visual effect of the corrective action independently of execution recovery, we instead restrict to the *paired-success* subset in which both Stage 0 and Stage 1 render successfully, as defined in Section 4.3.

C.4 Inference and Rendering Protocol

All paper-active evaluations use stochastic decoding with temperature 1.0, top-p 0.9, repetition penalty 1.0, a maximum of 8192 newly generated tokens, and early termination at the first complete `\end{document}`.

Direct models produce a single output from the target image. For our method, Stage 0 is generated from the target image alone and is reported as the single-call inference output. In correction-enabled evaluation, the same policy is called again after Stage 0 execution to produce Stage 1 according to the routing procedure in Appendix B.3.

The final *Selected* output is chosen only from the two programs already produced:

- if only Stage 0 renders, Stage 0 is retained;
- if only Stage 1 renders, Stage 1 is retained;
- if both render, the program with higher RSim is retained;
- ties are assigned to Stage 0;
- if neither renders, the selected output is unsuccessful.

No additional generation is performed by the selection procedure.

Generation parameters and evaluation metrics are unified across the paper-active evaluations. The Direct Final-945 evaluation uses a 120-second compilation timeout, whereas the correction-enabled renderer uses a 30-second timeout; all successful Direct predictions in Final-945 execute within 30 seconds. Both pipelines normalize rendered images to 448×448 before visual evaluation.

C.5 Checkpoint Selection

Checkpoints of our method are evaluated on Dev-150 at the initializer and after 25, 50, 75, 100, and 125 training iterations. For consistency with the training logs and figures, we retain the checkpoint

labels Step 25 through Step 125 throughout the paper. For each checkpoint, we evaluate Stage 0, raw Stage 1, and the execution-aware selected output using the same inference and metric protocol.

Table 7 reports the quantities used to compare the checkpoints. Among the evaluated checkpoints, checkpoint 100 achieves the highest selected render rate as well as the highest selected RSim, DINO, DreamSim, and CLIPImg scores. We therefore fix checkpoint 100 for subsequent single-policy behavioral analyses and for the final evaluation on Final-945. Training-dynamics analyses instead report the full checkpoint trajectory.

Table 7: Checkpoint evaluation for our method on Dev-150. Visual metrics are computed on the execution-aware selected output using the fixed-zero convention. Checkpoint 100 gives the highest selected render rate and all four selected visual metrics.

Checkpoint	Stage 0 Render	Stage 1 Render	Selected Render	RSim	DINO	DreamSim	CLIPImg
Initializer	82.00%	82.67%	82.67%	0.4736	0.7088	0.6497	0.7468
Step 25	87.33%	88.00%	89.33%	0.5161	0.7667	0.7076	0.8116
Step 50	89.33%	88.00%	90.67%	0.5223	0.7805	0.7160	0.8196
Step 75	91.33%	90.00%	92.00%	0.5454	0.7975	0.7386	0.8416
Step 100	**96.67%**	**94.67%**	**97.33%**	**0.5646**	**0.8326**	**0.7677**	**0.8774**
Step 125	95.33%	92.67%	95.33%	0.5549	0.8117	0.7538	0.8666

The selected checkpoint is fixed before Final-945 evaluation. Final-945 is therefore used only to report final performance rather than to choose between training checkpoints or analysis settings.

D Extended Learning Dynamics and Controls

This section provides the complete numerical results behind the learning dynamics reported in the main text, compares our method with continued single-stage Direct-GRPO under matched training-iteration budgets, and quantifies sample-level uncertainty for selected Final-945 comparisons. All Dev-150 checkpoint results follow the evaluation protocol described in Appendix C.

D.1 Complete Learning Dynamics

We evaluate the initializer and checkpoints at steps 25, 50, 75, 100, and 125 on Dev-150. Table 8 reports the execution behavior of both actions together with the four possible execution transitions. We denote successful and failed executions by S and F, respectively. For example, FS denotes a Stage 0 failure followed by a successful Stage 1 repair.

Table 8: Complete execution dynamics of our method on Dev-150. Each row contains 150 examples. SS, SF, FS, and FF denote the execution outcomes of Stage 0 and Stage 1, respectively.

Checkpoint	Stage 0 Render	Stage 1 Render	SS	SF	FS	FF
Initializer	82.00%	82.67%	123	0	1	26
Step 25	87.33%	88.00%	129	2	3	16
Step 50	89.33%	88.00%	130	4	2	14
Step 75	91.33%	90.00%	134	3	1	12
Step 100	96.67%	94.67%	141	4	1	4
Step 125	95.33%	92.67%	139	4	0	7

The execution rates improve substantially over training. Stage 0 rises from 82.00% for the initializer to 96.67% at step 100, while Stage 1 rises from 82.67% to 94.67%. Most of this change comes from the reduction of FF trajectories. As Stage 0 improves, failed initial executions also become less frequent, so the repair route is exercised less often on later Dev-150 checkpoints. Appendix F analyzes routing and repair in more detail.

D.2 Paired-Success Correction Dynamics

To isolate visual correction from execution recovery, we use the paired-success subset defined in Section 4.3. For each visual metric M, the reported quantity is

$$\Delta M = M(I_1, I^\star) - M(I_0, I^\star).$$

Positive values indicate that Stage 1 improves the already executable Stage 0 state.

Table 9 reports the complete paired-success visual dynamics. The correction-outcome categories follow the definition used in the main text: $\Delta\mathrm{RSim} > 0.005$ is an improvement, $|\Delta\mathrm{RSim}| \leq 0.005$ is neutral, and $\Delta\mathrm{RSim} < -0.005$ is a degradation.

Table 9: Visual correction dynamics on the paired-success subset of Dev-150. The four Δ columns report the mean Stage 1 minus Stage 0 change. The last three columns show the composition of correction outcomes under the ± 0.005 RSim neutral region used in the main text.

Checkpoint	Paired n	ΔRSim	ΔDINO	ΔDreamSim	ΔCLIPImg	Improve	Neutral	Degrade
Initializer	123	-0.0031	-0.0027	-0.0039	-0.0017	12.20%	65.04%	22.76%
Step 25	129	$+0.0013$	-0.0014	$+0.0043$	$+0.0042$	28.68%	48.06%	23.26%
Step 50	130	$+0.0098$	$+0.0053$	$+0.0077$	$+0.0016$	43.85%	26.92%	29.23%
Step 75	134	$+0.0121$	$+0.0058$	$+0.0112$	$+0.0063$	42.54%	30.60%	26.87%
Step 100	141	-0.0012	-0.0072	-0.0047	-0.0000	43.26%	19.86%	36.88%
Step 125	139	$+0.0065$	$+0.0057$	$+0.0025$	-0.0013	46.76%	19.42%	33.81%

The complete trajectory highlights the distinction between initial generation quality and corrective behavior. The initializer has negative average Stage 1 changes under all four visual metrics. By steps 50–75, correction becomes more effective, and all four mean changes are positive at step 75. The trend is not monotonic: at step 100, Stage 0 reaches its highest render rate, while the average paired-success change becomes slightly negative. At step 125, RSim, DINO, and DreamSim again show positive average changes.

Checkpoint selection is therefore based on the complete inference result rather than the paired-success correction gain alone. Step 100 provides the strongest selected result on Dev-150 (Appendix C.5), while Appendix F.1 shows that its corrective behavior varies with the quality of the intermediate state.

D.3 Continued Direct-GRPO Controls

We compare our method with continued single-stage Direct-GRPO from the same Direct-GRPO initializer and on the same 1K training images. The $+100$ comparison matches the additional training-iteration budget and target-image exposure at the Dev-selected checkpoint used for final evaluation. The $+125$ comparison uses the corresponding training endpoint. The controls match initialization, training data, training-iteration count, and target-image exposure, but not the number of low-level optimizer updates; our method additionally samples the execution-conditioned corrective action during training.

Table 10 compares single-call outputs on Final-945. For our method, this is the Stage 0 output generated directly from the target image without invoking correction at inference time.

Table 10: Continued Direct-GRPO single-call results on Final-945. Both methods start from the same Direct-GRPO checkpoint and use the same 1K training images. The $+100$ row corresponds to the Dev-selected checkpoint of our method; $+125$ reports the training endpoint.

Additional Iterations	Method	Render	RSim	DINO	DreamSim	CLIPImg
$+100$	Continued Direct-GRPO	88.25%	0.5135	0.7629	0.7019	0.7996
$+100$	Ours (Stage 0)	**92.70%**	**0.5382**	**0.8014**	**0.7330**	**0.8372**
$+125$	Continued Direct-GRPO	87.20%	0.5119	0.7557	0.6984	0.7952
$+125$	Ours (Stage 0)	**93.54%**	**0.5356**	**0.8011**	**0.7346**	**0.8431**

Continued Direct-GRPO improves over the original Direct-GRPO checkpoint, but our Stage 0 output is higher under render success and all four visual metrics at both corresponding training endpoints. The gain from training is therefore already present in the initial action and does not require the second call at inference time.

We also evaluate the continued Direct-GRPO policies with the same execution-conditioned second call. Table 11 reports paired-success visual changes on Final-945. At both corresponding endpoints, the continued single-stage policies produce changes close to zero, whereas our corrective action yields larger positive mean changes across the reported metrics.

Table 11: Continued Direct-GRPO paired-success correction on Final-945. ΔM denotes Stage 1 minus Stage 0 on examples where both outputs render successfully.

Additional Iterations	Method	Paired n	ΔRSim	ΔDINO	ΔDreamSim	ΔCLIPImg
$+100$	Continued Direct-GRPO	828	$+0.0002$	$+0.0019$	$+0.0020$	-0.0007
$+100$	Ours	863	$+0.0073$	$+0.0034$	$+0.0074$	$+0.0031$
$+125$	Continued Direct-GRPO	815	$+0.0017$	$+0.0013$	$+0.0009$	$+0.0008$
$+125$	Ours	869	$+0.0079$	$+0.0046$	$+0.0068$	$+0.0035$

For the $+125$ continued Direct-GRPO correction analysis, paired statistics use the Stage 0 records from the correction-enabled evaluation run. The canonical single-call Direct result is reported separately in Table 10.

D.4 Paired Bootstrap on Final-945

We quantify uncertainty over the fixed Final-945 evaluation set using a paired bootstrap. For each comparison, we resample the same 945 example indices with replacement for both methods and compute the difference in their mean performance. We use 10,000 paired resamples with a fixed random seed of 20260914 and report percentile 95% intervals. Positive differences indicate better performance for the second method.

Table 12 reports three fixed comparisons: from supervised learning to direct execution feedback, from Direct-GRPO to the Stage 1 output of our trained policy, and from raw Stage 1 to execution-aware selection.

Table 12: Paired bootstrap uncertainty on the fixed Final-945 evaluation set ($N = 945$, 10,000 paired resamples). Each entry reports the observed difference with its 95% bootstrap interval. Positive values favor the second method. Render differences are in percentage points; visual metrics use fixed-zero scores. Intervals quantify uncertainty over the fixed evaluation samples rather than variation across training seeds.

Metric	SFT → Direct-GRPO	Direct-GRPO → Ours Stage 1	Stage 1 → Selected
Δ Render Success	$+17.25$ pp $[+13.86, +20.63]$	$+10.90$ pp $[+8.25, +13.54]$	$+1.38$ pp $[+0.74, +2.12]$
Δ RSim	$+0.1154$ $[+0.0962, +0.1345]$	$+0.0738$ $[+0.0580, +0.0892]$	$+0.0203$ $[+0.0162, +0.0249]$
Δ DINO	$+0.1619$ $[+0.1334, +0.1905]$	$+0.1000$ $[+0.0765, +0.1238]$	$+0.0175$ $[+0.0113, +0.0243]$
Δ DreamSim	$+0.1508$ $[+0.1244, +0.1773]$	$+0.0932$ $[+0.0721, +0.1149]$	$+0.0192$ $[+0.0137, +0.0254]$
Δ CLIPImg	$+0.1629$ $[+0.1333, +0.1939]$	$+0.0999$ $[+0.0755, +0.1245]$	$+0.0174$ $[+0.0111, +0.0246]$

The intervals exclude zero for all five metrics in each comparison. Direct execution feedback provides a large improvement over supervised code learning, the trained policy provides a further positive shift over the preceding Direct-GRPO checkpoint, and execution-aware selection adds a smaller but consistent improvement over raw Stage 1.

These intervals quantify uncertainty over the fixed Final-945 sample set. They do not measure variation across independent training runs.

E Credit Assignment Analysis

This section expands the credit-assignment analysis in Section 4.4. We first report the controlled comparison between downstream and local credit for the initial action. We then compare the group-relative preferences induced by alternative credit definitions on the same sampled trajectories, followed by the optimization behavior of the four factorized variants.

E.1 Factorized Credit Variants

The ablation varies the credit assigned to the initial and corrective actions. Stage 0 either receives the final trajectory score S_1 or its local intermediate score S_0. Stage 1 either receives its improvement over the intermediate state, $S_1 - S_0$, or the absolute final score S_1. All variants use the same group centering and scale $\lambda = 10$ defined in Section 3.3.

Table 13: Factorized credit-assignment variants. Variant A is our method. The table reports the raw credit before group centering and scaling.

Variant	Stage 0 credit	Stage 1 credit
A (Ours)	S_1	$S_1 - S_0$
B	S_0	$S_1 - S_0$
C	S_1	S_1
D	S_0	S_1

A versus B isolates the Stage 0 credit while keeping improvement-relative Stage 1 credit fixed. A versus C isolates the Stage 1 credit while keeping downstream Stage 0 credit fixed. Variants C and D both use absolute Stage 1 credit with different Stage 0 definitions.

E.2 Downstream versus Local Stage 0 Credit

Table 14 gives the complete Dev-150 comparison between variants A and B at the checkpoints used in the main analysis. The paired visual changes are computed only on examples for which both stages render successfully.

Table 14: Comparison of downstream Stage 0 credit (A) and local Stage 0 credit (B) on Dev-150. ΔM denotes the mean Stage 1 minus Stage 0 change on the paired-success subset. A and B use the same improvement-relative Stage 1 credit and differ only in whether Stage 0 receives S_1 or S_0.

Checkpoint	Variant	Stage 0 Render	Stage 1 Render	ΔRSim	ΔDINO	ΔDreamSim	ΔCLIPImg
Step 25	A	87.33%	88.00%	+0.0013	−0.0014	+0.0043	+0.0042
	B	84.67%	86.00%	+0.0012	−0.0015	+0.0005	−0.0007
Step 50	A	89.33%	88.00%	+0.0098	+0.0053	+0.0077	+0.0016
	B	90.00%	89.33%	+0.0002	+0.0023	+0.0038	+0.0001
Step 75	A	91.33%	90.00%	+0.0121	+0.0058	+0.0112	+0.0063
	B	88.00%	88.67%	−0.0008	−0.0023	−0.0037	−0.0014

The comparison separates Stage 0 quality from the behavior of the subsequent corrective action. At step 50, B has a slightly higher Stage 0 render rate than A (90.00% versus 89.33%), and its Stage 0 RSim-FZ is also higher (0.5095 versus 0.4960). Its paired Stage 1 correction, however, is nearly neutral: $\Delta\mathrm{RSim} = +0.0002$, compared with $+0.0098$ for A. By step 75, A retains a positive correction gain of $+0.0121$, whereas B becomes slightly negative at -0.0008.

Thus, a stronger intermediate output at one checkpoint does not imply a stronger subsequent correction. Using S_1 rather than S_0 for Stage 0 changes which initial actions are favored according to the quality reached after the corrective action.

E.3 Direct Audit of Credit-Induced Preferences

The trained ablations compare the behavior produced by different credit definitions. We additionally inspect the credit signals while holding the sampled trajectories fixed. For this analysis, we use the completed A training run, containing 1,000 target-image groups and eight independently sampled complete trajectories per group, for a total of 8,000 trajectories.

For Stage 0, we compare the downstream advantage

$$A_0(S_1) = S_1 - \overline{S_1}$$

with the local advantage

$$A_0(S_0) = S_0 - \overline{S_0}.$$

For Stage 1, we compare the improvement-relative advantage

$$A_1(S_1 - S_0) = (S_1 - S_0) - \overline{(S_1 - S_0)}$$

with the absolute-score advantage

$$A_1(S_1) = S_1 - \overline{S_1}.$$

The common factor $\lambda = 10$ is omitted because it does not affect correlation, sign, or ranking.

Table 15: Comparison of group-relative advantages induced by alternative credit definitions on the same 8,000 trajectories from 1,000 target-image groups. Sign flips are computed over all trajectories, with exact-zero advantages counted as neither positive nor negative. Top-1 change denotes the fraction of groups whose highest-advantage trajectory changes under the alternative definition.

Advantage Comparison	Pearson	Spearman	Sign Flip	Group Top-1 Change	Pairwise Ordering Change
$A_0(S_1)$ vs. $A_0(S_0)$	0.7408	0.7083	21.4%	46.1%	–
$A_1(S_1 - S_0)$ vs. $A_1(S_1)$	0.3848	0.3146	37.2%	74.5%	41.76%

For Stage 0, the two definitions remain moderately correlated but produce different group-relative preferences. The advantage sign changes for $1{,}715/8{,}000 = 21.4\%$ of trajectories, and the highest-advantage trajectory changes in $461/1{,}000 = 46.1\%$ of target-image groups.

The distinction is larger for Stage 1. Improvement-relative and absolute credit have a Pearson correlation of 0.3848 and a Spearman correlation of 0.3146. The advantage sign changes for $2{,}974/8{,}000 = 37.2\%$ of trajectories, and the highest-advantage corrective trajectory changes in $745/1{,}000 = 74.5\%$ of groups.

We additionally compare every unordered pair of trajectories within each group. For a pair (i, j), we compare the sign of

$$A_{1,i}(S_1 - S_0) - A_{1,j}(S_1 - S_0)$$

with the sign of

$$A_{1,i}(S_1) - A_{1,j}(S_1).$$

Ties form a separate ordering category. Of the $28{,}000$ within-group trajectory pairs, $11{,}693$ change their relative ordering, corresponding to 41.76%. Thus, $S_1 - S_0$ and S_1 do not merely rescale the same corrective preference; they change the relative ordering of corrective actions within a group.

E.4 Optimization Behavior of the Credit Variants

The fixed-trajectory audit above compares the learning signals directly. The corresponding training runs also differ in optimization behavior. Table 16 summarizes the four variants.

Variant A completes all 125 training iterations without the length or termination failure observed in the alternative runs. Variant B behaves normally through the checkpoints used for the A–B comparison but develops a late-training length pathology. Its mean Stage 1 completion length is 482 tokens at step 80, 965 tokens at step 81, and 3,663 tokens at step 82. At step 83 it increases to 8,067 tokens, with 98.4% of Stage 1 completions reaching the generation limit and no Stage 1 completion terminating normally at `\end{document}`.

Table 16: Optimization behavior observed for the four factorized credit variants.

Variant	Stable Until	First Abnormal Step	Observed Behavior
A	125	–	Completed normally
B	~ 81	82	Late length/termination pathology
C	24	25–26	Early zero-signal collapse
D	25	26	Early zero-signal collapse

Variants C and D show an earlier shared failure regime. Both use absolute Stage 1 credit S_1 but different Stage 0 credits. Their execution behavior collapses around step 26. Table 17 shows the transition around this point.

Table 17: Training diagnostics around the C/D collapse. "Hit Max" is the fraction of Stage 1 generations reaching the token limit, "End Doc" is the fraction terminating normally at `\end{document}`, and A_0/A_1 nonzero counts the trajectories receiving nonzero group-relative advantages. Each training iteration contains 64 complete trajectories.

Variant	Step	Stage 1 Tokens	Hit Max	End Doc	Stage 0 Compile	Stage 1 Compile	FF	A_0 Nonzero	A_1 Nonzero
C	25	2723	31%	45%	10	11	52	48	48
C	26	5639	69%	0%	0	0	64	0	0
C	27	7425	91%	0%	0	0	64	0	0
D	25	1313	11%	80%	38	39	23	64	57
D	26	7041	86%	0%	0	0	64	0	0
D	29–31	8192	100%	0%	0	0	64	0	0

From step 26 onward, both C and D enter an all-zero execution regime. All 64 trajectories in the observed training iteration fail at both stages, so the fixed-zero execution score gives identical zero returns within the group. The resulting group-relative advantages, policy loss, and gradient norm are therefore zero, while Stage 1 generations become substantially longer and normal document termination disappears.

At the corresponding steps, A and B remain executable and retain nonzero training signals on the same sample groups. C and D use different Stage 0 credits but share the absolute Stage 1 definition, making it the common credit choice associated with the observed early failure regime.

E.5 Summary

The ablations and fixed-trajectory audit show that the two credit choices affect different parts of learning. Using downstream Stage 0 credit changes which initial programs are favored according to the quality reached after the corrective action. Using improvement-relative Stage 1 credit produces substantially different corrective preferences from absolute final-score credit, changing advantage signs and rankings for a large fraction of the same sampled trajectories.

These differences also appear in training behavior. Variant A combines downstream Stage 0 credit with improvement-relative Stage 1 credit and completes the full training run, while the alternative variants exhibit distinct correction behavior or optimization failures.

F State-Dependent Correction and Post-Execution Behavior

This section expands the state-dependent correction analysis in Section 4.5. We first report the complete statistics behind Figure 4, then examine the same behavior across nearby checkpoints. We finally analyze revision and repair routing, execution transitions, and the optional selection rule applied when both outputs are available.

F.1 Full State-Dependent Correction Analysis

Figure 4 analyzes the Dev-selected checkpoint at step 100. As in the main text, we restrict this analysis to paired-success examples for which both Stage 0 and Stage 1 produce valid renders. We partition examples according to their Stage 0 RSim and define

$$\Delta M = M(I_1, I^\star) - M(I_0, I^\star).$$

We separately report

$$P(S_1 > S_0),$$

which denotes the fraction of examples with a strictly higher Stage 1 RSim. This quantity uses the threshold $\Delta\mathrm{RSim} > 0$, whereas the correction-outcome categories used in Section 4.3 use a ± 0.005 neutral region.

Table 18: Complete state-dependent correction statistics for our method at checkpoint 100 on Dev-150. The analysis is restricted to paired-success examples. Δ values denote Stage 1 minus Stage 0. The 95% intervals are bootstrap intervals over examples within each bucket. Improve, neutral, and degrade use the ± 0.005 RSim threshold from the main learning-dynamics analysis, whereas $P(S_1 > S_0)$ uses the strict $\Delta\mathrm{RSim} > 0$ criterion.

Stage 0 RSim	n	ΔRSim	95% CI	$P(S_1 > S_0)$	Improve	Neutral	Degrade
$[0.0, 0.4)$	17	$+0.0260$	$[+0.0103, +0.0442]$	76.5%	58.8%	29.4%	11.8%
$[0.4, 0.5)$	29	$+0.0029$	$[-0.0103, +0.0168]$	55.2%	41.4%	20.7%	37.9%
$[0.5, 0.6)$	39	$+0.0021$	$[-0.0087, +0.0128]$	53.8%	48.7%	7.7%	43.6%
$[0.6, 0.7)$	26	-0.0026	$[-0.0287, +0.0194]$	53.8%	53.8%	15.4%	30.8%
$[0.7, 1.0]$	30	-0.0235	$[-0.0593, +0.0026]$	23.3%	20.0%	33.3%	46.7%

The full statistics reinforce the pattern shown in the main figure. The weakest intermediate states receive the largest average correction: for Stage 0 RSim below 0.4, Stage 1 improves RSim by 0.0260 on average and achieves a higher RSim in 76.5% of paired examples. The average gain decreases as Stage 0 quality increases. For the highest-quality bucket, the mean change becomes -0.0235, and only 23.3% of examples obtain a strictly higher Stage 1 RSim.

The same contrast is visible under metrics that are not used to train the policy. Table 19 reports the corresponding DINO, DreamSim, and CLIPImg changes. All four metrics improve on average in the weakest bucket, whereas all four decrease in the highest-quality bucket.

Table 19: Cross-metric Stage 1 changes at checkpoint 100. RSim is the optimization reward; DINO, DreamSim, and CLIPImg are evaluation-only metrics.

Stage 0 RSim	ΔRSim	ΔDINO	ΔDreamSim	ΔCLIPImg
$[0.0, 0.4)$	$+0.0260$	$+0.0195$	$+0.0352$	$+0.0191$
$[0.4, 0.5)$	$+0.0029$	$+0.0002$	-0.0001	$+0.0015$
$[0.5, 0.6)$	$+0.0021$	-0.0166	-0.0093	-0.0033
$[0.6, 0.7)$	-0.0026	-0.0222	-0.0189	$+0.0030$
$[0.7, 1.0]$	-0.0235	-0.0044	-0.0133	-0.0109

The intermediate buckets are less uniform across metrics, while the endpoint contrast is consistent: weak intermediate states leave more room for correction, whereas additional modification of already strong renders is more likely to degrade the result.

F.2 State Dependence across Checkpoints

The main analysis uses step 100 because it is the checkpoint selected on Dev-150 for final evaluation. We repeat the same analysis at steps 75 and 125 using identical RSim buckets and definitions.

Table 20 summarizes the weakest and strongest buckets. Across all three later checkpoints, weak Stage 0 states receive a larger average gain and are more likely to obtain a higher Stage 1 RSim than strong Stage 0 states.

Table 20: State-dependent correction across later checkpoints of our method. "Weak" denotes Stage 0 RSim below 0.4 and "strong" denotes Stage 0 RSim in $[0.7, 1.0]$. All statistics use paired-success examples and the same fixed bucket boundaries.

Checkpoint	Weak ΔRSim	Weak $P(S_1 > S_0)$	Strong ΔRSim	Strong $P(S_1 > S_0)$
Step 75	$+0.0412$	58.3%	$+0.0045$	28.0%
Step 100	$+0.0260$	76.5%	-0.0235	23.3%
Step 125	$+0.0337$	81.3%	-0.0158	28.0%

The exact magnitude changes over training, but the separation between weak and strong intermediate states is preserved. At step 75, the strongest bucket still has a small positive average ΔRSim, but Stage 1 achieves a higher RSim in only 28.0% of those examples. At steps 100 and 125, the strongest bucket has a negative average change.

F.3 Revision and Repair Routing during Training

The Stage 1 route is determined independently for every trajectory by its own Stage 0 execution result. A valid Stage 0 render enters visual revision; an unsuccessful execution enters repair.

Across the completed training run, the 1,000 target-image groups contain 8,000 complete trajectories. Of these, 7,447 trajectories (93.1%) enter visual revision and 553 (6.9%) enter repair. The distribution across groups is shown in Table 21.

Table 21: Routing composition of the 1,000 training groups. Each group contains eight independently sampled complete trajectories. A mixed group contains at least one revision trajectory and at least one repair trajectory.

Group composition	Groups	Fraction
Revision only	738	73.8%
Mixed revision / repair	259	25.9%
Repair only	3	0.3%
All groups	1,000	100%

The 259 mixed groups also show that a training group contains independently sampled Stage 0 actions rather than one shared Stage 0 state followed by multiple corrective candidates. Each Stage 1 action operates on the execution state produced by its own Stage 0 program.

F.4 Execution Transitions and Repair

We next examine how the corrective action changes execution status. We denote the four possible transitions by SS, SF, FS, and FF. The FS transition corresponds to execution repair, whereas SF denotes an initially valid program that becomes invalid after correction.

Table 22: Execution transitions for the initializer and the Dev-selected checkpoint of our method. "Repair success" is $P(S_1 \text{ succeeds} \mid S_0 \text{ fails})$.

Split	Policy	SS	SF	FS	FF	Stage 0 failures	Repair success
Dev-150	Initializer	123	0	1	26	27	3.7%
Dev-150	Step 100	141	4	1	4	5	20.0%
Final-945	Initializer	780	5	6	154	160	3.8%
Final-945	Step 100	863	13	14	55	69	20.3%

Training substantially improves Stage 0 executability, reducing Final-945 Stage 0 failures from 160 for the initializer to 69 at checkpoint 100. Among the remaining failed states, Stage 1 repairs 14 of 69 programs, compared with 6 of 160 for the initializer. The improvement is therefore already

visible before correction is invoked, while the second action adds repair capability for the failed states that remain.

The opposite SF transition also remains possible. On Final-945, 13 initially successful Stage 0 programs fail after Stage 1. This motivates retaining the initial output when the corrective action makes execution worse.

F.5 Execution-Aware Selection

For correction-enabled inference, both Stage 0 and Stage 1 are available. Because the effect of Stage 1 depends on the intermediate state, we additionally evaluate the selection rule defined in Appendix C.4. The rule chooses only between the two programs already produced and requires no additional model generation.

Table 23 compares Stage 0, raw Stage 1, and the selected output at the fixed checkpoint 100.

Table 23: Stage 0, raw Stage 1, and execution-aware selection for our method at checkpoint 100. Visual metrics use the fixed-zero convention.

Split	Output	Render	RSim	DINO	DreamSim	CLIPImg
Dev-150	Stage 0	96.67%	0.5479	0.8225	0.7554	0.8668
	Stage 1	94.67%	0.5358	0.7994	0.7348	0.8485
	Selected	**97.33%**	**0.5646**	**0.8326**	**0.7677**	**0.8774**
Final-945	Stage 0	92.70%	0.5382	0.8014	0.7330	0.8372
	Stage 1	92.80%	0.5446	0.8046	0.7401	0.8409
	Selected	**94.18%**	**0.5649**	**0.8221**	**0.7593**	**0.8583**

On Dev-150, Stage 0 is already strong at checkpoint 100, while raw Stage 1 is lower on the aggregate metrics. The state-dependent analysis nevertheless shows that the corrective action remains useful for weaker intermediate states. Selection retains those gains while keeping Stage 0 when it is preferable, raising selected RSim-FZ to 0.5646.

On Final-945, raw Stage 1 is slightly higher than Stage 0 under all four visual metrics, and selection provides a further improvement: render success rises from 92.80% for raw Stage 1 to 94.18%, while RSim-FZ increases from 0.5446 to 0.5649. As reported in Appendix D.4, the paired bootstrap interval for each Stage 1-to-Selected improvement excludes zero on the fixed Final-945 sample set.

Stage 0 remains the single-call output of the trained policy. The corrective action and selection provide an optional additional inference mode when execution and a second model call are available.

F.6 Summary

The extended analysis supports three observations. First, correction is state dependent: weak intermediate renders receive larger gains, while strong renders are more susceptible to degradation after further modification. Second, the same separation appears across nearby checkpoints rather than only at the Dev-selected checkpoint. Third, training improves the initial action while also producing visual revision and execution-repair behavior in the corrective action; this second action can be used optionally at inference time.