BEYOND INSTRUCTION FOLLOWING: LEARNING GROUNDED SKILL-FOLLOWING WITH SKILL CONTRACTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Instruction following typically enforces discrete, response-level requirements, whereas an expert-authored skill prescribes procedural requirements spanning multiple phases and environment interactions. Given such a skill, we aim to train the executor to execute all required phases successfully instead of focusing solely on the final answer. We therefore introduce **Grounded Skill-Following**, which requires an agent to execute a fixed, expert-authored skill across its required phases by grounding decisions in environment observations. To achieve verifiable procedural execution, we formulate each skill as a *skill contract* combining visible skill instructions with an explicit *contract runtime*. The contract runtime specifies the required phases, admissible actions, permitted transitions, and accepted termination. This contractual structure naturally provides a dense, verifiable training signal throughout execution. We leverage this by introducing *Verified Progress Credit*, which assigns rewards upon the initial completion of contract milestones and aggregates them into the trajectory return to guide policy optimization. During rollout, the contract runtime continuously tracks state transitions to provide *Contract-State Feedback*, which indicates whether the latest action is accepted and guides the agent toward valid next actions. To measure procedural compliance, we introduce the **Protocol Completion Rate (PCR)**—defined as reaching accepted termination through all required phases—and decouple it from the final Task Outcome. Jointly trained with our framework, Qwen3.5-4B achieves Protocol Completion Rates of 99.27% on Math and 99.96% on Search, while slightly outperforming original baselines in Task Outcome (82.95% and 46.61%, respectively). Controlled studies examine how skill instructions, training signals, and contract-state feedback affect both metrics, while withholding interventions evaluate behavioral dependence on observation content.

1 INTRODUCTION

LLMs are trained to generate responses that satisfy natural-language instructions, a capability commonly referred to as *instruction following* (Wei et al., 2021; Sanh et al., 2021; Ouyang et al., 2022), with existing benchmarks evaluating compliance mainly on discrete, response-level requirements (Zhou et al., 2023; Wen et al., 2024)(Figure 1(a)). Tool-using agents extend this capability by grounding actions and decisions in environment observations (Schick et al., 2023; Nakano et al., 2021; Yao et al., 2022). Recent systems package instructions, tools, and execution strategies into reusable skills (Zhang et al., 2025; Wang et al., 2023a; 2024). Here, an expert-authored *skill* prescribes procedural requirements spanning multiple phases and environment interactions, defining the order of admissible actions, the reliance on observation evidence, and valid completion conditions. Research on the skill lifecycle spans retrieval (Wang et al., 2026b; Su et al., 2026), adaptation (Dang et al., 2026; Lu et al., 2026b), composition (Xia et al., 2026; Shi et al., 2026a; Zhang et al., 2026), and verified execution (Zhu et al., 2026; Tan et al., 2026) (Figure 1(b)). While these efforts effectively acquire, adapt, or evaluate skills, they rarely enforce or evaluate whether an agent strictly adheres to prescribed procedural constraints across all required phases. Because an agent can produce a correct final answer while bypassing the skill's required procedure, outcome correctness alone is insufficient to establish procedural compliance.

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.

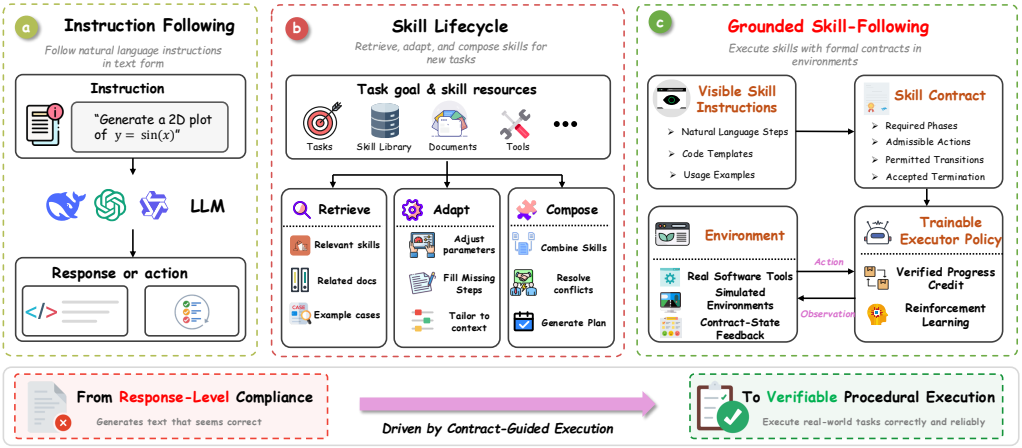

Figure 1: Complementary views of skills. Grounded skill-following fixes an expert-authored skill and trains its executor, alongside local instruction compliance and skill-lifecycle optimization. The views overlap and are not a historical taxonomy.

We term this unaddressed setup **Grounded Skill-Following**, a setting in which an expert-authored skill remains fixed and visible, and the executor must strictly fulfill all required phases (Figure 1(c)). Here, following the prescribed procedure is itself part of the task requirements. Methods such as Search-R1 (Jin et al., 2025b) and ReTool (Feng et al., 2026) optimize executors through reinforcement learning, primarily targeting final Task Outcomes. These outcomes alone do not establish procedural compliance. Moreover, when rewards require completing the entire procedure, unsuccessful trajectories can receive identical rewards despite making different progress. The challenge is therefore to make skill requirements verifiable during execution and provide learning signals that distinguish intermediate procedural progress.

To operationalize fixed skill specifications during agent-environment interaction, we represent each skill as a *skill contract*. Beyond static prompt instructions (e.g., natural language steps or code templates), a contract couples visible skill instructions with an explicit *contract runtime* that formalizes procedural bounds, including required phases, admissible actions, permitted transitions, and accepted termination. By continuously inspecting agent actions against environment observations, the runtime dynamically tracks phase progression, blocks protocol violations, and guides valid next actions.

The contract runtime connects execution checking, training, and evaluation: it checks action admissibility, assigns credit for verified progress, and determines whether the prescribed procedure has been completed. For training, we optimize the executor policy via reinforcement learning using GRPO (Shao et al., 2024; Zeng et al., 2025; Lu et al., 2026a) (Figure 1(c)). To address sparse task rewards that appear only upon accepted termination, the runtime provides dense learning signals by assigning *Verified Progress Credit* as contract milestones are reached, while offering *Contract-State Feedback* during rollout to guide valid next steps. For evaluation, the same runtime acts as an objective judge, recording *Protocol Completion* when a trajectory satisfies all required phases and achieves accepted termination. We report the resulting **Protocol Completion Rate (PCR)** by decoupling it from Task Outcome, thereby isolating procedural compliance from end-task performance.

We evaluate our approach using five executors across the Qwen2.5 and Qwen3.5 model families on Math and Search domains. Contract training achieves remarkably high Protocol Completion Rates (PCR) across both domains. Crucially, higher PCR does not uniformly translate into superior Task Outcomes, demonstrating that procedural compliance is largely decoupled from end-task performance. Controlled studies and observation-withholding experiments analyze the individual contributions of visible skill instructions, training signals, observation content, and Contract-State Feedback. Together, these findings establish that verifiable procedural execution with a fixed, expert-authored skill can be effectively trained and evaluated through a unified contract runtime.

Our main contributions are summarized as follows:

- **Formulation:** We formalize Grounded Skill-Following to achieve verifiable procedural execution on fixed expert skills across required phases, explicitly decoupling Protocol Completion from Task Outcome.

- **Methodology:** We present a *skill contract* runtime featuring *Contract-State Feedback* for step-level action guidance and *Verified Progress Credit* for dense reinforcement learning rewards.
- **Findings:** Across two domains and five executors, we show that high Protocol Completion Rate can be trained without proportional gains in Task Outcome, revealing task-dependent impacts of instructions and observations.

2 Related Work

2.1 From Instruction Following to Agent Skills

Research on instruction compliance primarily targets response-level constraints within static contexts (Wei et al., 2021; Sanh et al., 2021; Wang et al., 2023b; Ouyang et al., 2022), measured by benchmarks that evaluate explicit and compositional requirements (Zhou et al., 2023; Wen et al., 2024). Tool-using architectures extend this setting to interactive environments, requiring models to interleave reasoning actions with observation feedback (Schick et al., 2023; Nakano et al., 2021; Yao et al., 2022). More recently, agent skills have emerged as a paradigm to package instructions, tool invocations, and execution strategies into structured, reusable procedures (Zhang et al., 2025; Wang et al., 2023a; 2024). Literature in this space heavily emphasizes skill discovery and retrieval (Wang et al., 2026b; Su et al., 2026), skill evolution and composition (Xia et al., 2026; Shi et al., 2026a; Zhang et al., 2026), or skill extraction and internalization into model weights (Dang et al., 2026; Lu et al., 2026b; Wang et al., 2026a; Lin et al., 2026).

Closest to our setting are SkillCoach and SKT (Zhu et al., 2026; Tan et al., 2026), which evaluate or adapt provided skills to maximize task outcomes. However, they allow skills to be modified or treat them as soft prompts, leaving phase-level execution unverified. In contrast, *grounded skill-following* treats an expert skill as a fixed, non-negotiable contract. Instead of adapting the skill or taking outcome-driven shortcuts, our contract runtime enforces and evaluates strict phase-by-phase protocol compliance during agent-environment interaction.

2.2 Verifiable Control and Intermediate Supervision

Verifiable control uses finite-state reward machines to structure long-horizon reinforcement learning (Icarte et al., 2018), while process supervision and turn-level rewards provide dense credit assignment for reasoning and tool-use trajectories (Lightman et al., 2024; Zeng et al., 2025; Jin et al., 2025b; Feng et al., 2026; Lu et al., 2026a). In parallel, constrained decoding forces output grammars during inference (Ugare et al., 2024), dynamic interventions like HASP convert skills into execution rules (Liu et al., 2026), and auditing methods motivate testing procedural execution independently from outcomes (Ma et al., 2025; Huang et al., 2026; Zhou et al., 2026).

Our approach differs fundamentally in how procedural specifications are formalized, trained, and evaluated. Unlike reward machines that require rigid state definitions or constrained decoding that acts as an inference-only filter, our *skill contract* pairs visible instructions with an explicit contract runtime to handle dynamic phase transitions. Furthermore, while existing agentic RL remains primarily outcome-driven, our *contract runtime* provides *contract-state feedback* and *verified progress credit* to optimize phase-by-phase compliance during training. For evaluation, it decouples *Protocol Completion Rate (PCR)* from *Task Outcome* to measure procedural compliance independently.

3 Learning Grounded Skill-Following with Skill Contracts

Given a task and an expert-authored skill, we train an executor to follow the required procedure while grounding actions in environment observations. A skill contract pairs visible skill instructions with an explicit contract runtime, providing Contract-State Feedback during rollout and Verified Progress Credit added to the trajectory return. Figure 2 summarizes the framework.

3.1 Problem and Learning Requirement

Contract representation and runtime. For skill j, we formalize the static skill contract as

$$\mathcal{C}_j = (z_j, \mathcal{Q}_j, \mathcal{M}_j, \mathrm{Step}_j). \tag{1}$$

Here, z_j contains the visible skill instructions, $\mathcal{Q}_j$ is the set of required phases, $\mathcal{M}_j$ is the set of credit-bearing milestones, and Step_j is the transition function governing the contract runtime.

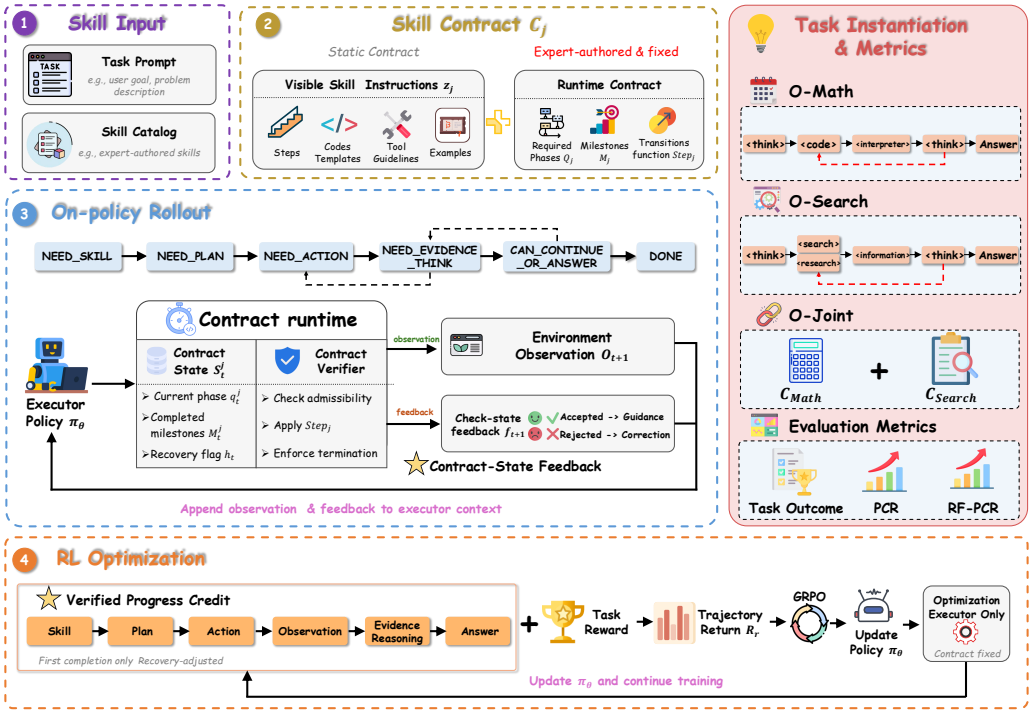

Figure 2: Skill-contract training for grounded skill-following. The contract $\mathcal{C}_j$ pairs visible skill instructions z_j with a contract runtime. The runtime checks each action, credits newly verified progress, and provides contract-state feedback after each nonterminal transition. The feedback describes the current phase without exposing internal bookkeeping.

Operationalized during agent-environment interaction, the contract runtime integrates two core mechanisms:

- **Dynamic Contract State (S_t^j):** Maintained at step t as $S_t^j = (q_t^j, \mathcal{M}_t^j, h_t)$, comprising the current required phase $q_t^j \in \mathcal{Q}_j$, completed milestones $\mathcal{M}_t^j \subseteq \mathcal{M}_j$, and recovery flag $h_t \in \{0, 1\}$. The current phase dictates admissible next actions and valid termination, while milestones track progress along the sequence.
- **Contract Verifier:** Implements the state transition function Step_j alongside a task adapter. The verifier checks action admissibility against environment observations, verifies phase transitions $S_t^j \to S_{t+1}^j$, and enforces accepted termination conditions.

Both visible skill instructions z_j and the contract runtime logic are specified from the expert-authored skill (Appendix Tables 3–6).

Why Task Outcome alone is uninformative. GRPO samples G trajectories for a prompt and normalizes their trajectory returns within the group (Shao et al., 2024). For trajectory return R_i, group standard deviation s_R, and a positive stabilizer ϵ, the group-normalized advantage is

$$\widehat{A}_i = \frac{R_i - \bar{R}}{s_R + \epsilon}, \qquad \bar{R} = \frac{1}{G}\sum_{g=1}^{G} R_g. \tag{2}$$

When trajectory returns depend solely on the binary Task Outcome, all trajectories in a group frequently share the same return, yielding $s_R = 0$ and $\widehat{A}_i = 0$ for all i. Under these zero-advantage conditions, the policy update provides no learning signal to distinguish among trajectories. Adding Verified Progress Credit to the trajectory return can break these zero-variance ties when trajectories accumulate different recovery-adjusted progress totals, even when Task Outcome is sparse or uniformly zero.

3.2 CONTRACT-STATE FEEDBACK AND VERIFIED PROGRESS CREDIT

One contract transition. Let x_t^j be the model-visible context before action t, and let $S_t^j = (q_t^j, \mathcal{M}_t^j, h_t)$ be the current contract state. Interaction starts at $t = 0$ with the initial state $S_0^j = (\texttt{NEED_SKILL}, \varnothing, 0)$. The initial context x_0^j contains the system scaffold and task prompt, while visible skill instructions z_j are appended upon the first accepted skill call. At step t, the policy samples an action:

$$a_t \sim \pi_\theta(\cdot \mid x_t^j). \tag{3}$$

The contract runtime then executes a single state transition via Step_j:

$$(S_{t+1}^j, o_{t+1}, f_{t+1}) = \mathrm{Step}_j(S_t^j, a_t), \tag{4}$$

where $S_{t+1}^j = (q_{t+1}^j, \mathcal{M}_{t+1}^j, h_{t+1})$. Specifically, the contract verifier evaluates action a_t against the current phase q_t^j. If a_t is inadmissible, the verifier rejects the action, leaves q_t^j and $\mathcal{M}_t^j$ unchanged, sets $h_{t+1} = 1$, sets $o_{t+1} = \perp$, and generates Contract-State Feedback f_{t+1}. If a_t is admissible, the verifier accepts the action, applies its action-specific phase transition, updates completed milestones $\mathcal{M}_{t+1}^j$, and resets $h_{t+1} = 0$. Accepted environment actions additionally trigger the task adapter to return environment observation o_{t+1}, whereas accepted non-environment actions set $o_{t+1} = \perp$; an accepted action need not advance the phase, as in the optional Search evidence-organization transition. The recovery flag h_t tracks whether actions follow rejected feedback and is used to scale credit for newly completed milestones. The feedback message f_{t+1} explicitly specifies action admissibility, the current required phase, and an admissible next action. When Contract-State Feedback is disabled, h_t remains zero and f_{t+1} is omitted. The next context is updated via concatenation:

$$x_{t+1}^j = x_t^j \oplus a_t \oplus \begin{cases} z_j \oplus f_{t+1}, & \text{if the skill call is accepted for the first time,} \\ o_{t+1} \oplus f_{t+1}, & \text{otherwise,} \end{cases} \tag{5}$$

where $\perp$ elements are omitted. Accepted termination ends the rollout without trailing feedback. Appendix A.1 details the complete runtime execution order.

Trajectory return. For trajectory i, let $S_{i,t}^j = (q_{i,t}^j, \mathcal{M}_{i,t}^j, h_{i,t})$ be the contract state before action t.The newly verified milestones at step t are

$$\Delta\mathcal{M}_{i,t}^j = \mathcal{M}_{i,t+1}^j \setminus \mathcal{M}_{i,t}^j. \tag{6}$$

We call the reward assigned to these first completions *Verified Progress Credit*.For milestone weight $w_m > 0$, recovery coefficient $\rho \in (0, 1]$ ($\rho = 0.90$ in our main experiments), and total steps T_i, the trajectory return is

$$R_i = r_{\mathrm{task},i} + \sum_{t=0}^{T_i-1} \rho^{h_{i,t}} \sum_{m \in \Delta\mathcal{M}_{i,t}^j} w_m. \tag{7}$$

The binary Task Outcome $r_{\mathrm{task},i} \in \{0, 1\}$ equals 1 only if the trajectory reaches accepted termination with a correct final answer, and 0 otherwise.Each milestone contributes to credit at most once per trajectory.The recovery flag $h_{i,t} \in \{0, 1\}$ specifically discounts progress credit completed immediately after a rejected action ($h_{i,t} = 1$).When Task Outcomes tie across a sampled group, Verified Progress Credit distinguishes trajectories whenever their recovery-adjusted credit totals differ.Different milestone sets do not by themselves guarantee distinct totals because milestone weights and recovery discounts can coincide.Appendix A.3 derives conditions under which Verified Progress Credit alters normalized advantages and explains why return variance alone is insufficient for a useful parameter gradient.Contract-State Feedback updates the model-visible context without altering trajectory return values.Appendix Table 5 reports all exact milestone weights.

3.3 OPTIMIZATION AND TASK INSTANTIATION

Policy optimization. We optimize the policy using a clipped GRPO objective (Shao et al., 2024). For trajectory i, let $\mathcal{L}_i$ denote the set of trainable token positions and $\widehat{A}_i$ the trajectory-level group-normalized advantage. The objective is

$$\mathcal{J}(\theta) = \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\mathcal{L}_i|} \sum_{\ell \in \mathcal{L}_i} \left[\min\left(r_{i,\ell}(\theta)\widehat{A}_i, \mathrm{clip}\left(r_{i,\ell}(\theta), 1-\epsilon_-, 1+\epsilon_+\right)\widehat{A}_i \right) - \beta d_{i,\ell}^{\mathrm{KL}} \right], \tag{8}$$

where

$$r_{i,\ell}(\theta) = \frac{\pi_\theta(a_{i,\ell} \mid x_{i,\ell})}{\pi_{\theta_{\text{old}}}(a_{i,\ell} \mid x_{i,\ell})},$$

β is the KL penalty coefficient, and ϵ_- and ϵ_+ specify the clipping bounds. Here, $a_{i,\ell}$ is a generated token and $x_{i,\ell}$ its preceding context in trajectory i. The term $d_{i,\ell}^{\text{KL}}$ regularizes the policy relative to a frozen reference policy, with its estimator and reference initialization specified in Appendix C.

Visible skill instructions, environment observations, and Contract-State Feedback remain in context with a loss mask of zero, while all model-generated tokens in a trajectory share the single trajectory-level advantage $\widehat{A}_i$. The method therefore uses trajectory-level credit assignment rather than exact action-level or token-level credit assignment. The skill contract shapes the trajectory return, but neither Verified Progress Credit nor feedback guarantees a nonzero expected parameter gradient, convergence, or improved Task Outcome.

Task instantiation. Math and Search tasks use identical contract fields and optimizer specifications, while Appendix Table 6 gives their task-specific actions, observations, and completion checks. The runtime maintains task-specific bookkeeping used by Step_j (e.g., tool budget tracking and duplicate action checks), detailed in Appendix A.1. For Math, Step_j maps a code action executing Python to an interpreter observation and requires reasoning over that result before taking another action or producing the final answer, following program-aided and tool-integrated reasoning paradigms (Gao et al., 2023; Chen et al., 2022; Gou et al., 2024b). For Search, Step_j maps the initial `<search>` and subsequent `<research>` actions to retrieval observations, following multi-turn retrieval reasoning (Jin et al., 2025b). Search evidence processing may also include an optional `<summary>` phase that organizes existing evidence without calling the retriever or producing a new observation. After the summary, the policy may retrieve additional evidence or provide a final answer, motivated by prior work on evidence refinement and context summarization (Shi et al., 2026b; Wu et al., 2025). In both tasks, a trajectory achieves *Protocol Completion* when it satisfies all required phases $\mathcal{Q}_j$ and reaches accepted termination. Each skill contract is written manually from the corresponding expert-authored skill; automatic conversion of arbitrary skill text is outside the present scope.

4 Experiments

4.1 Experimental Setup

Tasks and evaluation. We train on MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), Natural Questions (Kwiatkowski et al., 2019), and HotpotQA (Yang et al., 2018). Evaluation additionally includes TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2023), 2WikiMultiHopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). Math uses pass@1 after numerical normalization, while Search uses normalized exact match. Domain averages are computed uniformly across datasets within each domain.

We report Task Outcome, PCR, and RF-PCR for skill-contract conditions. Task Outcome measures answer correctness, using accepted answers for contract-trained policies and the baseline extractors described in Appendix C. For N evaluated trajectories, let T_i denote the number of model actions in trajectory i, q_{i,T_i}^j its final phase, and $h_{i,t}$ its recovery flag before action t. The contract enters `DONE` only when a trajectory completes all required phases and reaches accepted termination, so Protocol Completion Rate (PCR) is

$$\text{PCR} = \frac{100}{N} \sum_{i=1}^{N} \mathbf{1}\left[q_{i,T_i}^j = \texttt{DONE}\right]. \tag{9}$$

Under the runtime with feedback enabled, Recovery-free PCR (RF-PCR) additionally requires that the recovery flag never become one:

$$\text{RF-PCR} = \frac{100}{N} \sum_{i=1}^{N} \mathbf{1}\left[q_{i,T_i}^j = \texttt{DONE}\right] \mathbf{1}\left[\sum_{t=0}^{T_i-1} h_{i,t} = 0\right]. \tag{10}$$

Thus, RF-PCR cannot exceed PCR and is distinct from disabling feedback at inference. Protocol Completion does not guarantee a correct answer, so these measures remain separate. Appendix C provides dataset sizes and complete evaluation rules.

Table 1: Task Outcome, PCR, and RF-PCR on Math and Search (%). RF-PCR excludes samples that receive rejected-action feedback. Direct evaluates frozen checkpoints without the contract runtime, while Prompt preloads the skill and uses the runtime. Recipe pairs task-specific adaptations, Ours-S trains each domain separately, Ours-J trains them jointly, and *n/a* denotes an absent protocol metric. Bold marks the highest value in each metric column within each model block, including ties. Averages are unweighted, while ^{tr} and ^{ho} mark training and held-out datasets.

	Math					Search									
Setting	MATH^{tr}	GSM^{tr}	Avg.	PCR	RF-PCR	NQ^{tr}	TQA^{ho}	Pop^{ho}	HQA^{tr}	2W^{ho}	MuS^{ho}	Bam^{ho}	Avg.	PCR	RF-PCR
Qwen2.5-3B															
Direct	17.00	26.08	21.54	*n/a*	*n/a*	2.27	6.31	3.11	2.84	5.16	0.12	1.60	3.06	*n/a*	*n/a*
Prompt	0.16	0.00	0.08	0.16	0.00	0.08	0.14	0.14	0.09	0.25	0.00	0.00	0.10	0.20	0.00
Recipe	45.16	**81.20**	63.18	*n/a*	*n/a*	45.26	**61.73**	44.00	**32.14**	29.21	6.37	10.40	32.73	*n/a*	*n/a*
Ours-S	**48.62**	80.59	**64.61**	**97.29**	**94.54**	**46.34**	60.53	**45.67**	30.34	**29.87**	**6.91**	16.00	**33.67**	**99.82**	**98.54**
Ours-J	45.02	79.22	62.12	94.46	93.78	43.23	58.23	43.96	29.09	28.53	5.13	**18.40**	32.37	98.46	97.00
Qwen3.5-4B-Base															
Direct	70.60	88.93	79.77	*n/a*	*n/a*	12.96	35.18	12.87	15.65	22.65	2.69	20.80	17.54	*n/a*	*n/a*
Prompt	2.16	4.32	3.24	3.84	0.00	0.89	5.41	1.92	1.80	1.33	0.21	3.20	2.11	7.32	0.00
Recipe	**78.28**	**95.22**	**86.75**	*n/a*	*n/a*	48.59	65.34	47.06	38.46	35.97	12.78	31.20	39.91	*n/a*	*n/a*
Ours-S	72.02	91.81	81.92	**98.92**	96.12	**49.39**	**65.75**	**48.28**	**47.13**	**46.91**	**20.19**	**44.80**	**46.06**	99.76	**99.11**
Ours-J	71.86	93.40	82.63	98.73	**96.31**	48.42	63.18	45.92	41.36	36.80	14.52	38.40	41.23	**99.99**	95.37
Qwen2.5-3B-Instruct															
Direct	48.26	77.79	63.02	*n/a*	*n/a*	8.86	21.52	8.71	13.49	22.49	1.32	3.20	11.37	*n/a*	*n/a*
Prompt	2.72	4.09	3.41	4.83	0.39	0.17	0.53	0.29	0.18	0.14	0.04	0.00	0.19	75.07	33.28
Recipe	47.16	**83.40**	65.28	*n/a*	*n/a*	**45.10**	**61.43**	**44.24**	**32.06**	27.72	**7.41**	**14.40**	**33.19**	*n/a*	*n/a*
Ours-S	**51.24**	80.29	**65.76**	**98.83**	**96.54**	44.99	60.62	43.94	31.74	**28.58**	7.03	**14.40**	33.04	99.92	**99.69**
Ours-J	45.34	80.29	62.81	96.72	91.12	42.99	59.78	43.22	29.95	27.77	6.37	9.60	31.38	**99.97**	99.41
Qwen3.5-4B															
Direct	38.78	72.10	55.44	*n/a*	*n/a*	11.27	35.01	11.54	7.87	2.32	0.99	17.60	12.37	*n/a*	*n/a*
Prompt	0.38	1.29	0.83	0.93	0.00	0.47	6.51	1.55	1.90	1.23	0.29	2.40	2.05	5.71	0.00
Recipe	75.44	**94.92**	85.18	*n/a*	*n/a*	48.56	65.22	**49.98**	37.14	40.86	19.97	23.20	40.70	*n/a*	*n/a*
Ours-S	**78.48**	92.72	**85.60**	98.25	**96.27**	48.59	64.49	47.36	**47.05**	**46.10**	19.03	41.60	44.89	99.50	**96.58**
Ours-J	75.76	90.14	82.95	**99.27**	92.90	**50.86**	**66.15**	47.56	46.87	45.01	**22.63**	**47.20**	**46.61**	**99.96**	95.40
Qwen2.5-7B															
Direct	15.70	23.96	19.83	*n/a*	*n/a*	5.48	12.72	4.35	5.51	9.05	1.41	5.60	6.30	*n/a*	*n/a*
Prompt	2.06	4.09	3.08	5.17	1.28	0.75	1.78	0.95	0.55	1.30	0.12	1.60	1.01	12.36	3.22
Recipe	**58.96**	86.42	72.69	*n/a*	*n/a*	45.83	**66.38**	45.82	**37.14**	32.29	8.13	24.00	**37.08**	*n/a*	*n/a*
Ours-S	58.72	**86.73**	**72.73**	**96.08**	**95.50**	43.63	64.28	**46.93**	35.56	**33.19**	10.05	**25.60**	37.03	**98.28**	**98.27**
Ours-J	56.96	83.40	70.18	96.00	95.40	**47.81**	59.01	42.43	34.21	30.44	**16.79**	24.00	36.38	97.00	96.67

Models. Qwen2.5-3B (Yang et al., 2024) is the primary base executor for method analysis and controlled studies. Qwen3.5-4B-Base tests transfer to a newer base initialization (Qwen Team, 2026). Qwen2.5-3B-Instruct and Qwen3.5-4B test the same skill-contract design after instruction or general post-training. Qwen2.5-7B provides a same-family scale extension.

Comparisons and training. Direct and Prompt evaluate frozen executors without and with the contract runtime. Recipe denotes matched ReTool-style and Search-R1 adaptations (Feng et al., 2026; Jin et al., 2025b), Ours-S denotes separate O-Math and O-Search policies, and Ours-J denotes joint training. All online conditions use GRPO, and controlled studies preserve the data, task adapter, contract verifier, and task reward while changing only the named component. Search comparisons share one retriever and corpus (Jin et al., 2025a), while Appendix C records baseline provenance and Appendix Table 7 gives the complete settings.

Research questions. The analysis first establishes grounded skill-following, then examines training behavior and inference behavior.

1. **RQ1: Grounded skill-following.** Does skill-contract training produce high Protocol Completion and useful Task Outcomes across domains and executors?
2. **RQ2: Training behavior.** How do verified progress credit, contract-state feedback, and the recovery coefficient affect learning?
3. **RQ3: Inference behavior.** How does execution depend on visible skill instructions, observation content, and contract-state feedback?

4.2 RQ1: Grounded Skill-Following Results

Contract training yields high Protocol Completion with mixed Task Outcome gains. Across separate and joint policies, Math PCR ranges from 94.46% to 99.27%, while Search PCR ranges from 97.00% to 99.99% (Table 1). For Qwen3.5-4B, joint training achieves Math/Search Task Outcomes of 82.95%/46.61%, PCR of 99.27%/99.96%, and RF-PCR of 92.90%/95.40%. The gap between PCR and RF-PCR is configuration-dependent: for example, Qwen3.5-4B joint training reaches 99.27% Math PCR but 92.90% RF-PCR, whereas Qwen2.5-7B separate Math training reaches 96.08% PCR and 95.50% RF-PCR. Task Outcome does not uniformly improve over Recipe, nor does joint training consistently outperform separate training. Appendix D provides complete execution traces.

4.3 RQ2: Training Behavior

Training signals. Verified progress credit drives learning, with task-dependent support from feedback. Credit-only reaches Math/Search Task Outcomes of 59.76%/29.28% and PCR of 93.17%/98.69%, while Feedback + Credit reaches 64.61%/30.11% and 97.29%/98.90%, respectively (Figure 4(a), Appendix Table 8). Outcome-only and Feedback-only remain at zero in these conditions. For the matched learning curves, Outcome-only and Credit-only use the same contract runtime with contract-state feedback disabled. Outcome-only uses the task reward alone, while Credit-only adds verified progress credit to the trajectory return. Credit-only develops nonzero Task Outcome and PCR while Outcome-only remains at zero under this contract runtime (Figure 3).

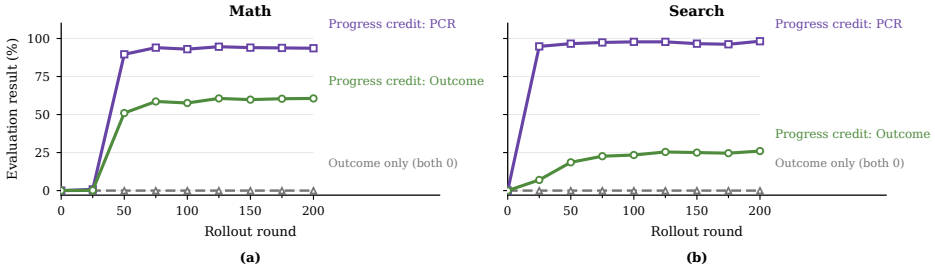

Figure 3: Qwen2.5-3B learning dynamics under the same contract runtime with feedback disabled. Each point averages a fixed monitor of 250 examples per dataset: (a) MATH and GSM8K, (b) Natural Questions and MuSiQue. Verified progress credit yields nonzero PCR and Task Outcome in both domains, while Outcome-only remains at zero.

Recovery coefficient. The recovery coefficient of 0.90 yields higher RF-PCR in the tested joint conditions. With $\rho = 0.90$, Math/Search RF-PCR is 93.78%/98.18%, compared with 0.00%/88.33% for $\rho = 1.00$ (Table 2, Panel A). The $\rho = 1.00$ condition has higher PCR and Task Outcome, with Math/Search Task Outcome higher by 2.49/0.70 points. These independently trained conditions show differences in recovery-free Protocol Completion, without establishing a consistent advantage across metrics or training seeds.

Table 2: Training- and inference-time recovery interventions. Panel A compares independently trained Qwen2.5-3B joint conditions with $\rho = 0.90$ and $\rho = 1.00$. Panel B removes feedback at inference from frozen Qwen2.5-3B single-task checkpoints. Both panels report unweighted averages over MATH and GSM8K for Math and 2WikiMultiHopQA and HotpotQA for Search. Values are percentages, and changes are percentage points.

Domain	A: Recovery coefficient				B: Feedback removal				
	ρ	Task Outcome	PCR	RF-PCR	Task Outcome			PCR	
					On	Off	Change	On	Off
Math	0.90	62.12	94.46	93.78	64.61	58.25	−6.36	97.29	86.93
	1.00	64.61	96.23	0.00					
Search	0.90	28.81	98.90	98.18	30.11	13.15	−16.96	98.90	43.05
	1.00	29.51	99.53	88.33					

4.4 RQ3: Inference Behavior

We intervene on frozen Qwen2.5-3B single-task policies evaluated on MATH, GSM8K, 2WikiMultiHopQA, and HotpotQA. Full intervention conditions are reported in Appendix E.

Visible skill instructions. **Execution depends on visible skill instructions.** Removing them lowers Math/Search Task Outcome by 5.89/9.57 points, while paraphrasing preserves most performance (Figure 4(b), Appendix Table 9). With the rest of $\mathcal{C}_j$ fixed, this tests instruction dependence within the runtime. Wrong-skill substitution also changes the action grammar, limiting its interpretation to compatibility.

Observation and feedback dependence. **Observation content supports Task Outcome, while feedback dependence is stronger on Search.** Withholding observation content lowers Task Outcome by 13.62, 11.22, 4.88, and 13.20 points on MATH, GSM8K, 2Wiki, and HotpotQA, respectively (Figure 4(c)). Removing feedback lowers Math/Search Task Outcome by 6.36/16.96 points (Table 2, Panel B). PCR falls from 97.29% to 86.93% on Math and from 98.90% to 43.05% on Search. These inference interventions establish behavioral dependence, without isolating token-level evidence use or feedback's effect during training.

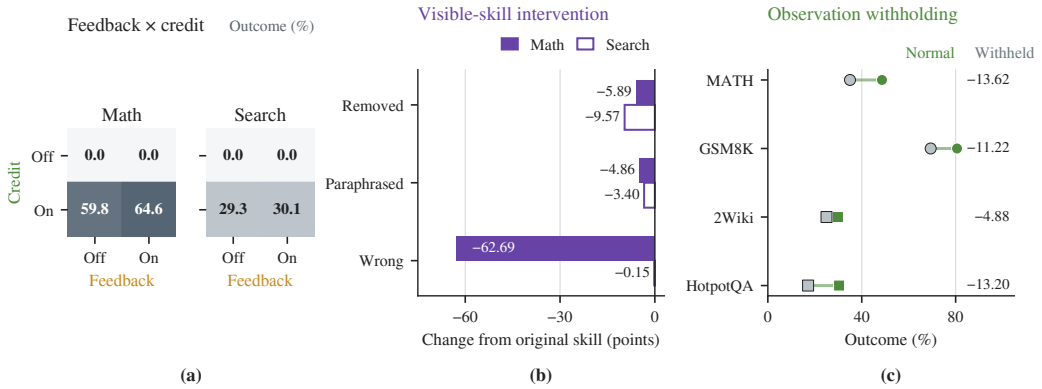

Figure 4: Qwen2.5-3B interventions on (a) contract-state feedback and verified progress credit during training, (b) visible skill instructions at inference, and (c) observation content at inference. Values are percentages or percentage-point changes.

4.5 Empirical Insights

High Protocol Completion does not imply improved Task Outcome. Trained executors achieve high PCR, but their Task Outcome gains over matched Recipe vary (Section 4.2). **High Protocol Completion can coexist with different execution dependencies.** Math is more sensitive to cross-task skill substitution, whereas Search is more sensitive to feedback removal, although substitution also changes the action grammar (Section 4.4). **Small training-time gains do not imply feedback independence.** On Search, adding feedback during training yields a small Task Outcome increment, yet removing it at inference substantially lowers Task Outcome and PCR (Sections 4.3 and 4.4). Feedback can guide subsequent attempts or output refinement, as also explored in Reflexion and CRITIC (Shinn et al., 2023; Gou et al., 2024a).

5 Conclusion

We define grounded skill-following as learning to execute a fixed expert-authored skill through its required phases and to use environment observations to complete the task. Skill contracts make procedural progress verifiable during post-training and evaluation. Math and Search experiments report PCR, RF-PCR, and Task Outcome separately, while controlled learning curves show nonzero Protocol Completion and Task Outcome with verified progress credit under a contract runtime where outcome-only training remains at zero. Reliance on visible skill instructions and contract-state feedback varies by task, and the findings remain limited to manually specified contracts and the tested executors (Appendix F).

AI USE STATEMENT

We used generative AI tools to assist with writing and polishing, including drafting and revising sections of this manuscript. AI tools also supported literature retrieval and discovery, including identifying relevant prior work. We used these tools for research ideation and experimental execution, as well as assistance in developing proofs of mathematical claims. The authors take full responsibility for the final content, including the references, mathematical claims and proofs, experimental results, analyses, and text.

REPRODUCIBILITY STATEMENT

Appendix A specifies the contract runtime, transition logic, milestone definitions, and reward construction. Appendix B provides the exact model-visible skill interfaces. Appendix C reports training configurations, optimization hyperparameters, rollout budgets, retrieval settings, dataset sizes, and evaluation procedures. Appendix D provides complete representative execution traces, and Appendix E specifies the controlled intervention settings and results. The code and implementation artifacts will be made publicly available after consolidation and documentation.

REFERENCES

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Yan-Lun Chen, Yi-Ru Wei, Chia-Yi Hsu, Chi-Yu Li, Chun-Ying Huang, Ying-Dar Lin, Yu-Sung Wu, and Wei-Bin Lee. Layer-aware task arithmetic: Disentangling task-specific and instruction-following knowledge. In *EMNLP (Findings)*, pp. 12033–12054, 2025.

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

Chenhao Dang, Siyuan Xiong, Conghui He, and Weijia Li. Skiller: Language-level reinforcement learning for reusable skill extraction in small language models. *arXiv preprint arXiv:2608.10538*, 2026.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. In *International Conference on Learning Representations*, volume 2026, pp. 37909–37926, 2026.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. 2024:57734–57811, 2024a.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. 2024:48362–48395, 2024b.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Zining Huang, Haoran Que, Hong Zeng, Ge Zhang, Zuo Wang, Jin Chen, Haodong Wang, Zhongfei Hou, Changxin Pu, Shen Yan, et al. Harness-if: Evaluating instruction following across instruction surfaces in coding agents. *arXiv preprint arXiv:2608.11727*, 2026.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pp. 2107–2116. PMLR, 2018.

Bowen Jin, Jinsung Yoon, Priyanka Kargupta, Sercan O Arik, and Jiawei Han. An empirical study on reinforcement learning for reasoning-search interleaved llm agents. *arXiv preprint arXiv:2505.15117*, 2025a.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025b.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Hongxiang Lin, Zhirui Kuai, Erpeng Xue, and Lei Wang. Skillc: Learning autonomous skill internalization in llm agents via contrastive credit assignment. *arXiv preprint arXiv:2605.27899*, 2026.

Hongjun Liu, Yifei Ming, Shafiq Joty, and Chen Zhao. Harnessing llm agents with skill programs. *arXiv preprint arXiv:2605.17734*, 2026.

Siyuan Lu, Zechuan Wang, Hongxuan Zhang, Qintong Wu, Leilei Gan, Chenyi Zhuang, Jinjie Gu, and Tao Lin. Don't just fine-tune the agent, tune the environment. In *International Conference on Learning Representations*, volume 2026, pp. 6237–6283, 2026a.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026b.

SHengjie Ma, Chenlong Deng, Jiaxin Mao, Jiadeng Huang, Teng Wang, Junjie Wu, Changwang Zhang, et al. Proof-of-use: Mitigating tool-call hacking in deep research agents. *arXiv preprint arXiv:2510.10931*, 2025.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 9802–9822, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. volume 37, pp. 55249–55285, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026a.

Yaorui Shi, Sihang Li, Chang Wu, Zhiyuan Liu, Junfeng Fang, Hengxing Cai, An Zhang, and Xiang Wang. Search and refine during think: Facilitating knowledge refinement for improved retrieval-augmented reasoning. *Advances in Neural Information Processing Systems*, 38:155930–155958, 2026b.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. Skill retrieval augmentation for agentic ai. *arXiv preprint arXiv:2604.24594*, 2026.

Zelin Tan, Yiqun Zhang, Hao Li, Zhiyao Cui, Hejia Geng, Shao Zhang, Hangfan Zhang, Yang Chen, Xiaosong Wang, Lilong Wang, et al. Skt: Skill-use training at scale via verified synthetic data generation. *arXiv preprint arXiv:2608.02287*, 2026.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. Syncode: Llm generation with grammar augmentation. *arXiv preprint arXiv:2403.01632*, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, et al. Skill-sd: Skill-conditioned self-distillation for multi-turn llm agents. *arXiv preprint arXiv:2604.10674*, 2026a.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550, 2026b.

Xiaozhi Wang, Kaiyue Wen, Zhengyan Zhang, Lei Hou, Zhiyuan Liu, and Juanzi Li. Finding skill neurons in pre-trained transformer-based language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11132–11152, 2022.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, et al. Benchmarking complex instruction-following with multiple constraints composition. *Advances in Neural Information Processing Systems*, 37:137610–137645, 2024.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, et al. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

An Yang et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, Yang Katie Zhao, and Mingyi Hong. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. In *ICML 2025 Workshop on Computer Use Agents*, 2025.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills, 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Qi Zhang, Zhaopeng Feng, Xiaonan Shi, Xiaomeng Hu, Chu Liu, Pengjun Xie, Xiaobin Wang, Jieping Ye, Bryan Hooi, Haobo Wang, et al. Skillcomposer: Learning to evolve agent skills for specification and generalization. *arXiv preprint arXiv:2606.06079*, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Xiaolin Zhou, Jinbo Liu, Li Li, Ryan A Rossi, and Xiyang Hu. Counterfactual trace auditing of llm agent skills. *arXiv preprint arXiv:2605.11946*, 2026.

Jiayin Zhu, Kelong Mao, Yudong Guo, Dengbo He, Sulong Xu, Simiu Gu, and Yutao Yue. Skill-coach: Self-evolving rubrics for evaluating and enhancing agentic skill-use. *arXiv preprint arXiv:2607.01874*, 2026.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A SKILL CONTRACT RUNTIME AND LEARNING SIGNAL

A.1 CANONICAL CONTRACT RUNTIME

The contract runtime maintains the contract state $S_t^j = (q_t^j, \mathcal{M}_t^j, h_t)$, updating the current phase q_t^j and completed milestones $\mathcal{M}_t^j$ alongside environment observations o_{t+1} and contract-state feedback f_{t+1} after each action. The symbol $\perp$ denotes the absence of an environment observation. The executor receives feedback regarding the contract phase rather than internal bookkeeping details. The recovery flag h_t is initialized to zero and is set to 1 upon receiving feedback from a rejected action. When an action is accepted, any newly verified milestones are recorded using the pre-action recovery flag h_t, after which h_{t+1} is reset to 0 (even if no new milestone is completed). When contract-state feedback is disabled, the recovery flag remains zero throughout the rollout. The runtime also tracks task-specific bookkeeping omitted from Equation 4. Observation validity is initially false and is updated after each environment call to evaluate phase transitions following evidence reasoning. Environment-action counts enforce tool-call budgets and distinguish initial from subsequent retrieval actions; the most recent query prevents duplicate retries, and a record of evidence organization prevents redundant summary phases.

Table 3: Phase set $\mathcal{Q}_j$ shared by Math and Search. A rejected action preserves the current phase and returns one admissible next action. Table 6 defines substantive thought.

Phase	Accepted action	Next phase
`NEED_SKILL`	Expected skill call	`NEED_PLAN`
`NEED_PLAN`	Substantive planning thought	`NEED_ACTION`
`NEED_ACTION`	Admissible environment action	`NEED_EVIDENCE_THINK`
`NEED_EVIDENCE_THINK`	Substantive evidence reasoning	`CAN_CONTINUE_OR_ANSWER` after a valid observation, otherwise `NEED_ACTION`
`CAN_CONTINUE_OR_ANSWER`	Repeated environment action, first optional Search evidence-organization action with `<summary>`, or final answer	`NEED_EVIDENCE_THINK`, `CAN_CONTINUE_OR_ANSWER`, or `DONE`, respectively. A repeated evidence-organization action is rejected
`DONE`	None	Terminal

Table 4: Runtime execution order for one skill-contract trajectory.

Order	Operation	Update
1	Initialize	Set initial contract state $S_0^j = (\texttt{NEED_SKILL}, \varnothing, 0)$. Initialize context x_0^j with system scaffold and task prompt (Appendix B.1).
2	Generate	Sample action $a_t \sim \pi_\theta(\cdot \mid x_t^j)$.
3	Transition	Execute state transition via Equation 4. Set $o_{t+1} = \perp$ when no observation is generated. Action rejection preserves q_t^j and $\mathcal{M}_t^j$, setting $h_{t+1} = 1$ when feedback is enabled. Action acceptance records new milestones scaled by ρ^{h_t}, then resets $h_{t+1} = 0$.
4	Record	Record newly verified milestones $\Delta\mathcal{M}_{i,t}^j$ and update context via Equation 5, omitting $\perp$ and prepending skill instructions z_j upon first accepted skill call.
5	Repeat	Repeat Order 2–4 until accepted termination or budget exhaustion. Exhaustion does not constitute Protocol Completion.
6	Score	Compute final trajectory return R_i using Equation 7.

Each model turn is truncated after its first complete leading action. The environment responds in a separate loss-masked turn. A rejected action remains trainable, whereas visible skill instructions, contract-state feedback, and environment observations are assigned a loss mask of zero. An inadmissible action does not advance the phase or milestone set, terminate the rollout, or impose a negative penalty. The executor may continue generating actions until accepted termination or budget depletion. Training on full interaction trajectories is motivated by research on learning to recover

from unsuccessful attempts (Qu et al., 2024). Our evaluation assesses this trajectory-level formulation without isolating the individual gradient contributions of rejected-action tokens.

Validity of the accepted procedure. Assume that the initial contract state $S_0^j = (q_0^j, \mathcal{M}_0^j, h_0)$ satisfies the contract specification and that the contract verifier accepts an action a_t only when its induced transition $S_t^j \to S_{t+1}^j$ is valid. Because action rejection leaves the current phase q_t^j and completed milestones $\mathcal{M}_t^j$ unchanged, it trivially preserves state validity. Each accepted action preserves validity by definition, including optional Search evidence-organization transitions that maintain the current phase. By mathematical induction over contract transitions, any accepted sequence of transitions strictly satisfies the procedure specified by the contract along its traversed phases. When accepted termination requires satisfying all phases in $\mathcal{Q}_j$, Protocol Completion Rate (PCR) precisely measures this structural property.

This property guarantees structural adherence under valid verifier logic; it does not guarantee that the executor will reach termination. It neither validates raw rejected actions nor guarantees answer correctness or meaningful semantic processing of observations. Furthermore, removing rejected actions and replaying the remaining action sequence does not reproduce the original trajectory, as rejections directly alter contract-state feedback f_{t+1} and recovery flag bookkeeping h_t.

A.2 Feedback, Milestones, and Task Adapters

Each contract-state feedback message f_{t+1} explicitly specifies action admissibility, the current phase, and an admissible next action. It never exposes the ground-truth task answer. The contract runtime prepends a status prefix to a phase-specific message body, wrapping the combined message in `<hint>...</hint>`.

```
NEED_SKILL:
State: no skill is loaded. Next: select one listed skill with <skill_call
    >...</skill_call>.

NEED_PLAN:
State: the skill is loaded. Next: write one complete <think>...</think>
    plan.

NEED_ACTION, Math:
State: Python evidence is required. Next: run calculation in <code>...</
    code> and print a result.

NEED_ACTION, first Search action:
State: retrieval evidence is required. Next: use a concise <search>...</
    search> query.

NEED_ACTION, later Search action:
State: retrieval evidence is required. Next: use a concise <research
    >...</research> query.

NEED_EVIDENCE_THINK:
State: an environment observation is ready. Next: write one complete <
    think>...</think>.

CAN_CONTINUE_OR_ANSWER, Math:
State: evidence is analyzed. Next: use <code>...</code> again or give
    Answer: \boxed{...}.

CAN_CONTINUE_OR_ANSWER, Search before summary:
State: evidence is analyzed. Next: use <research>...</research> again,
    organize it with <summary>...</summary>, or give <answer>...</answer>.


CAN_CONTINUE_OR_ANSWER, Search after summary:
State: evidence is summarized. Next: use <research>...</research> again
    or give <answer>...</answer>.
```

The status prefixes are `Action accepted.` and `Action rejected.`. An accepted final answer terminates the rollout immediately without appended feedback. In ablation studies where contract-state feedback is disabled, f_{t+1} is omitted after both accepted and rejected actions while contract validation logic remains fully active.

For Math, a valid observation o_{t+1} requires successful Python code execution with non-empty stdout. For Search, a valid observation requires at least one non-empty retrieved document. An admissible environment action transitions to evidence reasoning (`NEED_EVIDENCE_THINK`) even if the returned observation is invalid. Following evidence reasoning, the task adapter permits state transition to `CAN_CONTINUE_OR_ANSWER` only if the preceding observation was valid; otherwise, it requires an additional environment action. These checks verify structural non-emptiness rather than semantic relevance or sufficiency.

Search additionally permits one optional evidence-organization action following evidence reasoning. This action uses `<summary>`, preserves the `CAN_CONTINUE_OR_ANSWER` phase, generates no environment observation ($o_{t+1} = \perp$), and earns verified progress credit only upon its first acceptance. Its content check enforces a minimal length requirement of at least 24 normalized characters and four word tokens without embedded XML tags, preventing empty or trivial summaries.

Table 5: Milestone set $\mathcal{M}_j$ and first-completion weights used for verified progress credit. Each contract location maps the credit event to the shared phase or transition in Table 3. The task reward is a separate binary term. Table 6 defines substantive thought. For Search, evidence organization through `<summary>` is optional for accepted termination. Its verified progress credit encourages this optional action, so permitted trajectories need not receive equal progress credit.

Milestone	Scope	Weight	Contract location	First-completion condition
Skill	Both	0.10	`NEED_SKILL`	The expected `<skill_call>` is accepted.
Plan	Both	0.10	`NEED_PLAN`	A substantive `<think>` is accepted.
Action	Both	0.20	`NEED_ACTION`	The first admissible `<code>`, `<search>`, or `<research>` passes its task adapter.
Observation	Both	0.20	`NEED_ACTION` → `NEED_EVIDENCE_THINK`	Math execution produces nonempty output, or Search returns at least one nonempty document.
Evidence reasoning	O-Math	0.20	`NEED_EVIDENCE_THINK`	A substantive `<think>` is accepted.
Evidence reasoning	O-Search	0.10	`NEED_EVIDENCE_THINK`	A substantive `<think>` is accepted.
Evidence organization	O-Search	0.10	`CAN_CONTINUE_OR_ANSWER` → same phase	The first optional `<summary>` passes the content checks after evidence reasoning without calling the environment.
Answer	Both	0.20	`CAN_CONTINUE_OR_ANSWER` → `DONE`	A nonempty boxed Math answer or tagged Search answer is accepted.

At trajectory end, each milestone contributes at most once to R_i. A milestone first completed by the next accepted transition after rejected-action feedback contributes ρw_m. Otherwise, it contributes w_m.

The main runs use $\rho = 0.90$. The registered no-discount comparison independently trains a condition with $\rho = 1.00$. The coefficient does not modify earlier progress or the independent task reward.

O-Search retains separate 0.10 first-completion credits for evidence reasoning and optional evidence organization through `<summary>`.

A.3 Motivation for Group Return Variance under Sparse Rewards

For a given task prompt, let $R_1, \ldots, R_G$ be the trajectory returns of G sampled rollouts, with empirical mean $\bar{R}$ and standard deviation s_R. Let $r_{\text{task},i}$ denote the sparse task-completion reward and p_i denote the total recovery-adjusted progress credit defined in Equation 7. The trajectory return decomposes as:

$$R_i = r_{\text{task},i} + p_i. \tag{11}$$

When all rollouts within a sampled group fail to solve the task (or all succeed), the task reward $r_{\text{task},i}$ is uniform across the group. In this all-zero (or all-one) reward regime, the pairwise return difference

reduces solely to differences in verified progress credit:

$$R_i - R_k = p_i - p_k. \tag{12}$$

If at least two trajectories have different recovery-adjusted progress-credit totals $p_i \neq p_k$, their returns differ and yield a non-zero empirical variance $s_R > 0$. This can occur when trajectories complete different milestones or incur different recovery discounts, but different milestone sets alone do not guarantee distinct totals. This allows the normalized advantages in Equation 2 to provide meaningful learning signals for step-by-step progress, even before correct final answers become frequent. Figure 3 illustrates training dynamics under these reward formulations.

Conversely, if every trajectory receives an identical progress credit c, group mean centering cancels this additive offset, where $\bar{r}_{\text{task}}$ denotes the group mean task reward:

$$R_i = r_{\text{task},i} + c \quad \Longrightarrow \quad R_i - \bar{R} = r_{\text{task},i} - \bar{r}_{\text{task}}. \tag{13}$$

Because standard deviation remains invariant under constant shifts ($s_{R+c} = s_R$), a uniform progress credit leaves normalized GRPO advantages completely unchanged.

Table 6: Task-specific instantiation of Step_j by phase. Table 3 defines the shared transitions, and Table 5 locates verified progress credit. For <think>, "substantive" requires at least 12 characters after whitespace normalization, including at least 6 alphanumeric characters. These are form-based checks and do not assess reasoning quality.

Phase	O-Math	O-Search
NEED_SKILL	Accept the expected code-interpreter-protocol skill call.	Accept the expected qa-search-protocol skill call.
NEED_PLAN	Accept a substantive <think> that plans the calculation.	Accept a substantive <think> that plans retrieval.
NEED_ACTION	Accept executable Python that prints a result in <code>...</code>. The restricted interpreter returns <interpreter>. The observation is valid when execution succeeds and printed output is nonempty. The accepted action enters NEED_EVIDENCE_THINK.	Accept the first admissible query in <search> and later nonrepeated admissible queries in <research>. The frozen retriever returns <information>. The observation is valid when at least one retrieved document is nonempty. The accepted action enters NEED_EVIDENCE_THINK.
NEED_EVIDENCE_THINK	Accept a substantive <think> that interprets the interpreter result. A valid observation opens continuation or answering. An invalid one returns to NEED_ACTION.	Accept a substantive <think> that assesses the retrieved evidence. A valid observation opens continuation or answering. An invalid one returns to NEED_ACTION.
CAN_CONTINUE_OR_ANSWER	Repeat <code>...</code> and re-enter evidence reasoning, or accept a nonempty boxed answer and enter DONE.	Repeat retrieval with <research> and re-enter evidence reasoning. Optionally accept one substantive <summary> as evidence organization, with no retriever call, no observation, and no phase change. Alternatively, accept a nonempty <answer> and enter DONE.
DONE	Terminal. Score the accepted answer by normalized numerical match.	Terminal. Score the accepted answer by dataset-normalized exact match.

Shift and Scale Invariance in GRPO. Let $\mathbf{R} = [R_1, \ldots, R_G]^\top$ denote the group return vector, b a common additive shift, and $a > 0$ a positive scale factor. Under the GRPO normalization in Equation 2, additive shifts are strictly invariant:

$$\widehat{A}_i(\mathbf{R} + b) = \widehat{A}_i(\mathbf{R}). \tag{14}$$

For positive scaling $a > 0$, the normalized advantage transforms as:

$$\widehat{A}_i(a\mathbf{R}) = \frac{a(R_i - \bar{R})}{as_R + \epsilon} = \frac{R_i - \bar{R}}{s_R + \epsilon/a}, \qquad a > 0. \tag{15}$$

These identities confirm that mean-centering eliminates additive offsets, while standard deviation scales linearly by a. In the absence of the numerical stabilizer $\epsilon \to 0^+$, exact scale invariance holds whenever $s_R > 0$. With a non-zero stabilizer $\epsilon > 0$, Equation 15 precisely characterizes the advantage adjustment when milestone rewards scale the return distribution.

When a group contains distinguishable returns. Consider a task prompt and a fixed policy generating independent, identically distributed trajectories. Assume a sampled group contains $G \geq 2$ trajectories with exactly two possible distinct returns, where each trajectory achieves the higher return with probability $p \in (0, 1)$. The empirical group variance s_R^2 is strictly positive if and only if both return values are present in the group:

$$\Pr(s_R > 0) = 1 - p^G - (1-p)^G, \qquad G \geq 2. \tag{16}$$

This holds because zero variance occurs exclusively in all-high (p^G) or all-low ($(1-p)^G$) groups. The probability of non-zero within-group variance reaches its maximum at an equal mixture ($p = 0.5$) and vanishes as $p \to 0$ or $p \to 1$. Consequently, increasing the success rate does not monotonically increase within-group return variation. This two-return model illustrates a sampling constraint on gradient existence under sparse rewards rather than full multi-stage training dynamics.

From return differences to gradients. Now consider an unclipped expected-return objective under a fixed contract specification, task adapter, and environment. Assuming a differentiable policy π_θ with parameter-independent support and sufficient regularity to swap differentiation and integration, the score-function gradient identity yields:

$$\nabla_\theta \mathbb{E}_{\tau \sim P_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim P_\theta}[(R(\tau) - \mathbb{E}[R(\tau)]) \nabla_\theta \log P_\theta(\tau)]\,. \tag{17}$$

Because the score function has zero expectation ($\mathbb{E}[\nabla_\theta \log P_\theta(\tau)] = 0$), subtracting the mean return leaves the overall gradient unchanged. Since environment state transition dynamics are parameter-independent, only model-generated action tokens contribute directly to the trajectory score:

$$\nabla_\theta \log P_\theta(\tau) = \sum_t \nabla_\theta \log \pi_\theta(a_t \mid x_t^j). \tag{18}$$

Non-zero return variance alone is insufficient to guarantee non-zero policy gradients; the trajectory return must also exhibit non-zero covariance with the trajectory score $\nabla_\theta \log P_\theta(\tau)$. Loss-masked feedback modifies subsequent prompt contexts x_t^j and action generation distributions without receiving direct gradient updates. Note that Equation 17 serves as an idealized theoretical baseline; the practical implementation differs via GRPO group normalization, token-level averaging, PPO clipping, and KL regularization.

Design rationale. Initial milestone completions increment the recovery-adjusted progress credit p_i and provide return variance when rollouts accumulate different recovery-adjusted totals. Following each non-terminal transition $S_t^j \to S_{t+1}^j$, contract-state feedback f_{t+1} updates the context x_{t+1}^j for the subsequent decision step. While feedback text carries no direct reward weight, action rejection reduces future verified progress credit via the recovery coefficient ρ. When contract-state feedback is enabled, accepted non-terminal transitions receive positive phase guidance, whereas rejected transitions trigger recovery feedback and set $h_{t+1} = 1$, so milestones first completed on the next accepted transition are discounted by ρ.

Deriving both feedback f_{t+1} and credit p_i from the unified contract verifier guarantees state-consistency between requested next actions and rewarded progress. This structural coupling ensures implementation consistency without asserting equal necessity of feedback and credit. This motivation specifically addresses the normalized advantage $\widehat{A}_i$ in GRPO; feedback modifies prompt contexts rather than directly modifying advantage estimates. This derivation characterizes learning signal availability without claiming convergence bounds or guaranteed policy improvement.

B Skill Contracts and Visible Skill Instructions

B.1 Exact Visible Skill Interface

In the canonical skill-loading conditions, the initial context contains a system scaffold and a user message. The scaffold exposes the skill name, purpose, and loading rule, but not the visible skill

instructions. After an accepted skill call, the runtime appends the visible skill instructions inside <skill_body> and adds contract-state feedback with loss mask zero. Low-saturation purple marks skill content, blue marks policy output, green marks environment observations, orange marks feedback, and teal marks verified progress credit throughout the Appendix.

```
Initial model context:
  [SYSTEM] skill catalog + generic loading rule
  [USER] question + skill-selection instruction

After an accepted skill call:
  [ASSISTANT] <skill_call>...</skill_call>
  [RUNTIME] <skill_body>...</skill_body>
            <hint>accepted state and next action</hint>

Each later transition:
  [ASSISTANT] one complete action
  [RUNTIME] optional environment observation
            optional <hint>...</hint>
```

Shared system scaffold. The shared scaffold uses the following two-entry catalog and loading rule:

```
Part 1: Available skills
- name: code-interpreter-protocol
  purpose: Use this skill for mathematical problems that need reliable
    calculation, enumeration, symbolic checking, or program-aided
    verification.
- name: qa-search-protocol
  purpose: Use this skill for factual questions that need external
    evidence or multi-hop verification.

Part 2: Skill loading
Choose the useful skill before reasoning or answering. Output exactly <
    skill_call>SKILL_NAME</skill_call> and replace SKILL_NAME with one
    listed name. The harness will return its instructions inside <
    skill_body>...</skill_body>. Follow the returned instructions and
    environment observations.
```

Each single-task condition exposes only its corresponding catalog entry, while the Joint condition exposes both entries. These conditions use the following user message, with the preloaded Prompt exception described in Appendix C:

```
Question: {normalized question}

Select and load the useful skill before reasoning or answering.
```

Math skill instructions.

```
Use Python calculations as evidence for the mathematical solution. First
    write one complete <think>...</think> to plan the calculation. Then
    put executable Python inside <code>...</code>. The code must perform
    the calculation and print a result. Wait for the environment-owned <
    interpreter>...</interpreter> observation and never write that tag
    yourself. Write another complete <think>...</think> to interpret the
    result. Repeat <code>...</code> and result reasoning when more
    calculation is useful. Finish with Answer: \boxed{...} only after
    analyzing a valid result.
```

Program-aided methods motivate executable computation (Gao et al., 2023; Chen et al., 2022). Tool-integrated methods motivate repeated action-observation reasoning (Gou et al., 2024b; Feng et al., 2026).

Search skill instructions.

```
Use retrieved evidence to answer the factual question. First write one
    complete <think>...</think> to plan the retrieval. Use <search>...</
    search> for the first concise query. Wait for the environment-owned <
    information>...</information> observation and never write that tag
    yourself. Write another complete <think>...</think> to assess the
    evidence. If more evidence is needed, use a refined <research>...</
    research> query and repeat result reasoning. An optional <summary
    >...</summary> may organize gathered evidence but does not replace
    reasoning. Finish with <answer>...</answer> only after analyzing
    valid evidence.
```

The retrieval loop follows Search-R1 (Jin et al., 2025b). AutoRefine (Shi et al., 2026b) and ReSum (Wu et al., 2025) are adjacent precedents for evidence refinement and context summarization rather than reproduced algorithms.

C Experimental Details and Provenance

C.1 Training Configuration

All online experiments use the open-source `slime` framework with its Megatron optimization backend and SGLang rollout engine (Zhu et al., 2025). The reported runs use four GPUs and tensor parallelism two. Math and Search use 200 rollout rounds, while all Joint conditions, including the recovery comparison, use 400. Math training contains 7,500 MATH and 7,473 GSM8K examples. Search training uses 79,168 Natural Questions and 90,447 HotpotQA source rows, and Joint combines all 14,973 Math examples with 13,827 sampled Search examples.

Training uses Adam with learning rate 10^{-6}, betas $(0.9, 0.98)$, warmup over 0.285 of the planned optimizer updates, weight decay 0.01, entropy coefficient 0.001, KL coefficient 0.003, gradient clipping 1.0, policy clipping $(0.20, 0.28)$, and TIS clipping 4.0. Rollout and evaluation temperatures are 1. Table 7 gives the task-specific batch and update budgets. Contract training and evaluation allow at most 20 executor turns per trajectory, including rejected actions, and 8 environment calls. Each executor turn generates at most 1,024 tokens, subject to a total context limit of 16,384 tokens that includes the prompt, generated actions, observations, and contract-state feedback. Exhausting the turn or token budget without accepted termination counts as protocol failure.

The KL reference policy π_{ref} is frozen at the executor's initial checkpoint, distinct from the sampling policy $\pi_{\theta_{\text{old}}}$. For each trainable token, the regularizer in Equation 8 uses the sampled approximation

$$\delta_{i,\ell} = \log \frac{\pi_\theta(a_{i,\ell} \mid x_{i,\ell})}{\pi_{\text{ref}}(a_{i,\ell} \mid x_{i,\ell})}, \qquad d^{\text{KL}}_{i,\ell} = \text{clip}\big(e^{-\delta_{i,\ell}} - 1 + \delta_{i,\ell}, -10, 10\big)\,.$$

The 500-example monitor runs every 25 rounds using a separate evaluation plan, not necessarily the training datasets. For Figure 3, Math uses 250 MATH and 250 GSM8K examples, while Search uses 250 Natural Questions and 250 MuSiQue examples. Figure 3 compares matched Outcome-only and Credit-only conditions with feedback disabled. All-zero groups remain in training and receive zero normalized advantages after group centering. Each plotted point is one monitor evaluation, so no repeated-run interval is reported. Search uses an E5-base-v2 retriever over frozen 2018 Wikipedia with a FAISS index and top-3 retrieval (Jin et al., 2025b). Math uses a restricted Python sandbox with nonempty output and fixed resource limits.

Table 7: Online training configurations. P/R is prompts per rollout, S/P is samples per prompt, and U/R is optimizer updates per rollout. Model-scale runs reuse the corresponding task profile, and the Feedback + Credit conditions in RQ2 reuse the main single-task runs.

Condition	GPU	TP	Rounds	P/R	S/P	U/R	Batch
O-Math / Math interventions	4	2	200	72	5	5	72
O-Search / Search interventions	4	2	200	512	5	10	256
O-Joint	4	2	400	72	5	5	72
Recovery-coefficient comparison	4	2	400	72	5	5	72

C.2 Evaluation and Baselines

Each full-suite condition is evaluated on 6,319 Math and 51,713 Search examples. The Math suite contains 5,000 MATH and 1,319 GSM8K examples. The Search suite contains 3,610 Natural Questions, 11,313 TriviaQA, 14,267 PopQA, 7,405 HotpotQA, 12,576 2WikiMultiHopQA, 2,417 MuSiQue, and 125 Bamboogle examples. Direct omits the contract runtime, while Prompt preloads the skill and uses the contract runtime without training for all executors. Both use one sample per problem and serve as deployment context rather than controls for Outcome-only RL.

These Direct and Prompt runs score an answer extracted from the whole response, even when the runtime has not accepted termination. Their differences measure behavior under different evaluation conditions, not a loss of model knowledge.

For skill-contract evaluation, Equations 9 and 10 define PCR and RF-PCR from accepted termination and the recovery flag. RF-PCR further requires that no rejected-action feedback occur under the runtime with feedback enabled. It can include feedback after accepted actions and is distinct from disabling feedback in RQ3. For contract-trained policies, Task Outcome scores the accepted final answer, while Hint Rate is the percentage of samples that receive at least one rejected-action feedback message. We do not pool Math and Search or combine Protocol Completion with Task Outcome.

The ReTool-style Math row is a matched adaptation that shares the sandbox, data, rollout budget, and pass@1 task verifier with O-Math. Its incorrect-trajectory score is capped at -0.6, while the referenced recipe uses 0. The Search-R1 Recipe row shares the prepared data, retriever, corpus, action environment, and evaluator with O-Search. These matched rows provide local performance context rather than untouched official reproductions.

D Complete Model-Visible Execution Traces

The following correct examples come from the Ours-S policies. Role labels and LaTeX-safe punctuation replace chat-template formatting, while message wording and environment observations are otherwise retained.

D.1 Math

Math — O-Math — Correct and observation-grounded

System

Part 1: Available skills
`- name: code-interpreter-protocol`
purpose: Use this skill for mathematical problems that need reliable calculation, enumeration, symbolic checking, or program-aided verification.
Part 2: Skill loading
Choose the useful skill before reasoning or answering. Output exactly `<skill_call>SKILL_NAME</skill_call>` and replace `SKILL_NAME` with one listed name. The harness will return its instructions inside `<skill_body>...</skill_body>`. Follow the returned instructions and environment observations.

User

Question: A circle of radius 5 with its center at $(0, 0)$ is drawn on a Cartesian coordinate system. How many lattice points (points with integer coordinates) lie within or on this circle?
Select and load the useful skill before reasoning or answering.

Assistant

`<skill_call>code-interpreter-protocol</skill_call>`

Math — O-Math — Correct and observation-grounded (continued)

Runtime skill body

<skill_body>Use Python calculations as evidence for the mathematical solution. First write one complete <think>...</think> to plan the calculation. Then put executable Python inside <code>...</code>. The code must perform the calculation and print a result. Wait for the environment-owned <interpreter>...</interpreter> observation and never write that tag yourself. Write another complete <think>...</think> to interpret the result. Repeat <code>...</code> and result reasoning when more calculation is useful. Finish with Answer: $\boxed{\ldots}$ only after analyzing a valid result.</skill_body>

Contract-state feedback

<hint>Action accepted. State: the skill is loaded. Next: write one complete <think>...</think> plan.</hint>

Assistant

<think>We need to count all lattice points within and on the circle of radius 5 centred at the origin $(0, 0)$. The equation of the circle is $x^2 + y^2 \leq 25$. We need to check all integer pairs (x, y) satisfying this condition.</think>

Contract-state feedback

<hint>Action accepted. State: Python evidence is required. Next: run calculation in <code>...</code> and print a result.</hint>

Assistant code action

```
<code>
count = 0
for x in range(-5, 6):
    for y in range(-5, 6):
        if x**2 + y**2 <= 25:
            count += 1
print(count)
</code>
```

Environment observation

<interpreter>Output: 81</interpreter>

Contract-state feedback

<hint>Action accepted. State: an environment observation is ready. Next: write one complete <think>...</think>.</hint>

Assistant

<think>Based on the written Python script, the number of lattice points within or on the circle is 81.</think>

Contract-state feedback

<hint>Action accepted. State: evidence is analyzed. Next: use <code>...</code> again or give Answer: $\boxed{\ldots}$.</hint>

Final answer

Answer: $\boxed{81}$

D.2 SEARCH

Search — O-Search — Correct and observation-grounded with evidence organization

System

Part 1: Available skills
- name: qa-search-protocol
purpose: Use this skill for factual questions that need external evidence or multi-hop verification.
Part 2: Skill loading
Choose the useful skill before reasoning or answering. Output exactly <skill_call>SKILL_NAME</skill_call> and replace SKILL_NAME with one listed name. The harness will return its instructions inside <skill_body>...</skill_body>. Follow the returned instructions and environment observations.

User

Question: who played harmonica on bring it on home?
Select and load the useful skill before reasoning or answering.

Assistant

<skill_call>qa-search-protocol</skill_call>

Runtime skill body

<skill_body>Use retrieved evidence to answer the factual question. First write one complete <think>...</think> to plan the retrieval. Use <search>...</search> for the first concise query. Wait for the environment-owned <information>...</information> observation and never write that tag yourself. Write another complete <think>...</think> to assess the evidence. If more evidence is needed, use a refined <research>...</research> query and repeat result reasoning. An optional <summary>...</summary> may organize gathered evidence but does not replace reasoning. Finish with <answer>...</answer> only after analyzing valid evidence.</skill_body>

Contract-state feedback

<hint>Action accepted. State: the skill is loaded. Next: write one complete <think>...</think> plan.</hint>

Assistant

<think>I will begin by running a query focusing on the musician who played harmonica in the song "Bring It On Home". My first query should be concise and relevant.</think>

Contract-state feedback

<hint>Action accepted. State: retrieval evidence is required. Next: use a concise <search>...</search> query.</hint>

Assistant

<search>who played harmonica on bring it on home?</search>

Search — O-Search — Correct and observation-grounded with evidence organization (continued)

Environment observation

```
<information>
Doc 1: "Bring It On Home (Sonny Boy Williamson II song)"
Bring It On Home (Sonny Boy Williamson II song) ""Bring It On Home
    "" is a blues song written by American music arranger and
    songwriter Willie Dixon. The first known recording of the song
    was by Sonny Boy Williamson II in 1963. It was later recorded
    by Led Zeppelin in 1969, and has been covered by many artists
    since. Sonny Boy Williamson's version of the song was recorded
    on January 11, 1963 in Chicago. Accompanying Williamson on
    vocals and harmonica were Matt ""Guitar"" Murphy on guitar,
    Milton Rector on bass guitar, Al Duncan on drums, and either
    Lafayette Leake or Billy
```

Environment observation (continued)

```
Doc 2: "Bring It On Home (album)"
Bring It On Home (album) Bring It On Home is the seventh studio
    album by Joan Osborne released under Saguaro Road Records on
    March 27, 2012. It was her first album in five years. On this
    album she co-produced with the guitarist Jack Petruzzelli. The
    album consists entirely of Blues and R&B covers. The record
    also includes tracks originally made famous by American blues
    masters, such as Sonny Boy Williamson (""Bring it on Home""),
    Muddy Waters (""I Want to Be Loved""), as well as recordings
    originally released by some of the best-known R&B performars---
    including Ray Charles (""I Don't Need No
```

Environment observation (continued)

```
Doc 3: "Bring It On Home (Sonny Boy Williamson II song)"
in the concert film ""The Song Remains the Same"".
```

Environment observation (continued)

```
""Bring It On Home"" was played at the reunion of surviving Led
    Zeppelin members staged at Jason Bonham's wedding reception in
    May 1990. Bring It On Home (Sonny Boy Williamson II song) ""
    Bring It On Home"" is a blues song written by American music
    arranger and songwriter Willie Dixon. The first known recording
     of the song was by Sonny Boy Williamson II in 1963. It was
    later recorded by Led Zeppelin in 1969, and has been covered by
     many artists since. Sonny Boy Williamson's version of the song
     was recorded on
</information>
```

Contract-state feedback

```
<hint>Action accepted.  State:  an environment observation is ready.
Next:  write one complete <think>...</think>.</hint>
```

Assistant

<think>The document titled "Bring It On Home (Sonny Boy Williamson II song)", provides evidence that Sonny Boy Williamson played harmonica on the song "Bring It On Home".</think>

Search — O-Search — Correct and observation-grounded with evidence organization (continued)

Contract-state feedback

```
<hint>Action accepted. State: evidence is analyzed.
Next: use <research>...</research> again, organize it with
<summary>...</summary>, or give <answer>...</answer>.</hint>
```

Assistant

<summary>Based on the information from the music document on the 1963 recording of Sonny Boy Williamson's "Bring It On Home," the harmonica player for this song was indeed Sonny Boy Williamson. </summary>

Contract-state feedback

```
<hint>Action accepted. State: evidence is summarized. Next: use
<research>...</research> again or give <answer>...</answer>.</hint>
```

Final answer

```
<answer>Sonny Boy Williamson</answer>
```

E Controlled Intervention Results

RQ2 compares independently trained Qwen2.5-3B conditions for training signals and the recovery coefficient. RQ3 tests inference-time interventions on frozen O-Math and O-Search checkpoints. PCR denotes completion under the runtime configured for each condition and can include recovery after rejected-action feedback. Hint Rate is the percentage of samples that receive at least one rejected-action feedback message.

The feedback-credit factorial, recovery comparison, and visible skill interventions each use MATH and GSM8K for Math and 2WikiMultiHopQA and HotpotQA for Search. We interpret each PCR value only within its intervention and do not pool these controls as one evaluation pass.

E.1 RQ2: Training Behavior

E.1.1 Feedback-Credit Factorial

Each condition starts from the same base checkpoint within its task. All four retain the same task reward, training data, rollout budget, contract verifier, and visible skill instructions. Outcome-only removes both contract-state feedback and verified progress credit, Feedback-only retains feedback, Credit-only retains credit, and Feedback + Credit retains both. In Outcome-only, a rejected action receives no negative reward, preserves the phase, and does not terminate the trajectory. Only a final answer accepted after all required phases is sent to the binary task verifier. The factorial therefore tests learning under one contract runtime rather than comparing Feedback + Credit with permissive terminal-only RL. It does not assume a symmetric interaction between the two components. Figure 3 reports the complete monitor curves for the matched Outcome-only and Credit-only pair.

Table 8: Training-time feedback-credit factorial on Qwen2.5-3B. Each condition is trained independently from the same base checkpoint within its domain. The task reward and contract runtime are fixed, while Feedback toggles contract-state feedback and Credit toggles verified progress credit. Outcome-only uses the contract runtime, in which rejected actions do not receive a penalty or terminate the trajectory, but only a final answer accepted after all required phases can receive the task reward. Task Outcome and PCR are unweighted averages over MATH and GSM8K for Math and 2WikiMultiHopQA and HotpotQA for Search.

Domain	Condition	Feedback	Credit	Outcome	PCR
Search	B00 Outcome-only	0	0	0.00	0.00
Search	B10 Feedback-only	1	0	0.00	0.00
Search	B01 Credit-only	0	1	29.28	98.69
Search	B11 Feedback + Credit	1	1	30.11	98.90
Math	B00 Outcome-only	0	0	0.00	0.00
Math	B10 Feedback-only	1	0	0.00	0.00
Math	B01 Credit-only	0	1	59.76	93.17
Math	B11 Feedback + Credit	1	1	64.61	97.29

E.1.2 Recovery Coefficient

The main setting uses $\rho = 0.90$, while the no-discount condition uses $\rho = 1.00$. Both are independently trained Ours-J policies with the same task mixture and runtime. Main-text Table 2 reports the result. For Math with $\rho = 1.00$, RF-PCR is zero despite high PCR, indicating that every trajectory reaching accepted termination receives rejected-action feedback. RF-PCR still permits feedback after accepted actions and does not measure execution without feedback.

E.2 RQ3: Inference Behavior

E.2.1 Visible Skill Interventions

S0 uses the original visible skill instructions, S1 removes them, S2 uses the independently written paraphrases below, and S3 injects the visible skill instructions of the other task. Only the visible skill instructions z_j change in this test. The contract runtime and task verifier remain fixed. The result therefore identifies an effect of the visible skill instructions while the rest of the skill contract remains fixed. S3 changes the action grammar and is therefore a compatibility stress test rather than a grammar-controlled semantic contradiction.

S2 paraphrased instructions. The registered S2 intervention replaces z_j with an independently written paraphrase. The Math rewrite changes the opening instruction, describes the plan as identifying the needed calculation, and expresses repeated execution as alternating code and reasoning. The Search rewrite describes the plan as identifying what to retrieve, evaluates relevance after each observation, and requests a revised query when evidence is insufficient. Both rewrites retain the action inventory and order, environment ownership of observations, repeated-action permission, and final-answer format.

Paraphrased Math skill instructions.

```
Solve the mathematical problem with evidence produced by Python. Begin
    with one complete <think>...</think> that identifies the needed
    calculation. Place executable Python in <code>...</code> and print
    the computed value. Only the environment may return <interpreter>...</
    interpreter>. After each result, write a complete <think>...</think>
    that interprets it. Alternate further code and reasoning when needed.
     Give Answer: \boxed{...} after using a valid result.
```

Paraphrased Search skill instructions.

```
Answer the factual question from retrieved evidence. Start with one
    complete <think>...</think> that identifies what to retrieve. Put the
```

```
 first focused query in <search>...</search>. Only the environment
may return <information>...</information>. After each observation,
write a complete <think>...</think> that evaluates its relevance. Use
 a revised <research>...</research> query when evidence is
insufficient. A <summary>...</summary> may organize evidence. Finish
with <answer>...</answer> only after evidence analysis.
```

Table 9: Inference-time visible skill instruction interventions on frozen Qwen2.5-3B single-task checkpoints. Only the visible skill instructions z_j change. Paraphrased rewrites wording, plan descriptions, and transition language while preserving action tags, action order, environment-owned observations, repeated-action permission, and the termination format. Appendix E.2.1 gives the exact Math and Search bodies. Outcome is pass@1 for Math and exact match for Search.

Domain	Visible skill	MATH	GSM	2Wiki	Hotpot	Avg.	PCR	Hint Rate
Math	Original	48.62	80.59	–	–	64.61	97.29	5.59
Math	Removed	42.76	74.68	–	–	58.72	91.34	11.70
Math	Paraphrased	43.92	75.59	–	–	59.75	94.36	7.27
Math	Wrong Search	1.48	2.35	–	–	1.92	4.71	99.69
Search	Original	–	–	29.87	30.34	30.11	98.90	0.99
Search	Removed	–	–	22.57	18.51	20.54	63.81	14.86
Search	Paraphrased	–	–	26.92	26.49	26.71	98.86	1.08
Search	Wrong Math	–	–	29.00	30.93	29.96	98.31	1.74

E.2.2 Observation and Feedback Interventions

For observation withholding, the environment first executes the admissible action and the task adapter validates the returned observation. The runtime then replaces only the model-visible payload with a non-evidential placeholder while preserving the recorded validity and contract transition. This intervention removes observation content available to the executor without changing whether the original action and observation passed runtime validation. The four dataset-level changes are reported in Section 4.4 and visualized in Figure 4(c).

The feedback-removal intervention suppresses the complete feedback channel at inference while retaining validation. Its result is reported in main-text Table 2.

F Discussion

Grounded skill-following treats a fixed expert-authored skill as a multi-step behavioral standard rather than a local prompt. A skill contract makes this behavior trainable and auditable by verifying ordered progress. Under the tested contract runtime, Credit-only develops nonzero PCR and Task Outcome while Outcome-only remains at zero. PCR and Task Outcome report Protocol Completion and answer correctness separately, while RF-PCR identifies completion without rejected-action feedback.

The Math/Search interventions and joint/separate comparisons suggest that high Protocol Completion under a common contract design can coexist with task-dependent reliance on execution signals and differing Task Outcome gains. Motivated by prior studies of skill neurons and layer-aware task arithmetic (Wang et al., 2022; Chen et al., 2025), we hypothesize that these capabilities involve both shared and task-specific parameters or representations. Our behavioral results do not identify such components, and differences in task exposure further limit the interpretation of joint versus separate training. Testing this hypothesis would require targeted interventions on parameters or representations under controlled training exposure.

The skill contracts are manually specified. The interventions establish dependence on visible skill instructions, observations, and contract-state feedback within this implementation, but not token-level causal evidence use or scaffold-free internalization. The results establish the proposed problem and method in the tested Math, Search, and five Qwen executor configurations. They do not establish automatic contract construction, universal transfer, guaranteed optimization improvement, or failure of terminal-only RL under a permissive runtime.