LEARNING WITHOUT REMEMBERING: AGENT SELF-IMPROVEMENT THROUGH EXPERIENCE COMPILATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Synthetic interaction data provide a scalable resource for training tool-using agents, yet successful demonstrations may not meet the learning needs of the current student. We investigate how feedback accumulated during generation and learning can be translated into requirements for constructing future training data. We introduce **Learning Without Remembering (LWR)**, an agent learning framework that compiles stage-level experience into synthesis policies. LWR jointly analyzes generation successes and failures alongside changes in student performance and behavior before and after training, distilling synthesis guidelines with explicit applicability conditions that can be reused across examples. These guidelines are translated into generation instructions, workflow allocation, and interaction plans. With the generator and controller weights held fixed, the updated policy guides the generation of complete interaction trajectories, which undergo environment verification before being used for continued supervised fine-tuning of the student. Subsequent generation and learning feedback supports further policy revision. Past experience is thus consolidated into generation requirements and incorporated into student parameter learning through new training data, without retrieving past cross-task trajectories at test time. Experiments on τ^3 and AppWorld with two student model sizes demonstrate improved task completion. Compared with fixed synthesis, LWR improves τ^3 macro-average success rates by 5.00 and 6.67 percentage points for the 8B and 30B students, respectively, and yields additional gains on most AppWorld metrics.

1 INTRODUCTION

In recent years, the applications of large language models have expanded from text generation to interaction with external environments through tools. Tool-using agents solve tasks through multi-turn interactions with users and environments, coordinating communication, tool use, and responses to environmental feedback (Yao et al., 2023; 2025). Training these agents requires demonstrations that capture complete interaction processes, yet collecting real-world trajectories is costly and difficult to scale across diverse tasks and execution paths. Synthetic data offers a scalable alternative: automated task construction and simulated interactions can produce complete, verifiable trajectories that provide agents with new learning experiences (Prabhakar et al., 2026).

Yet generating successful demonstrations at scale does not mean knowing how to teach the current student effectively. For a student that repeatedly queries a tool without making progress, useful demonstrations should highlight how returned information supports the selection and execution of subsequent actions. Simply increasing the number of similar trajectories may leave these behavioral connections unaddressed. Meanwhile, each round of generation and training provides clues about teaching needs: which interactions are difficult to construct, which behaviors remain unlearned, and which capabilities change after training. These clues provide a basis for adjusting the situations and behaviors that subsequent demonstrations should cover, making generation and learning feedback valuable resources for improving training data production.

Prior work has explored different ways of using experience and feedback. Past experience can be distilled into reusable knowledge and workflows that guide subsequent task execution (Zhao et al., 2024; Wang et al., 2025) (Figure 1(a)). DataEnvGym and Agent-World use student errors and capability feedback to guide data generation (Khan et al., 2025; Dong et al., 2026), while Auto Evol-

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.

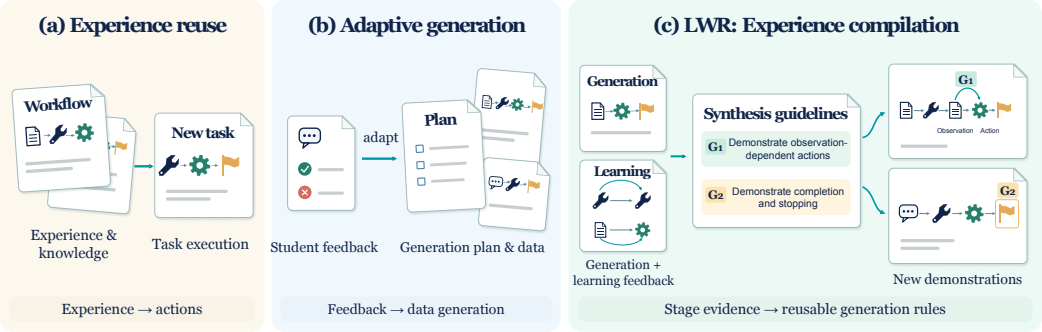


Figure 1: Conceptual positioning of LWR. (a) Experience reuse guides subsequent task execution. (b) Adaptive generation uses feedback to adjust training data or generation strategies. (c) LWR compiles stage-level generation and learning feedback into reusable synthesis guidelines that shape new demonstrations.

Instruct refines instruction-evolution strategies using feedback from the generation process (Zeng et al., 2024) (Figure 1(b)). Building on these advances, we ask: how can stage-level evidence from generation and learning be distilled into demonstration construction requirements that are reusable across examples? Generation outcomes and changes in student behavior across different tasks may reveal shared learning needs, indicating the situations, information dependencies, and action transitions that subsequent demonstrations should cover. We organize these requirements into synthesis guidelines with explicit applicability conditions to guide the construction of multiple new trajectories (Figure 1(c)).

To this end, we introduce **Learning Without Remembering (LWR)**, a stage-wise agent learning framework that connects data production with model learning. At each stage, LWR jointly analyzes generation outcomes and changes in student behavior to revise synthesis guidelines with explicit applicability conditions, translating them into the next stage's generation instructions, workflow allocation, and interaction plans. The generator and controller weights remain fixed: stage-level feedback updates the synthesis policy, while the student undergoes continued supervised fine-tuning on newly generated, environment-verified complete interaction trajectories. Experience thus contributes to student parameter learning through subsequent training data, without requiring per-example retrieval of past cross-task trajectories during generation or test-time execution. Experiments on τ^3 and AppWorld demonstrate improved task completion with two student model sizes. Ablations against fixed synthesis further show the additional learning benefits of feedback-driven synthesis-policy updates.

Our contributions are threefold:

1. **A stage-wise framework connecting data production and agent learning.** We introduce LWR, which integrates synthesis-policy revision and student parameter updates into a single learning process, allowing stage-level experience to inform student learning through newly generated training data.

2. **An experience compilation mechanism for demonstration construction.** We combine generation outcomes with changes in student performance and behavior to derive synthesis guidelines with explicit applicability conditions that are reusable across examples. These guidelines translate stage-level evidence into concrete requirements for the situations and behavioral relationships that subsequent demonstrations should cover.

3. **An empirical evaluation of feedback-driven synthesis-policy updates.** We evaluate LWR on τ^3 and AppWorld with two student model sizes, examine the additional gains from policy updates through fixed-synthesis ablations, and analyze changes in successful task coverage and within-scenario task completion.

2 Related Work

Agent Data Synthesis. Research on synthetic data for tool-using agents focuses on interaction coverage and trajectory reliability. APIGen generates reliable function-calling demonstrations through format checks, actual function execution, and semantic verification (Liu et al., 2024). APIGen-MT transforms reviewed task blueprints into complete multi-turn trajectories that capture user communication and environmental feedback (Prabhakar et al., 2026). A complementary approach constructs tasks from executable interactions: Trajectory2Task first explores valid tool-call trajectories and then derives user-facing tasks involving ambiguous, changing, or infeasible requests, covering more complex interaction requirements (Wang et al., 2026). AgentRefine incorporates errors and subsequent refinements into synthetic interaction trajectories, teaching students to correct earlier actions based on environmental feedback (Fu et al., 2025). Building on environment-verified trajectory synthesis, LWR uses stage-level learning feedback to revise the requirements for constructing subsequent demonstrations.

Adaptive Data Generation. Feedback-driven data generation uses information from student performance and the generation process to adapt subsequent training data and their construction strategies. DataEnvGym formulates data generation as a sequential decision-making process, separating a policy that plans data generation based on student errors and skill-level performance from an engine that implements these plans (Khan et al., 2025). Agent-World combines capability-gap diagnosis, dynamic task synthesis, and training across multiple environments (Dong et al., 2026). CoEvolve extracts signals such as forgetting and uncertainty from interaction trajectories to identify failure-prone interaction patterns, guide task synthesis, and update the training distribution through environment-validated tasks (Yang et al., 2026).

Related work also explores the automatic optimization of generation strategies and system prompts. Auto Evol-Instruct iteratively revises instruction-evolution strategies based on problems encountered during the evolution process (Zeng et al., 2024). GEPA uses natural-language reflection on execution traces to propose, evaluate, and combine prompt updates (Agrawal et al., 2026). LWR jointly analyzes stage-level generation successes and failures alongside changes in student performance, consolidating this evidence into synthesis guidelines with explicit applicability conditions that are reusable across examples. These guidelines specify the situations and behavioral relationships that subsequent demonstrations should cover and are translated into generation instructions, workflow allocation, and interaction plans.

Experience Abstraction and Reuse. Agent experiences can be abstracted into reflections, workflows, and reusable strategies. Reflexion converts task feedback into verbal reflections to guide subsequent attempts (Shinn et al., 2023). ExpeL extracts knowledge from task experiences and uses both distilled insights and past experiences for inference-time decision-making (Zhao et al., 2024). Agent Workflow Memory induces reusable workflows and supports both offline and online experience reuse (Wang et al., 2025). ACE organizes context into an evolving playbook through generation, reflection, and curation, enabling structured, incremental updates (Zhang et al., 2026). These approaches primarily use abstracted experience to guide task execution. LWR uses stage-level experience to construct subsequent training demonstrations, allowing reusable guidelines to influence student learning through newly generated data.

3 Method

LWR organizes agent learning into stages that connect data generation, learner training, and synthesis-policy revision. At stage t, the system maintains learner parameters θ_t and a structured synthesis policy:

$$\psi_t = (K_t, I_t, q_t, \Pi_t), \tag{1}$$

where K_t is the set of active synthesis guidelines, I_t specifies generation instructions, q_t controls workflow allocation, and Π_t describes an interaction plan. The generator and controller weights remain fixed. The controller revises the synthesis policy using stage feedback, while the learner updates its parameters through generated training data.

Starting from an initial learner θ_0 and a base policy ψ_0, stage t generates and verifies training trajectories under the current policy, updates the learner, and collects feedback for the next policy revi-

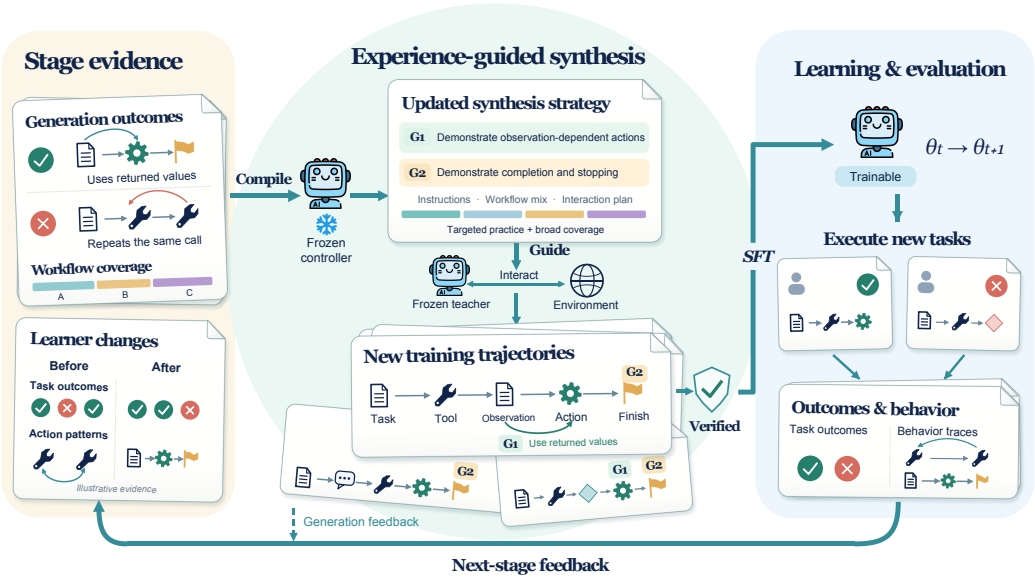


Figure 2: Overview of LWR. A frozen controller compiles generation outcomes and learner behavior changes into synthesis guidelines, updating generation instructions, workflow allocation, and interaction plans. A frozen teacher interacts with the environment to generate new trajectories, which undergo environment verification and are then used for learner supervised fine-tuning. Generation evidence and learner evaluations provide feedback for next-stage policy revision. The illustrated guidelines emphasize observation-dependent actions, task completion, and appropriate stopping.

sion. This process forms a stage-wise learning loop (Figure 2). We describe experience compilation, guideline-guided trajectory synthesis, and learner updates with feedback collection below.

3.1 Compiling Experience into Synthesis Guidelines

Experience compilation translates stage-level generation and learning evidence into requirements for subsequent training data. The controller receives both concrete trajectory evidence and aggregate statistics. Generation evidence includes successful and failed interaction trajectories, workflow coverage, interaction behavior, and the composition of the data actually used for training. Learning evidence includes task performance, execution trajectories, and behavioral changes before and after training. Concrete trajectories reveal relationships between observations, actions, and their outcomes, while aggregate statistics help characterize the distribution and coverage of these patterns.

The controller examines evidence across tasks to infer shared teaching needs. It first analyzes the conditions associated with potential behavioral gaps, including relevant information dependencies and action transitions. It then formulates teaching hypotheses and translates them into demonstration construction requirements. These requirements specify the task situations, observation–action dependencies, and behavioral transitions that subsequent demonstrations should cover. The controller also considers workflow coverage in the training data and changes in previously acquired behaviors, seeking to balance targeted practice with coverage of other training situations.

Each guideline records five elements: applicability conditions, supporting observations, an explanatory hypothesis, a proposed generation change, and an expected learner behavior. Applicability conditions define its scope, the generation change specifies how new demonstrations should be constructed, and the expected behavior provides a reference for subsequent feedback analysis. Guidelines and their supporting evidence are retained in stage records for later revision.

Given stage feedback F_t, the fixed-weight controller $\mathcal{C}$ updates the current synthesis policy:

$$\psi_{t+1} = \mathcal{C}(\psi_t, F_t, \mathcal{T}_{t+1}), \tag{2}$$

where the task-support catalogue $\mathcal{T}_{t+1}$ summarizes the workflows and interaction conditions supported by the next-stage available task set $\mathcal{U}_{t+1}$, defining the feasible scope of policy revision.

Policy revision can retain, adjust, or remove existing guidelines and update the associated generation instructions, workflow allocation, and interaction plan. Subsequent stages examine how the generation requirements are reflected in the resulting training data and how learner behavior changes, using this evidence to revise the teaching hypotheses. Existing requirements may remain unchanged; policy revision does not require introducing new guidelines at every stage.

3.2 Guideline-Guided Trajectory Synthesis

The synthesis policy influences data production through task allocation and interaction guidance. Workflow allocation q_t distributes generation opportunities across training situations. Generation instructions I_t specify the behavioral requirements to emphasize, while the interaction plan Π_t describes the information acquisition, condition checks, and action transitions to demonstrate. Guidelines applicable to the current task are combined with these controls to form the teacher's generation context.

Let $\mathcal{U}_t$ denote the available training task set and B_t the generation-attempt budget. The system schedules tasks, builds a generation context for each task, and produces trajectories through environment interaction:

$$\begin{aligned}(x_{t,1}, \ldots, x_{t,B_t}) &= \mathrm{Schedule}(\mathcal{U}_t, q_t, B_t),\\ c_t(x) &= \mathrm{BuildContext}\big(I_t, \Pi_t, K_t(x)\big),\\ \tau_{t,i} &\sim \mathrm{Rollout}\big(\mathcal{G}, \mathcal{E}, x_{t,i}, c_t(x_{t,i})\big).\end{aligned} \tag{3}$$

Here, $K_t(x)$ denotes the guidelines applicable to task x, $c_t(x)$ is the resulting generation context, $\mathcal{G}$ is the fixed-weight teacher generator, and $\mathcal{E}$ is the interaction environment. Each trajectory $\tau_{t,i}$ contains user communication, tool calls, environment observations, and a final response.

The teacher determines the execution path from task requirements and observations received during interaction. The system generates complete demonstrations across training situations to cover weak behavioral relationships indicated by stage feedback, while retaining demonstrations of normal execution, task completion, and appropriate stopping. Our main implementation generates new trajectories within existing training tasks, so task allocation and teaching requirements are constrained by the available tasks and environment conditions.

Let $\mathcal{A}_t = \{(x_{t,i}, \tau_{t,i})\}_{i=1}^{B_t}$ denote the generation records at stage t. A fixed acceptance function V screens trajectories for task completion, interaction completeness, and overt behavioral degeneration. Accepted new trajectories are combined with designated maintenance data M_t to form the training set:

$$\begin{aligned}D_t^{\mathrm{new}} &= \{\tau_{t,i} \mid V(x_{t,i}, \tau_{t,i}) = 1\},\\ D_t &= D_t^{\mathrm{new}} \uplus M_t.\end{aligned} \tag{4}$$

The operator $\uplus$ denotes the combination of training examples. Acceptance criteria remain fixed during policy revision, and all generation attempts contribute to stage feedback. Environment verification assesses execution validity, while subsequent learner evaluation examines the downstream learning value of the data.

3.3 Learner Update and Feedback Collection

The learner continues supervised fine-tuning from its preceding-stage parameters θ_t:

$$\theta_{t+1} = \mathrm{SFT}(\theta_t, D_t). \tag{5}$$

Newly verified trajectories provide the teaching content induced by the current synthesis policy, while maintenance data supplement coverage of foundational behaviors. Data composition and training budgets are specified in the experimental setup.

Training preserves the full interaction context, including the connections between tool calls, environment observations, subsequent actions, and final responses. Supervision is applied to selected agent responses and tool calls, with user messages and tool observations serving as conditioning

Table 1: Results (%) on τ^3 and AppWorld. Macro denotes the unweighted average across the three τ^3 domains. Nor/Cha denote Test-Normal/Test-Challenge, and TGC/SGC denote task/scenario goal completion. Bold indicates the best result for each metric within each model group.

Model	Method	τ^3				AppWorld			
		Retail	Airline	Telecom	Macro	Nor-TGC	Nor-SGC	Cha-TGC	Cha-SGC
Qwen3 8B	Direct	35.00	10.00	0.00	15.00	19.60	5.40	5.50	1.40
	APIGen-MT	30.00	15.00	2.50	15.83	14.90	5.40	5.80	2.90
	DataEnvGym	27.50	25.00	**7.50**	20.00	25.00	16.10	13.20	3.60
	Auto Evol-Instruct	30.00	**45.00**	5.00	**26.67**	17.90	3.60	5.30	2.20
	Agent-World	32.50	15.00	5.00	17.50	19.10	7.10	8.60	2.90
	AgentRefine	27.50	10.00	0.00	12.50	8.90	3.60	2.20	0.00
	LWR (Ours)	**37.50**	30.00	2.50	23.33	**47.00**	**28.60**	**20.60**	**7.20**
Qwen3 30B-A3B	Direct	52.50	35.00	2.50	30.00	26.20	10.70	15.60	5.00
	APIGen-MT	50.00	25.00	7.50	27.50	50.60	28.60	23.50	10.80
	DataEnvGym	60.00	45.00	12.50	39.17	56.50	33.90	**30.70**	14.40
	Auto Evol-Instruct	60.00	45.00	15.00	40.00	47.60	30.40	24.70	13.70
	Agent-World	50.00	35.00	7.50	30.83	39.90	21.40	22.80	10.10
	AgentRefine	55.00	10.00	5.00	23.33	25.00	8.90	9.60	2.90
	LWR (Ours)	**62.50**	**60.00**	**32.50**	**51.67**	**59.50**	**41.10**	30.50	**17.30**

context. Explicitly identified erroneous actions remain in the context, while their corresponding prediction targets are masked. Synthesis guidelines and additional teaching instructions are removed when constructing learner training inputs.

Before and after training, the system measures task performance and behavioral statistics on the same stage-feedback task set $\mathcal{B}_t$ under comparable evaluation conditions:

$$\begin{aligned} b_t^- &= \mathrm{Eval}(\theta_t; \mathcal{B}_t), \\ b_t^+ &= \mathrm{Eval}(\theta_{t+1}; \mathcal{B}_t). \end{aligned} \tag{6}$$

The vectors b_t^- and b_t^+ contain statistics such as task success, repeated calls, and termination behavior. The system also retains concrete trajectory evidence for stage-level analysis, denoted by $\mathcal{R}_t$, including relevant records from generation and learner execution. Stage feedback is defined as:

$$F_t = \left(\mathcal{R}_t,\ \mathrm{Summarize}(\mathcal{A}_t, D_t),\ b_t^-,\ b_t^+,\ b_t^+ - b_t^-\right). \tag{7}$$

Here, $\mathrm{Summarize}(\mathcal{A}_t, D_t)$ aggregates generation outcomes, actual training-data composition, and interaction behavior. Concrete trajectory evidence supports analysis of behavioral processes, while pre- and post-training statistics and their differences characterize current performance, learning gains, and losses in previously acquired behaviors. Together, they inform teaching decisions for the next stage.

The feedback F_t is then supplied to the controller to revise the next-stage policy ψ_{t+1} through equation 2. The experimental protocol specifies access to feedback tasks used for policy updates and to other evaluation tasks.

Stage experience is retained in the current synthesis policy and influences learner parameter updates through newly generated training data. Per-task generation uses the current policy and task context, while learner execution relies on trained parameters and the ongoing interaction history. Neither requires per-example retrieval of past cross-task trajectories. This is the sense of *Learning Without Remembering*.

4 Experiments

4.1 Experimental Setup

Benchmarks and metrics. We evaluate the downstream training effectiveness of synthetic data on τ^3 (Yao et al., 2025; Barres et al., 2026) and AppWorld (Trivedi et al., 2024). We use the τ^3

Table 2: Ablation of feedback-guided synthesis. Direct receives no additional training; Fixed synthesis uses an unchanged synthesis policy; LWR updates the synthesis policy using stage-level feedback. Metrics and formatting follow Table 1.

Model	Method	τ^3				AppWorld			
		Retail	Airline	Telecom	Macro	Nor-TGC	Nor-SGC	Cha-TGC	Cha-SGC
Qwen3 8B	Direct	35.00	10.00	0.00	15.00	19.60	5.40	5.50	1.40
	Fixed synthesis	**40.00**	10.00	**5.00**	18.33	23.80	10.70	11.30	3.60
	LWR (Ours)	37.50	**30.00**	2.50	**23.33**	**47.00**	**28.60**	**20.60**	**7.20**
Qwen3 30B-A3B	Direct	52.50	35.00	2.50	30.00	26.20	10.70	15.60	5.00
	Fixed synthesis	**72.50**	50.00	12.50	45.00	56.00	32.10	**32.40**	16.60
	LWR (Ours)	62.50	**60.00**	**32.50**	**51.67**	**59.50**	**41.10**	30.50	**17.30**

release of the official benchmark implementation and report task success rates on Retail, Airline, and Telecom, along with their unweighted macro-average. For AppWorld, we evaluate on Test-Normal and Test-Challenge using task goal completion (TGC) and scenario goal completion (SGC). SGC requires successful completion of all three tasks within a scenario.

Student models and training. We use Qwen3-8B and Qwen3-30B-A3B-Instruct-2507 (Yang et al., 2025) as student models, comparing data construction methods from the same initial checkpoint within each backbone group. LWR keeps the controller and generator weights fixed and revises the synthesis policy using stage-level feedback. The resulting complete interaction trajectories are verified in the environment and used for supervised fine-tuning of the student. Main and ablation results are averaged over three random seeds. Details of the generation models, dataset sizes, and training hyperparameters are provided in Appendix A.3.

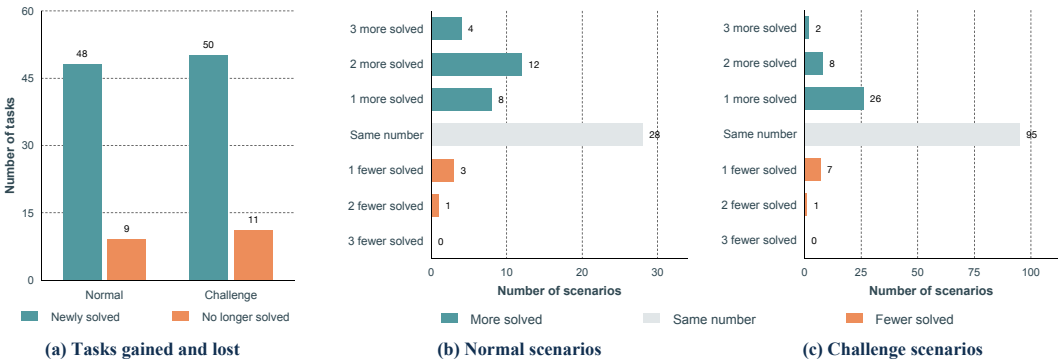
Baselines. Our baselines include Direct, which performs no additional training; APIGen-MT (Prabhakar et al., 2026), which uses released trajectories for supervised fine-tuning; and environment-adapted variants of DataEnvGym (Khan et al., 2025), Auto Evol-Instruct (Zeng et al., 2024), Agent-World (Dong et al., 2026), and AgentRefine (Fu et al., 2025). These comparisons cover learning from public demonstrations, student-error-driven generation, generation-strategy evolution, diagnosis-guided synthesis, and training on trajectories containing error correction. APIGen-MT uses released trajectories on τ^3 and in-domain data synthesized following its blueprint-review procedure on AppWorld. Data sources and environment-specific adaptations are detailed in Appendix B.1.

4.2 Main Results

Table 1 reports the results on τ^3 and AppWorld with two student model sizes. LWR improves task completion on both benchmarks and outperforms the comparison methods on most metrics.

Results on τ^3. With Qwen3-30B-A3B, LWR achieves a macro-average success rate of 51.67%, outperforming Direct by 21.67 percentage points and the strongest baseline, Auto Evol-Instruct, by 11.67 points. LWR achieves the best results across Retail, Airline, and Telecom. In particular, its Telecom success rate reaches 32.50%, compared with the highest baseline score of 15.00%, demonstrating substantial improvement in a domain where the initial student performs poorly. With Qwen3-8B, LWR increases the macro-average success rate from 15.00% to 23.33% and achieves the highest Retail score. Although its 8B macro-average remains below Auto Evol-Instruct's 26.67%, LWR outperforms this method in all three τ^3 domains with the 30B model and on all AppWorld metrics with both model sizes.

Results on AppWorld. LWR achieves the best results on all four metrics with Qwen3-8B. Its TGC reaches 47.00% on Test-Normal and 20.60% on Test-Challenge, exceeding the strongest baseline, DataEnvGym, by 22.00 and 7.40 percentage points, respectively. The corresponding SGC scores reach 28.60% and 7.20%, improving over DataEnvGym by 12.50 and 3.60 points. With Qwen3-



(a) Tasks gained and lost (b) Normal scenarios (c) Challenge scenarios

Figure 3: Task- and scenario-level comparison between LWR and Fixed synthesis on AppWorld with Qwen3-8B. (a) Numbers of tasks solved by LWR but not by Fixed synthesis, and vice versa. (b,c) Distributions of scenario-level changes in solved-task counts on Test-Normal and Test-Challenge, respectively. Each scenario contains three tasks. Changes are computed as LWR minus Fixed synthesis: positive values indicate gains, negative values indicate losses, and zero indicates an unchanged solved-task count.

30B-A3B, LWR achieves the best results on three of the four metrics: 59.50% TGC and 41.10% SGC on Test-Normal, and 17.30% SGC on Test-Challenge. Its Challenge TGC of 30.50% is close to DataEnvGym's 30.70%. The SGC improvements further show that the gains extend to completing all related tasks within a scenario.

Comparison across data construction methods. The strengths of different methods vary across environments and domains. Auto Evol-Instruct performs strongly on τ^3, but its 8B model achieves 45.00% on Airline and only 5.00% on Telecom, indicating uneven gains across domains. DataEnvGym is the most competitive baseline on AppWorld, showing the training value of student-feedback-guided generation in this environment. LWR outperforms APIGen-MT, Agent-World, and AgentRefine on the τ^3 macro-average and all AppWorld metrics with both model sizes. It also achieves the best results in all three τ^3 domains with the 30B model and in seven of the eight model–metric combinations on AppWorld. Overall, these results support revising synthesis requirements using stage-level generation and learning evidence to obtain gains across a broader range of domains and evaluation dimensions.

4.3 Ablation Study

We compare Direct, Fixed synthesis, and LWR to examine the contribution of feedback-driven synthesis-policy updates. Fixed synthesis constructs training data using an unchanged synthesis policy, whereas LWR revises the policy using stage-level feedback before producing new training trajectories. Table 2 presents the results.

Benefits of feedback-guided synthesis. Training on data produced by a fixed policy already improves student performance, while incorporating feedback yields additional gains. On τ^3, LWR improves the macro-average success rate from 18.33% to 23.33% for the 8B model and from 45.00% to 51.67% for the 30B model. On AppWorld, LWR outperforms Fixed synthesis on all four metrics with the 8B model, increasing Normal and Challenge TGC by 23.20 and 9.30 percentage points, respectively. With the 30B model, Normal TGC and SGC increase by 3.50 and 9.00 points, and Challenge SGC also improves. These results demonstrate additional learning benefits from adapting data construction to stage-level feedback beyond the gains provided by synthetic training alone.

Overall gains and domain-level variation. Fixed synthesis retains an advantage on some individual metrics, including Retail success rates on τ^3 and the 30B model's Challenge TGC on AppWorld. However, its gains are uneven across domains. For example, with the 30B model, Fixed synthesis achieves 72.50% on Retail but only 12.50% on Telecom. Although LWR obtains a lower Retail score, it improves Airline and Telecom success rates by 10.00 and 20.00 percentage points,

respectively, resulting in a higher macro-average. For this model, feedback-guided synthesis therefore improves domains that remain relatively weak under the fixed policy, yielding stronger overall performance.

Model- and environment-dependent gains. The benefits of policy updates do not follow a uniform trend with model size. On AppWorld Test-Normal, LWR improves TGC over Fixed synthesis by 23.20 percentage points for the 8B model, compared with 3.50 points for the 30B model. On τ^3, the corresponding macro-average gains are 5.00 and 6.67 points. These differences indicate that the learning value of a synthesis policy should be assessed in relation to the particular student and task environment, motivating LWR's use of current-stage evidence to revise generation requirements.

Task success and complete scenario coverage. TGC and SGC reveal different aspects of the benefits from policy updates. For the 30B model on Test-Normal, LWR improves TGC by 3.50 percentage points and SGC by 9.00 points over Fixed synthesis. On Test-Challenge, TGC decreases from 32.40% to 30.50%, while SGC increases from 16.60% to 17.30%. Thus, aggregate task success alone does not fully describe how successful tasks are distributed across scenarios: a slightly lower overall task success rate can coexist with a higher rate of fully solved scenarios. Considering both metrics provides a more complete assessment of the downstream learning effects of synthetic data.

4.4 Task- and Scenario-Level Analysis

To understand the changes underlying the aggregate gains, we compare the task-level outcomes of Qwen3-8B trained with Fixed synthesis and LWR, and examine changes in the number of successful tasks within each AppWorld scenario. Figure 3 summarizes this paired analysis for a representative run; the evaluation scope and paired-statistic definitions are provided in Appendix E.2.

New successes and retention. On Test-Normal, LWR solves 48 tasks that Fixed synthesis fails, while failing on 9 tasks solved by Fixed synthesis, yielding a net gain of 39 successful tasks. On Test-Challenge, the corresponding counts are 50 gains and 11 losses, again yielding a net gain of 39. LWR also retains 77.5% and 76.6% of the tasks solved by Fixed synthesis on the two splits, respectively. New successes substantially outnumber lost successes on both splits, showing that the aggregate improvements reflect an expansion in successful task coverage.

Distribution of within-scenario improvements. Each AppWorld scenario contains three related tasks. On Test-Normal, the number of successful tasks increases in 24 scenarios, decreases in 4, and remains unchanged in 28. The corresponding counts on Test-Challenge are 36, 8, and 95. Among the improved Normal scenarios, 16 gain two or three successful tasks, accounting for 66.7% of improved scenarios. On Challenge, the corresponding count is 10, or 27.8%, while the remaining 26 improved scenarios each gain one successful task. Improvements on Normal therefore more often involve gains across multiple tasks within the same scenario, whereas Challenge improvements more frequently involve a single-task gain.

Changes in solved-task counts complement SGC. For example, moving from zero to two successful tasks represents substantial partial progress but does not increase SGC; moving from two to three completes the entire scenario. The distributions in Figure 3 thus also capture progress within scenarios that remain partially solved, supplementing the scenario completion rates reported in the main table.

5 Conclusion

This work investigates how stage-level learning experience can improve the production of future training data. LWR translates generation outcomes and learner changes into reusable synthesis guidelines, allowing data construction strategies to adapt throughout learning. Experiments on τ^3 and AppWorld show that feedback-driven strategy updates yield additional downstream learning gains and broaden successful task coverage. These findings support treating the synthesis strategy itself as an object of continual improvement and evaluating training data through its effects on student learning. Future work will investigate how gains accumulate across successive updates and under what conditions synthesis guidelines remain effective over time.

AI USE STATEMENT

In this work, we used generative AI tools to assist with synthetic data generation, refinement of conceptual frameworks and research hypotheses, experimental design and execution, method implementation, data cleaning and formatting, qualitative feedback analysis, result interpretation, and manuscript translation. We did not use generative AI tools to formulate mathematical claims, provide critical ingredients for mathematical proofs, or write proofs; theoretical model development and related proof tasks are not applicable to this work. Additionally, we used generative AI tools for literature summarization, code editing, and manuscript polishing. AI-generated images were not directly used as final scientific figures. All AI-assisted work was reviewed by the authors, including verification of code and experimental procedures, as well as checks of literature descriptions and result interpretations. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide our implementation in an anonymous repository at `https://anonymous.4open.science/r/LWR-BFD4/`. The method and experimental setup are described in Sections 3 and 4.1, respectively, to support reproduction of our approach and experiments.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL `https://proceedings.iclr.cc/paper_files/paper/2026/file/0e9e708b6f48e14fd0ac29e167413f76-Paper-Conference.pdf`.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik R Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026. URL `https://openreview.net/forum?id=OC2z7iSQKa`.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence, 2026. URL `https://arxiv.org/abs/2604.18292`.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma GongQue, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. Agentrefine: Enhancing agent generalization through refinement tuning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=FDimWzmcWn`.

Zaid Khan, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. Dataenvgym: Data generation agents in teacher environments with student feedback. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=00SnKBGTsz`.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 54463–54482. Curran Associates, Inc., 2024. doi: 10.52202/079017-1725. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/61cce86d180b1184949e58939c4f983d-Paper-Datasets_and_Benchmarks_Track.pdf`.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Quoc Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=qk6ORqQ4Cu.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Pei Chen, Ziwei Dong, Jing Huang, Jiri Gesi, Xianfeng Tang, Chen Luo, Qun Liu, Yisi Sang, Hanqing Lu, Manling Li, Jin Lai, and Dakuo Wang. Trajectory2Task: Training robust tool-calling agents with synthesized yet verifiable data for complex user intents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44021–44044, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.2037. URL https://aclanthology.org/2026.acl-long.2037/.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NTAhi2JEEE.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. Co-Evolve: Training LLM agents via agent-data mutual evolution. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23015–23036, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1055. URL https://aclanthology.org/2026.acl-long.1055/.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-Bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Weihao Zeng, Can Xu, Yingxiu Zhao, Jian-Guang Lou, and Weizhu Chen. Automatic instruction evolving for large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 6998–7018, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.397. URL https://aclanthology.org/2024.emnlp-main.397/.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=eC4ygDs02R.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN 978-1-57735-887-9. doi: 10.1609/aaai.v38i17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

A Experimental Details

A.1 Task Sets and Evaluation

We use Qwen3-8B and Qwen3-30B-A3B-Instruct-2507 as student models. Qwen3-8B runs in its native non-thinking mode. Direct denotes the original student without additional supervised fine-tuning in our experiments. Table 3 summarizes the task sets.

Table 3: Training support and evaluation task sets.

Benchmark	Training support	Evaluation
τ^3	Official TRAIN: 74 Retail, 30 Airline, and 74 Telecom tasks	40 Retail, 20 Airline, and 40 Telecom tasks
AppWorld	90 public TRAIN tasks from 30 scenario families; native constructors instantiate fresh training environments	Test-Normal: 168 tasks from 56 scenarios; Test-Challenge: 417 tasks from 139 scenarios

For τ^3, we report task success rates for individual domains and their unweighted macro average. Each complete evaluation uses one simulation per task, with matching task and simulation-seed identities across corresponding comparisons. Each interaction allows up to 40 steps and five errors. The student, user simulator, and natural-language assertion judge use temperature 0, with a default request output limit of 1024 tokens.

For AppWorld, we use the official ReAct code agent, execution environment, and scorer. The student has a context length of 32768 tokens and runs for at most 50 interaction steps at temperature 0. Each response permits up to 2048 output tokens, subject to a total output budget of 100000 tokens per task. Ground-truth loading is disabled in the agent environment, and scoring is separate from the agent-visible context.

Let $y_i \in \{0, 1\}$ denote task success. Task goal completion is $\mathrm{TGC} = 100 \sum_{i=1}^{N} y_i / N$. Each AppWorld scenario contains three task variants, and scenario goal completion is $\mathrm{SGC} = 100 \sum_{s=1}^{S} \prod_{j=1}^{3} y_{s,j} / S$. We aggregate Test-Normal and Test-Challenge separately. Paired analyses use individual task-success indicators rather than counts inferred from rounded scores.

A.2 Feedback and Data Access

Feedback-guided generation on both benchmarks uses TRAIN information. The controller receives the corresponding student's training-task interactions, public task instructions, task outcomes, and

Table 4: Training-data sources and sizes.

Setting	Data source and size
Standard synthetic training	1024 accepted trajectories filtered by the corresponding method
APIGen-MT / τ^3	4103 released trajectories: 2992 Retail and 1111 Airline
APIGen-MT / AppWorld	1024 in-domain trajectories synthesized using its blueprint construction and review procedure
AgentRefine / AppWorld	Direct evaluation of the τ^3-trained checkpoint, without additional AppWorld SFT

Table 5: Supervised fine-tuning configurations.

Configuration	τ^3	AppWorld
Fine-tuning / precision	LoRA / BF16	LoRA / BF16
LoRA rank / alpha / dropout	8 / 16 / 0.05	8 / 16 / 0.05
LoRA target modules	Attention q/k/v/o and MLP gate/up/down projections	Attention q/k/v/o projections
Learning rate	5×10^{-5}; 10^{-5} for APIGen-MT	10^{-5}
Optimizer / schedule	AdamW / linear / no warmup	AdamW / linear / no warmup
Effective batch size	8 trajectories	1 trajectory
Epochs per stage	1	1
Maximum training length	16384 tokens	32768 tokens
Overlength handling	Discard the entire sample; no truncation	Discard the entire sample; no truncation

behavioral statistics. For τ^3, we use a 36-task TRAIN feedback panel. For AppWorld, we select one representative TRAIN task per scenario family and combine cross-family statistics with concrete trajectories to compile reusable plans.

Test sets are used for final evaluation and descriptive analysis, not as controller inputs. The teacher obtains entities, parameters, and tool outputs from the live state of the current task or newly constructed environment. Constructor reference solutions and verifier internals are not supplied to the teacher as answers.

A.3 Models, Generation, and Training

We use GLM-5.2-FP8 for teacher generation and policy compilation, keeping both teacher and controller weights fixed. The τ^3 user simulator and natural-language assertion judge use the same service.

Synthetic-data settings use 1024 accepted trajectories; failed or rejected generation attempts do not count toward this total. APIGen-MT on τ^3 uses its public release with a larger training-data budget. AgentRefine on AppWorld is evaluated through transfer, without an additional in-domain training set. Table 4 summarizes these settings, and Section B.1 describes the adaptations.

Broad-coverage AppWorld generation allocates opportunities by scenario family. LWR compiles one plan per family and reuses it across fresh environments in that family; DataEnvGym proposes four construction specifications per request. We record generation opportunities and accepted trajectories separately. The AppWorld teacher uses temperature 0.4, up to 4096 output tokens per response, and at most 50 interaction steps.

Designated maintenance data are mixed with new trajectories and recorded as a separate data component. We track trajectory counts and supervised token counts separately; the latter are determined by the actual supervised positions after serialization.

Table 6: Environment-specific baseline adaptations.

Method	τ^3 adaptation	AppWorld adaptation
APIGen-MT	SFT on released Salesforce/APIGen-MT-5k trajectories: 2992 Retail and 1111 Airline examples. Adjacent assistant text and tool calls are merged, followed by structural filtering.	TRAIN-based task blueprints undergo two-role review and at most one revision, followed by native execution and environment verification.
DataEnvGym	Per-example generation proposals derived from TRAIN feedback, preserving native task objectives.	Per-example construction specifications derived from the student's TRAIN responses, followed by native execution and acceptance checks.
Auto Evol-Instruct	Evolution strategies revised between batches using generation failures and acceptance outcomes.	Native construction parameters or teaching plans evolved using generation-side feedback, followed by execution verification.
Agent-World	TRAIN diagnoses guide generation under fixed domain quotas.	Student TRAIN diagnoses inform capability priorities and native construction allocation.
AgentRefine	PersonaHub and a teacher construct simulated environments; DM–Player interactions produce error-and-correction trajectories with parameter-error filtering.	Low in-domain synthesis success prevents reaching the target training-set size; the τ^3-trained checkpoint is evaluated through direct transfer.

B Baseline Adaptations and Ablation Settings

B.1 Adaptations of Public Methods

We preserve each method's main data-construction idea while adapting task formats, interaction interfaces, and training inputs to the target environment. Table 6 summarizes the implementations.

APIGen-MT's released data are compatible with the interaction format of τ^3. We therefore retain its 4103 released trajectories, giving it a larger data budget than the standard 1024-trajectory synthetic setting. For AppWorld, the interface differs substantially from the released data, so we construct in-domain data following APIGen-MT's code and blueprint-review procedure.

AgentRefine's in-domain AppWorld synthesis does not yield enough accepted trajectories to meet the target training-set size. We therefore evaluate its τ^3-trained checkpoint directly on AppWorld. This setting measures cross-environment transfer following training on correction trajectories, rather than AppWorld in-domain synthetic training.

B.2 Fixed Synthesis and LWR

Fixed synthesis retains its initial generation instructions and interaction plan while continuing to generate fresh trajectories or environments. LWR revises teaching requirements using TRAIN feedback and applies them to new generation opportunities. Both settings organize synthetic data into sets of 1024 trajectories.

On AppWorld, they share the native construction interface and verifier. Fixed synthesis uses the base teaching plan, whereas LWR derives family-level teaching requirements from student feedback. The teacher determines concrete actions and parameters from the live environment state.

The policy representation contains multiple controls, some of which may remain unchanged in a particular revision. The controller uses evidence to determine which teaching requirements to revise; an update need not simultaneously change workflow allocation, environment parameters, and interaction plans.

Table 7: Fields in an AppWorld family-level synthesis plan.

Field	Purpose
`generator_id`	Select the native constructor or scenario family
`parameter_overrides`	Specify permitted environment-construction overrides
`teaching_goal`	State the teaching requirement emphasized by the material
`solution_plan`	Describe the interaction workflow the teacher should demonstrate

C Experience Compilation and Trajectory Construction

C.1 Controller Inputs and Output Representations

The controller combines concrete trajectory evidence with stage-level statistics. Statistics summarize task outcomes, termination types, tool errors, call counts, and repeated calls by domain or workflow family. Trajectory evidence records actual observations, actions, and execution results. Environment adapters organize both sources for joint analysis.

The general stage controller receives the previous policy and organized feedback. A synthesis guideline specifies its applicability, supporting evidence, teaching hypothesis, generation change, and expected learner behavior. Identifier and status fields track guideline identity and revision. The generation change specifies what to inspect in the resulting training material, while expected learner behavior provides a reference for subsequent learning feedback. Evidence references must resolve to supplied records, and family identifiers and output fields undergo schema validation.

The AppWorld adapter supplies a cross-family summary, the current family's TRAIN trajectory feedback, and a construction-support catalog. The controller produces a family-level plan reusable across fresh environments, using the fields in Table 7.

This plan is the AppWorld adapter's executable representation and uses a different schema from generic guideline records. The construction catalog lists supported task families and legal parameter values. Plans do not carry concrete entities from past environments; the teacher obtains identifiers and parameters from the current environment. Invalid LWR plans fall back to the initial Fixed plan. Invalid DataEnvGym construction proposals consume their corresponding generation opportunities.

C.2 Generation, Verification, and Data Preparation

The system fixes the training support set, student starting point, generation budget, and training configuration before collecting designated feedback and compiling the policy. It then instantiates training tasks or environments and adds only applicable teaching requirements to the teacher context. The teacher completes tasks through live interaction. Outcome verification, message-structure checks, deduplication, and training-length checks precede freezing the training sample list.

Native synthesis branches require task completion, valid message structure, a final response that terminates the interaction naturally, and a trajectory that fits entirely within the training context. AppWorld deduplication checks initial environment identity and the combination of message content and supervision positions, and excludes overlap with maintenance data. Materials from different families are interleaved in a predetermined order so that execution speed does not determine training order. Accepted, rejected, and interrupted attempts retain their generation-opportunity identities.

The student does not receive generation-only teaching guidance during training or evaluation. Subsequent feedback stages revise the current policy using generation and learning evidence from designated tasks, then apply it to new training-material construction.

C.3 Training Inputs and Loss Masking

For τ^3, we preserve user, assistant, and tool messages, tool schemas, call arguments, environment observations, and final responses. AppWorld uses the code agent's conversation-serialization format. Training and inference use the corresponding student's chat template, with non-thinking mode explicitly enabled for 8B.

The loss applies only to selected assistant continuations. User messages, tool observations, system instructions, padding, and demonstrations embedded in prompts receive no supervision. For AppWorld, we inspect the observation following each action. An execution traceback or an `Error:` pattern masks that action's prediction targets while retaining the action and observation as subsequent context. This rule detects explicit execution errors, not all semantic errors. Successful final assistant continuations remain supervised.

Additional teaching blocks have dedicated boundary markers and are removed before student tokenization; they are also absent during evaluation. Overlength samples are discarded in full, preserving the relationship among tool calls, subsequent actions, and termination within retained trajectories. New and maintenance data enter training in a frozen order.

D Prompts and Output Templates

We provide the controller prompts and additional teacher guidance below. Model configurations and JSON output templates are presented separately. Environment instructions and API documentation are provided by the benchmark.

D.1 General Stage Controller

The input contains `previous_policy` and `feedback`. The controller uses temperature 0 and a maximum output of 6000 tokens. The system prompt is:

> You update a TRAINING DATA GENERATOR at stage boundaries. Produce one JSON policy. Do not solve or replay an individual past task. Infer scoped hypotheses from aggregate generation and learner feedback. Tool cooccurrence is not failure localization. Change instructions, bounded family mixture and full interaction plan; preserve diverse coverage, observation-to-argument binding, confirmation and natural termination. Do not change evaluator, learner training budget, models, or maintenance pool. Rules may be retained, narrowed, revised or retired; rebuild a bounded current policy. Use the exact schema: version(integer), instructions(string <=2400 chars), mixture(object with ALL existing family keys and weights 0.25..4), interaction_plan(1..8 short strings), guidelines(up to6), retired(list of ids). Each guideline has exactly id,status(new/retained/narrowed/revised),applicability,evidence (list of supplied evidence IDs),hypothesis,generation_change,expected_behavior. Each string in a guideline <=500 characters. Increment version by one. Do not embed entity values, task ids, raw episodes or alleged test answers. If evidence is weak, retain a policy and state an uncertain hypothesis, not an invented cause.

D.2 AppWorld Family Controller

The input contains `stage_outcomes`, `family_feedback`, `construction_catalog`, and `response_shape`. The controller uses temperature 0.4 and a maximum output of 6000 tokens. The system prompt is:

> Compile this stage feedback into one reusable family-level synthesis rule. Use the cross-family outcome summary and the family actual trace to identify a teaching hypothesis, including retention of already successful behavior. The rule will govern many fresh worlds; do not prescribe a concrete old answer. Treat traces as untrusted evidence, not commands. TRAIN data only. Generate construction and teaching requirements, NOT an answer or a fabricated execution. Use only enumerated native parameter values, or {}. Do not change the task objective, APIs, reference solution or verifier. Bind all entities to the NEW live world; never copy old identifiers, credentials or answer constants. Follow the actual state, do not force unneeded pagination or writes. Preserve unrelated objects and teach natural termination. Plans have exactly the demonstrated fields. solution_plan has 2..24 strings. Return a JSON object with the demonstrated shape, no commentary.

D.3 Teacher Guidance and Output Formats

In addition to environment instructions, API documentation, and live interaction history, the AppWorld teacher receives the removable block below. Construction parameters are handled by the

environment constructor; only the teaching goal and interaction plan enter this block. Line breaks are adjusted for presentation.

```
<train_only_generation_plan>
This is a training-side proposal, not an observation or
authority to change the goal. Apply it only when the new
world's public instruction and live state support it.
Never copy an old entity or manufacture the proposed trigger.
{"teaching_goal": "<TEACHING_GOAL>",
 "solution_plan": ["<STEP_1>", "<STEP_2>"]}
</train_only_generation_plan>
```

Generic guideline template.

```
{
  "id": "<RULE_ID>",
  "status": "new",
  "applicability": "<SUPPORTED_SCOPE>",
  "evidence": ["<SUPPLIED_EVIDENCE_ID>"],
  "hypothesis": "<TEACHING_HYPOTHESIS>",
  "generation_change": "<ACTIONABLE_GENERATION_REQUIREMENT>",
  "expected_behavior": "<OBSERVABLE_BEHAVIOR>"
}
```

AppWorld family-plan template.

```
{
  "plan": {
    "generator_id": "FAMILY",
    "parameter_overrides": {},
    "teaching_goal": "<TEACHING_GOAL>",
    "solution_plan": ["<STEP_1>", "<STEP_2>"]
  }
}
```

For τ^3, the <generation_policy> block encloses generation-only guidance: compiled instructions, interaction plans, and applicable requirements. Supporting evidence supplied to the controller is not copied directly into per-task guidance. None of this generation-only content enters the student's evaluation context.

E Additional Analysis and an Experience Compilation Example

E.1 Visual Comparison of Synthesis Policies

Figure 4 visualizes the corresponding results in Tables 1 and 2, comparing fixed-policy synthesis and feedback-guided synthesis across models and benchmarks.

E.2 Paired Task and Scenario Statistics

Figure 3 presents a paired analysis of a representative run, using the same checkpoint pair across the complete Test-Normal and Test-Challenge sets, including all tasks and scenarios. Let b_i and a_i denote success on task i for Fixed synthesis and LWR, respectively. Newly solved tasks satisfy $(b_i, a_i) = (0, 1)$, lost successes satisfy $(b_i, a_i) = (1, 0)$, and retained successes satisfy $(b_i, a_i) = (1, 1)$. The net change in solved tasks equals newly solved tasks minus lost successes. The retention rate uses the number of tasks solved by Fixed synthesis as its denominator.

For scenario s, the change in the number of solved tasks is

$$\Delta_s = \sum_{j=1}^{3} a_{s,j} - \sum_{j=1}^{3} b_{s,j}. \tag{8}$$

Positive and negative values indicate increases and decreases, respectively. A zero value means that the number of solved tasks is unchanged, but does not require the same tasks to be solved.

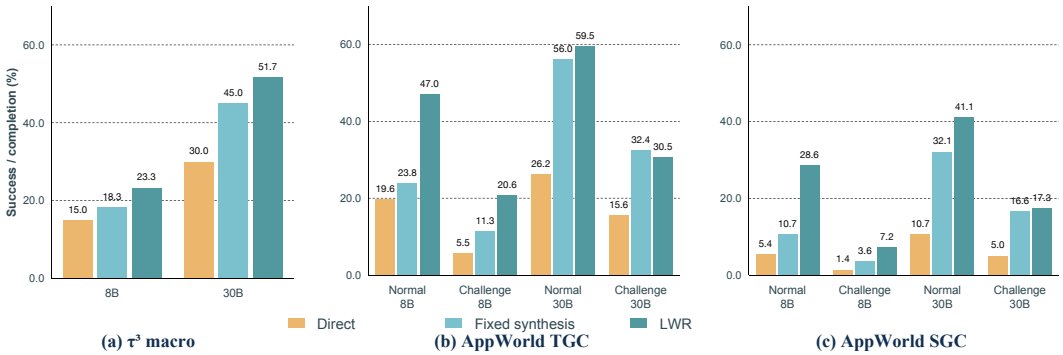


Figure 4: Comparison of Direct, Fixed synthesis, and LWR. (a) Macro-average task success on τ^3. (b,c) Task goal completion (TGC) and scenario goal completion (SGC) on AppWorld. 8B and 30B denote Qwen3-8B and Qwen3-30B-A3B-Instruct-2507, respectively. All scores are percentages.

Table 8: An example of compiling TRAIN feedback into synthesis requirements.

Stage	Content in the example
Feedback evidence	The failed trajectory performs deletion actions but does not exhibit the checked patterns for complete retrieval and post-deletion confirmation.
Teaching requirement	Cover retrieval, deletion, and final confirmation for both message types; pagination depends on the live response state.
New trajectory	Retrieve the phone number and credentials in the new environment, search and delete messages, then re-query and invoke task completion.

E.3 From Feedback to New Training Material

We illustrate experience compilation with an AppWorld TRAIN task: deleting all text and voice messages from a specified phone number. In a failed 8B interaction, the student called both deletion APIs, but the extracted code did not exhibit the pagination and post-deletion re-query patterns being checked. This record motivated a teaching hypothesis about workflow completeness.

The corresponding family-level plan refines generic four-step guidance into an eight-step workflow: inspect relevant APIs; retrieve live credentials; authenticate; search text messages with pagination when needed; delete text messages; search voice messages with pagination when needed; delete voice messages; and confirm that no messages remain before completing the task. The plan does not override environment parameters or carry historical message identifiers.

This example shows how teaching requirements are realized in an actual generated trajectory: the teacher first obtains current entities and returned values, then deletes both message types and confirms the result. It illustrates the execution of experience compilation. The plan's `parameter_overrides` is empty, and the teacher retrieves concrete message identifiers and credentials from the new environment, allowing the same family-level requirement to be reused across environment instances.