KNOW WHAT YOU DON'T KNOW: AGENTIC DATA SYNTHESIS BY EVOLVING GENERATION SPACE FOR CLOSED-LOOP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The continued improvement of large language models increasingly depends on synthetic experience, yet effective data synthesis for interactive agents requires not only executable and verifiable trajectories, but also interactions that provide learning value for the current task agent. Existing methods increasingly incorporate environment feedback, data validation, or learner failures, but still face a *feedback-to-decision gap*: observed weaknesses do not directly specify how future synthesis should change, or whether they can be represented by the current synthesis space. To address this gap, we propose EVOSYN, a learning-feedback-driven multi-agent framework that organizes synthesis around a revisable state specifying *what* interaction factors can be deliberately generated and *where* synthesis effort should be allocated. A role-specialized multi-agent system translates learner feedback into synthesis revisions, while a Designer allocates synthesis directions and a Planner grounds them into executable experience. This process enables *within-space adaptation* toward weak or under-covered regions and *synthesis-space expansion* that promotes newly discovered factors into explicit generation controls. Experiments on τ^2-Bench and CAR-Bench show that EVOSYN improves average task success by 13.3 points over the strongest synthesis baseline, while producing higher-quality and structurally broader synthetic experience and discovering controllable factors missed without learner feedback. Code is available at `https://anonymous.4open.science/r/evosyn_repo-4DB0`.

1 INTRODUCTION

Synthetic data has become a central mechanism for pushing large language models beyond the limits of human annotation (Wang et al., 2023; Xu et al., 2024; Shao et al., 2024). In reasoning domains, a synthetic example can often be formulated as a self-contained problem with a verifiable answer or solution trace, allowing synthesis to focus on concepts or reasoning patterns the model has not yet mastered (Gou et al., 2024; Li et al., 2024). Interactive agents impose an additional challenge (Pahuja et al., 2025; Wang et al., 2026b). A valid agentic example must be feasible in an environment, executable through tools, consistent with domain policies, and verifiable through state changes or task outcomes (Chen et al., 2026; Wang et al., 2026b). Yet validity alone does not imply **learning utility**. An executable and correct task may simply repeat behavior the current agent already handles, while interactions that expose its remaining weaknesses remain absent from training (Huang et al., 2025; Liang et al., 2026). Thus, the central question in agentic data synthesis is not simply how to generate valid tasks, but *what interactions are most useful for the current learner to experience next*.

Recent work has made substantial progress toward feedback-driven synthesis (Chen et al., 2025; Liang et al., 2026). Environment-grounded methods such as AutoPlay (Ramrakhya et al., 2025) and Trajectory2Task (Wang et al., 2026b) use interaction feedback to construct executable tasks; data-centric methods such as EigenData (Chen et al., 2026) audit and repair generated samples; and learner-adaptive methods such as R-Zero (Huang et al., 2025) and Socratic-Zero (Wang et al., 2025b) generate new problems from model failures or capability boundaries. These methods improve feasibility, data quality, and learner adaptivity, but using feedback is not the same as knowing what synthesis decision it should support. A common implicit assumption is that the learner's re-

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.

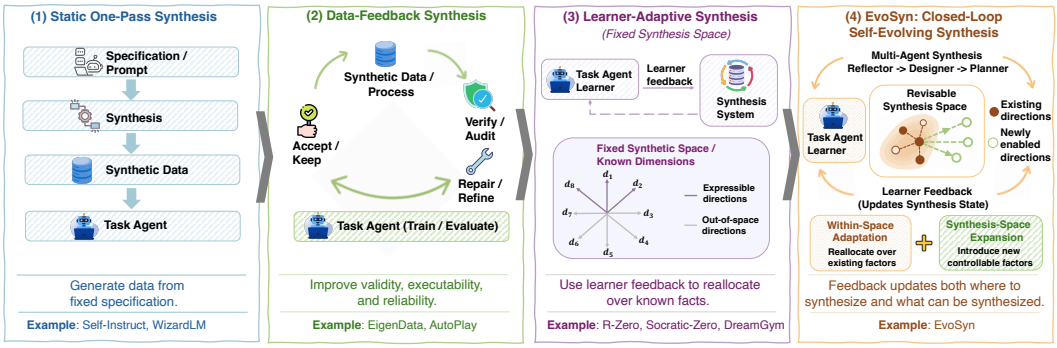


Figure 1: From static synthesis to closed-loop self-evolving agentic synthesis. Existing paradigms improve data quality or adapt within a fixed synthesis space. In contrast, EvoSyn uses learner feedback to revise both the allocation over known factors and the synthesis space itself by introducing new controllable interaction factors.

maining weaknesses are already expressible within the current synthesis space, such as predefined task categories or interaction dimensions. Yet observed failures are not themselves actionable synthesis coordinates: they reveal where the learner struggles without necessarily identifying which interaction factors should be controlled in future generation (Yang et al., 2026; Kwan et al., 2026). We refer to this missing bridge as the **feedback-to-decision gap**: translating learner feedback into actionable synthesis decisions while determining whether the current synthesis space can represent the learner's remaining weaknesses.

As illustrated in Figure 1, closing this gap requires the synthesis process itself to evolve together with the task agent (Cheng et al., 2025). Learner feedback can expose two fundamentally different forms of mismatch. When the learner's weakness is already expressible in the current synthesis space, the problem is one of **within-space adaptation**: synthesis effort should be reallocated toward under-covered or challenging regions (Yue et al., 2026). When the weakness is not captured by existing synthesis coordinates, reallocation alone is insufficient; the system must instead perform **synthesis-space expansion**, promoting previously unrepresented but actionable interaction factors into explicit controls for future generation (Li et al., 2026). For example, failures on refund or flight-change tasks may actually stem from numerical computation, policy-dependent action ordering, or the coordination of multiple state-changing operations. If such factors are absent from the synthesis space, generating more examples from the same task categories may increase data volume without addressing the underlying capability gap. Self-evolving agentic synthesis should therefore adapt not only the distribution over what can already be generated, but also the representation of what can be generated at all.

To address these challenges, we propose EvoSyn, a learning-feedback-driven multi-agent framework for self-evolving agentic data synthesis. EvoSyn organizes generation around a structured and revisable synthesis space over controllable interaction factors, which serves both as a coordinate system for learner diagnosis and as a control surface for future generation. Within this space, a Designer Agent selects synthesis directions and allocates generation budget, while a Planner Agent grounds these directions into environment-consistent task blueprints. A Generator Agent and rollout teacher then instantiate executable tasks and successful trajectories, which are further filtered by a Verifier Agent to ensure valid supervision. After task-agent training, a Reflector Agent analyzes performance, coverage, and failure signals and maps them back to the synthesis space. This feedback drives either **within-space adaptation**, which reallocates synthesis effort over existing coordinates, or **synthesis-space expansion**, which introduces new evidence-grounded and actionable factors as future generation controls. Through this closed loop, EvoSyn enables the task agent and its synthesis space to co-evolve through learning feedback.

Our main contributions are as follows:

- We identify a **feedback-to-decision gap** in self-evolving agentic data synthesis and formulate self-evolving synthesis over a **revisable synthesis space** that adapts with the learner.

- We propose EVOSYN, which combines **within-space adaptation** to reallocate synthesis effort over existing factors with **synthesis-space expansion** to introduce new actionable factors revealed by learner feedback.
- Experiments on τ^2-Bench and CAR-Bench improve average task success by 13.3 points over the strongest synthesis baseline, while analyses show that learner feedback drives allocation shifts and reveals useful synthesis factors missed without it.

2 RELATED WORK

Synthetic Data for Self-Evolution. Synthetic data has become increasingly important for LLM post-training, as it enables models to improve beyond the scale and coverage of manually annotated supervision. Prior work in this area can be viewed along a progression from offline data construction to learner-aware synthesis. Early methods generate instruction-following or reasoning corpora from seed demonstrations, teacher models, unlabeled corpora, or problem transformations, including instruction synthesis, instruction evolution, and mathematical data augmentation Wang et al. (2023); Xu et al. (2024); Li et al. (2023); Yu et al. (2023); Luo et al. (2025). Another line of work emphasizes verifiability and data quality, using answer checkers, tool-integrated traces, filtering, or curated large-scale reasoning pipelines to ensure that generated supervision can reliably support post-training Zelikman et al. (2022); Gou et al. (2024); Shao et al. (2024); Li et al. (2024). More recent self-evolving methods place data generation and model learning in a feedback loop, where solver failures, teacher diagnoses, or capability boundaries are used to generate increasingly challenging tasks Huang et al. (2025); Wang et al. (2025b); Yue et al. (2026); Kwan et al. (2026).

Feedback-Driven Agentic Data Synthesis. Synthetic data for interactive agents introduces additional requirements beyond problem–answer correctness: tasks must be feasible in an environment, executable through tools, consistent with domain policies, and verifiable through state changes or task outcomes. Existing work addresses these requirements through several complementary forms of feedback. Environment-grounded methods use exploration, state replay, or valid trajectories to construct executable tasks and broaden interaction coverage Pahuja et al. (2025); Sun et al. (2025); Ramrakhya et al. (2025); Wang et al. (2026b;a). Data-centric methods use checkers, validators, rejection reasons, or learned experience models to improve the validity and quality of tool-use or multi-turn synthetic data Chen et al. (2026; 2025). A further group of agent-evolution frameworks studies how environments, curricula, workflows, memories, or tool-use experiences can be revised over time Yang et al. (2026); Dong et al. (2026); Dai et al. (2026); Zhai et al. (2025); Wu et al. (2025); Zhang et al. (2026). These directions collectively establish the importance of feedback in agentic data generation, while leaving open how different feedback signals should be converted into reusable synthesis decisions rather than only valid instances, repaired samples, or broader trajectory collections. We provide a more complete discussion of related work in Appendix A.2 and C.

3 PROBLEM FORMULATION

We study synthetic data generation for improving language-model agents in interactive tool-use environments. Let $\mathcal{E}$ denote an environment with domain policies, tool interfaces, state constraints, and an automatic evaluator, and let π_{θ_r} denote the task agent at synthesis round r. At each round, the synthesis process generates a verified dataset $\mathcal{D}_r$ of executable and successful task–trajectory pairs, which is used to update the task agent:

$$\pi_{\theta_{r+1}} = \mathrm{Train}\left(\pi_{\theta_r}, \mathcal{D}_{\leq r}\right), \qquad \mathcal{D}_{\leq r} = \bigcup_{t=1}^{r} \mathcal{D}_t. \tag{1}$$

Unlike one-pass synthesis, however, the utility of $\mathcal{D}_r$ depends on the current learner: valid data may provide little learning signal if it does not target the interaction patterns where the agent remains weak. We therefore characterize synthesis at round r by a *revisable synthesis state* $S_r = (\mathcal{O}_r, \alpha_r)$, where the synthesis space $\mathcal{O}_r$ specifies *what interaction properties can be deliberately synthesized*, and α_r specifies *where synthesis effort is allocated within that space*. After training and evaluation, learner feedback F_r must be translated into a new synthesis state, $S_{r+1} = \Psi(S_r, F_r) = (\mathcal{O}_{r+1}, \alpha_{r+1})$, where Ψ denotes the feedback-driven synthesis revision process. This mapping captures the feedback-to-decision gap: observed learner behavior must be converted into actionable decisions. A complete formulation is provided in Appendix A.1.

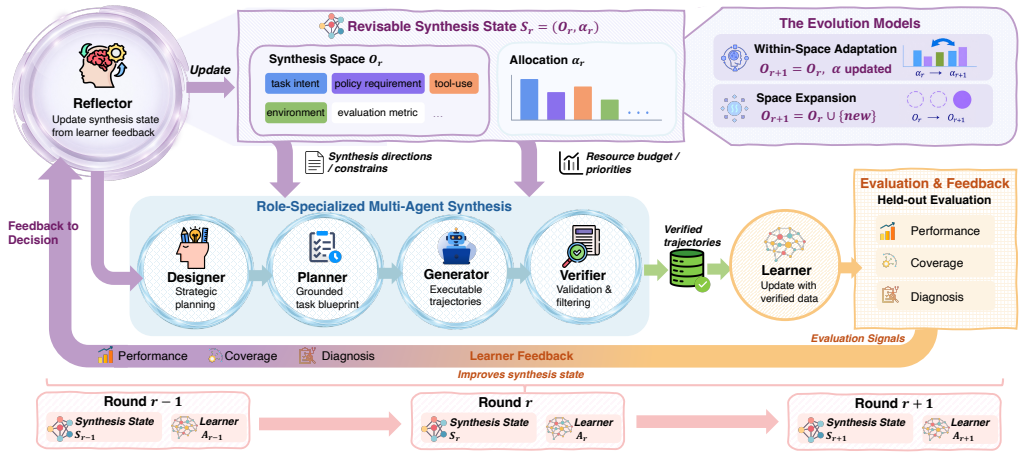


Figure 2: Overview of EvoSyn. Each round instantiates a closed-loop synthesis–learning process in which a revisable synthesis state mediates learner diagnosis and generation control, role-specialized agents realize verified experience, and feedback drives within-space adaptation or synthesis-space expansion to co-evolve synthesis with task-agent learning.

4 Method

EvoSyn is a multi-agent framework for learning-guided, self-evolving agentic data synthesis. As illustrated in Figure 2, it maintains a revisable synthesis state that guides grounded data generation and is updated from learner feedback, forming a closed synthesis–learning loop. Additional details of the complete closed-loop framework are provided in Appendix A.3.

4.1 Revisable Environment-Grounded Synthesis Space

The synthesis state $S_r = (\mathcal{O}_r, \alpha_r)$ requires a structured representation of what experience can be deliberately generated. Because agentic tasks must respect tool preconditions, domain policies, environment states, and evaluator requirements, EvoSyn instantiates $\mathcal{O}_r$ as a *revisable environment-grounded synthesis space* over controllable interaction factors. Formally,

$$\mathcal{O}_r = \{d_1^r, \ldots, d_{m_r}^r\}, \qquad d_j^r = (k_j^r, V_j^r, c_j^r), \tag{2}$$

where k_j^r denotes the factor name, V_j^r its finite set of admissible values, and c_j^r specifies how the factor constrains or guides generation. Typical factors include task intent, policy requirements, tool-use patterns, expected outcomes, and execution complexity, while $\mathcal{O}_1$ is initialized from environment priors such as tool schemas, domain policies, state constraints, task specifications, and evaluator requirements.

A *synthesis direction* specifies a controllable region of this space:

$$z = \{k_j^r = v_j\}_{j \in \mathcal{I}}, \qquad v_j \in V_j^r, \tag{3}$$

where $\mathcal{I} \subseteq \{1, \ldots, m_r\}$ and $z \in \mathcal{Z}(\mathcal{O}_r)$ represents a high-level interaction target rather than a concrete task. The Designer operates over such directions when allocating synthesis effort, while the Planner grounds each selected direction into an environment-consistent task blueprint.

Importantly, $\mathcal{O}_r$ serves as a shared representation for both *learner diagnosis* and *generation control*: its coordinates provide an interpretable basis for measuring learner performance and data coverage, while also specifying interaction properties that can be deliberately targeted in future synthesis. The space is therefore revisable: when current coordinates are sufficient, EvoSyn updates α_r to redistribute synthesis effort; when learner feedback reveals a capability-relevant property outside the current representation, it expands $\mathcal{O}_r$ with a new controllable factor.

4.2 FEEDBACK-GUIDED SYNTHESIS EVOLUTION

The role of learning feedback in EVOSYN is not merely to identify failed examples, but to determine how the synthesis process itself should change. After each round of task-agent training, we summarize learner behavior and synthesized experience as

$$F_r = \text{Feedback}\left(\pi_{\theta_{r+1}}, \mathcal{D}_{\leq r}, \mathcal{O}_r, \mathcal{E}\right), \tag{4}$$

where F_r combines *performance*, *coverage*, and *failure-diagnostic* signals. Performance identifies weak interaction regions, coverage reveals whether these regions have received sufficient synthesis effort, and failure diagnostics capture recurrent patterns that may not yet correspond to existing synthesis coordinates. A Reflector Agent converts this evidence into the next synthesis state:

$$(\mathcal{O}_{r+1}, \alpha_{r+1}) = \text{Reflector}\left(\mathcal{O}_r, \alpha_r, F_r\right). \tag{5}$$

Rather than repairing individual failures, the Reflector operates at the synthesis level, deciding whether the observed weakness can be addressed within the current synthesis space or requires revising the space itself.

Within-Space Adaptation. When the learner's weakness is already expressible by factors in $\mathcal{O}_r$, EVOSYN keeps the synthesis space fixed and updates only the allocation:

$$\mathcal{O}_{r+1} = \mathcal{O}_r, \qquad \alpha_{r+1} = \text{Reallocate}\left(\alpha_r, F_r; \mathcal{O}_r\right). \tag{6}$$

The Reflector shifts synthesis effort toward directions with poor performance or insufficient coverage, while reducing effort on well-covered or saturated regions. The resulting allocation is then realized by the Designer Agent in the next synthesis round.

Synthesis-Space Expansion. When recurrent failures reveal a capability-relevant property that is not adequately represented in $\mathcal{O}_r$, reallocation alone is insufficient. The Reflector instead identifies candidate interaction factors from diagnostic patterns in learner failures and synthesized trajectories. A factor is promoted only if it is *evidence-grounded*, *interpretable and finite-valued*, *non-redundant* with existing coordinates, and *actionable* by the downstream synthesis process. For an accepted factor $d_{\text{new}} = (k_{\text{new}}, V_{\text{new}}, c_{\text{new}})$, the synthesis space is expanded as

$$\mathcal{O}_{r+1} = \mathcal{O}_r \cup \{d_{\text{new}}\}. \tag{7}$$

The new factor becomes an explicit synthesis coordinate that can be measured, budgeted for, and deliberately instantiated in subsequent rounds.

Together, these two updates allow EVOSYN to adapt both *where* synthesis effort is placed and, when necessary, *what* the synthesis process can explicitly represent.

4.3 ROLE-SPECIALIZED MULTI-AGENT SYNTHESIS

The synthesis state $S_r = (\mathcal{O}_r, \alpha_r)$ specifies what interaction factors should be targeted and how synthesis effort should be distributed, but it does not directly define executable training examples. EVOSYN therefore uses a role-specialized multi-agent process to progressively translate these high-level synthesis decisions into environment-grounded and verified experience.

Designer: Strategic Synthesis Planning. The Designer Agent operates over the synthesis space rather than individual examples. Given $\mathcal{O}_r$ and α_r, it constructs a generation plan

$$\mathcal{G}_r = \{(z_i, b_i)\}_{i=1}^{K} = \text{Designer}\left(\mathcal{O}_r, \alpha_r\right), \tag{8}$$

where $z_i \in \mathcal{Z}(\mathcal{O}_r)$ denotes a synthesis direction and b_i its allocated budget. The Designer thus operationalizes the revised state into decisions about *what regions to target* and *how much experience to generate*, separating round-level synthesis planning from the Reflector's meta-level revision.

Planner: Grounding Synthesis Intent. A synthesis direction specifies desired interaction properties but is not itself an executable task. The Planner Agent bridges this abstraction gap by producing an environment-grounded blueprint

$$q_i = \text{Planner}\left(z_i, \mathcal{O}_r, \mathcal{E}\right), \tag{9}$$

which specifies the user goal, required environment conditions, relevant policy constraints, expected tool interactions, and target outcome. The Planner therefore serves as a grounding layer between synthesis-level intent and environment-level task realization.

Executable Realization and Verification. Given q_i, the Generator Agent instantiates a concrete task $x_i = \text{Generator}(q_i, \mathcal{E})$, and a rollout teacher interacts with the environment to produce a candidate successful trajectory $\tau_i = \text{RolloutTeacher}(x_i, \mathcal{E})$. A Verifier Agent then checks executability, tool-call validity, policy compliance, state consistency, and task success, retaining only valid task–trajectory pairs:

$$\mathcal{D}_r = \{(x_i, \tau_i) \mid \text{Verifier}(x_i, \tau_i, \mathcal{E}) = 1\}. \tag{10}$$

This role specialization forms a coarse-to-fine synthesis process: the Designer performs strategic allocation over the controllable space, the Planner grounds abstract targets into feasible task structures, and the Generator, rollout teacher, and Verifier realize and validate executable experience.

4.4 Task-Agent Learning and Closing the Loop

The verified task–trajectory pairs $\mathcal{D}_r$ are converted into supervision for the downstream task agent while preserving user messages, tool calls, environment observations, and target responses. To incorporate newly synthesized experience without discarding previously covered interactions, EvoSyn trains on accumulated verified data $\mathcal{T}_{\le r} = \bigcup_{t=1}^{r} \mathcal{T}_t$ using supervised fine-tuning:

$$\theta_{r+1} = \arg\min_{\theta} \sum_{(s,a)\in\mathcal{T}_{\le r}} -\log \pi_\theta(a \mid s), \tag{11}$$

where s denotes the interaction context and a the target response or tool action.

The updated task agent $\pi_{\theta_{r+1}}$ is then evaluated on a held-out diagnostic set in the target environment. Its performance, together with synthesis coverage and failure diagnostics, forms the feedback F_r described in Section 4.2, which the Reflector uses to revise the next synthesis state $S_{r+1} = (\mathcal{O}_{r+1}, \alpha_{r+1})$. The resulting closed loop is $S_r \to \mathcal{D}_r \to \pi_{\theta_{r+1}} \to F_r \to S_{r+1}$. Thus, each round updates the learner through synthesized experience while using the learner's resulting behavior to reshape future synthesis, enabling the task agent and synthesis process to co-evolve over rounds.

5 Experiments

5.1 Experimental Setup

Datasets and metrics. We evaluate EvoSyn on τ^2-Bench and CAR-Bench, both of which provide executable environments and automatic task-level evaluators. Across all benchmarks, we report task success rate under the corresponding environment evaluator. **LLM backbones.** We include closed-source models Claude Haiku 4.5, Claude Sonnet 4.5, Claude Opus 4.5, GPT-5.2, Gemini-3 Flash and Gemini-3 Pro, and open-source Qwen3 models including 8B, 14B and 30B. Qwen3-8B is the default task-agent backbone, Qwen3-14B and Qwen3-30B are used for task-agent scaling. Qwen3-30B, Qwen3-235B and Claude Opus 4.5 are used as synthesis LLMs. **Experimental protocol.** Full experimental details, hyperparameters and settings are provided in Appendix B.1.

5.2 RQ1: Overall Effectiveness

Table 1 shows that EvoSyn consistently outperforms multiple classes of competing approaches across all evaluated environments. With Opus 4.5 as the synthesis LLM, EvoSyn-Final-SFT achieves an average success rate of 77.8%, substantially exceeding all general agentic synthesis baselines as well as the stronger synthesis variants, including Monolithic, Fixed-State, R-Zero-style, and AutoPlay-style synthesis. It also outperforms all directly evaluated open-source foundation models despite training only a Qwen3-8B task agent, highlighting the value of targeted synthetic experience beyond model scale alone. The advantage is already visible after the first synthesis round, where EvoSyn-R1 reaches 70.9%, while feedback-guided multi-round evolution further improves performance to 77.8%, demonstrating additional gains from continually revising what experience the learner receives. Importantly, the same trend holds with both Qwen3-30B and Qwen3-235B synthesis backends, indicating that the framework is not tied to a particular frontier generator. Moreover, Appendix B.2 Table 4 shows that the benefit persists as the downstream learner scales: EvoSyn-Final-SFT reaches 80.9% with Qwen3-14B and 86.0% with Qwen3-30B task agents.

Table 1: Overall task success rate (%) on agentic benchmarks. For a consistent comparison across synthesis approaches, all synthesis methods use Qwen3-8B as the downstream task agent. EVOSYN-R1 uses first-round data, while EVOSYN-Final-SFT uses accumulated multi-round data.

Method	Synthesis LLM	τ^2-Bench			CAR-Bench	Avg.
		Airline	Retail	Telecom		
Frontier Closed-source Models						
Claude Opus 4.5	–	79.3	91.8	88.0	82.0	85.3
GPT-5.2	–	83.0	81.6	89.7	74.0	82.1
Gemini-3 Pro	–	80.5	75.9	91.0	67.3	78.7
Open-Source Foundation Models						
Qwen3-8B	–	37.3	40.1	40.3	40.0	39.4
Qwen3-30B	–	62.0	75.4	41.8	49.3	57.1
Qwen3-235B	–	64.7	83.3	65.8	65.3	69.8
General Agentic Synthesis Baselines						
AgentSkills	GPT-5.2	35.3	64.3	61.4	51.3	53.0
EnvScaler	GPT-4.1	36.0	58.2	38.6	48.7	45.3
KlearAgentForge	Strong LLMs	40.7	69.3	36.3	46.7	48.2
AgenticQwen	Qwen3-235B	45.3	65.5	47.1	56.0	53.4
Synthesis Variant Baselines						
Monolithic Synthesis	Opus 4.5	47.3	53.5	56.4	40.6	49.5
Fixed-State Synthesis	Opus 4.5	54.7	59.1	61.7	48.7	56.1
R-Zero Style	Opus 4.5	59.3	69.6	67.3	56.7	63.2
AutoPlay Style	Opus 4.5	60.7	71.3	70.8	55.3	64.5
Ours						
EVOSYN-R1	Qwen3-30B	62.0	71.4	43.8	47.3	56.1
EVOSYN-Final-SFT	Qwen3-30B	68.2	76.3	47.7	52.7	61.2
EVOSYN-R1	Qwen3-235B	65.3	80.1	63.5	58.6	66.9
EVOSYN-Final-SFT	Qwen3-235B	70.6	84.2	68.4	63.3	71.6
EVOSYN-R1	Opus 4.5	64.8	78.3	79.8	60.7	70.9
EVOSYN-Final-SFT	Opus 4.5	**71.3**	**83.6**	**83.9**	**72.6**	**77.8**

5.3 RQ2: ABLATION STUDY

Table 2 shows that the full EVOSYN achieves the strongest downstream performance and intrinsic data quality across both Airline and CAR-Bench, consistently leading all reported metrics. Removing learning feedback reduces success from 0.713 to 0.660 on Airline and from 0.726 to 0.613 on CAR-Bench, while trajectory diversity also drops markedly, indicating that feedback reshapes the structural support of synthesized experience rather than merely filtering individual samples. Removing the Reflector consistently degrades both task performance and data quality, with CAR-Bench factor coverage falling from 0.893 to 0.803 and trajectory diversity from 0.938 to 0.800. This supports the role of synthesis-space expansion: recurrent learner failures reveal useful interaction factors that cannot be recovered by allocation over the existing space alone. Removing the Designer exposes a complementary failure mode, reducing CAR-Bench success to 0.540 and factor coverage to 0.634, showing that discovering useful factors is insufficient unless synthesis effort is actively redirected toward them. The accompanying degradation in balance, interaction coverage, and trajectory diversity further suggests that allocation affects not only which regions are emphasized, but also the overall composition of the resulting training experience. Finally, Single-Agent Synthesis performs worst overall, with Airline trajectory diversity collapsing from 0.769 to 0.077 and CAR-Bench success falling to 0.406, demonstrating that role specialization is critical for producing broad, diverse, and effective agentic supervision.

5.4 RQ3: HOW DOES LEARNER FEEDBACK EVOLVE THE SYNTHESIS SPACE?

Figure 3 illustrates how EVOSYN revises its synthesis focus as the learner's capability boundary changes. After R1, failures concentrate on execution-heavy tasks with many state-changing actions, prompting the Reflector to introduce `num_write` and the Designer to shift R2 toward high-write trajectories. After R2, the residual weakness moves to read-heavy tasks, leading to the introduction of `num_read` and a corresponding R3 shift toward read–decide–act interactions. Two controls

Table 2: Ablation study on τ^2-Bench Airline and CAR-Bench Base. All variants use Opus 4.5 as the synthesis LLM, Qwen3-8B as the task agent, Final-SFT as the training protocol, and the same candidate generation budget. Intrinsic data-quality metrics are computed only on the final verified synthetic samples and normalized to $[0, 1]$, where higher is better.

Method	τ^2-Bench Airline					CAR-Bench Base				
	Success Rate	Factor Coverage	Factor Balance	Interaction Coverage	Trajectory Diversity	Success Rate	Factor Coverage	Factor Balance	Interaction Coverage	Trajectory Diversity
EvoSyn	**0.713**	**0.891**	**0.450**	**0.455**	**0.769**	**0.726**	**0.893**	**0.508**	**0.373**	**0.938**
w/o Learning Feedback	0.660	0.778	0.368	0.403	0.538	0.613	0.787	0.468	0.351	0.750
w/o Reflector	0.673	0.862	0.413	0.437	0.615	0.633	0.803	0.483	0.350	0.800
w/o Designer	0.613	0.766	0.387	0.412	0.462	0.540	0.634	0.450	0.339	0.733
Single-Agent Synthesis	0.547	0.365	0.165	0.133	0.077	0.406	0.465	0.425	0.287	0.467

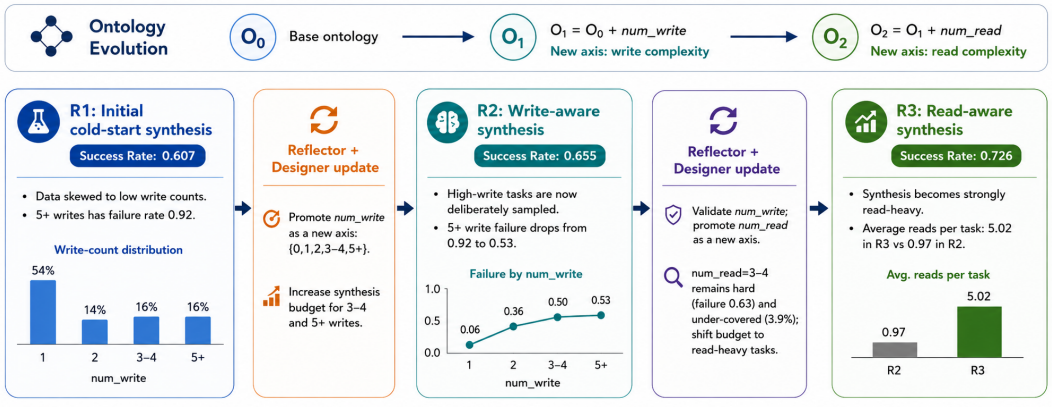


Figure 3: Closed-loop synthesis-space evolution on CAR-Bench. As learner weaknesses shift across rounds, EvoSyn promotes newly discovered interaction factors into explicit generation controls and redirects synthesis toward the evolving capability boundary, progressing from write-intensive execution to increasingly read-intensive decision making.

further support this mechanism. The feedback-conditioned Reflector recovers both subsequently adopted factors $(2/2)$, whereas a no-feedback control proposes six plausible dimensions but recovers neither $(0/6$; Appendix B.6, Table 6). The promoted coordinates are also operational generation controls: requested and realized factor values agree with 99.6–100% exact accuracy and Spearman $\rho = 0.997$–1.000 across R2 synthesis runs (Appendix B.7, Table 7). These results indicate that EvoSyn performs a learner-grounded synthesis-space revision rather than a generic expansion.

Figure 4 further shows that this evolution yields more useful experience than simply accumulating static samples. Starting from the same R1 data, Static Multi-Round changes from 0.607 to 0.627 and then 0.613, whereas EvoSyn improves monotonically from 0.607 to 0.655 and 0.726. Its final data also achieves higher factor coverage (0.893 vs. 0.787) and trajectory-pattern diversity (0.938 vs. 0.750). Thus, later rounds are beneficial not merely because they add data, but because feedback changes both what can be synthesized and where synthesis effort is allocated. Detailed CAR-Bench evidence is provided in Appendix B.13.1, with a τ^2-Bench case in Appendix B.13.2.

5.5 RQ4: EvoSyn Adapts Synthesis Toward Capability Gaps

We further examine whether the closed loop in EvoSyn produces targeted capability evolution, rather than merely changing the data distribution or improving average performance. Figure 5 illustrates the R1→R2 transition on CAR-Bench Base by aligning the task agent's pre-round capability gap, the next round's verified-data proportion shift, and the post-round capability gain over the same interaction slices. The resulting pattern reveals a two-stage self-correcting process. After R1, the largest actionable weakness lies in write-heavy execution; EvoSyn correspondingly shifts R2 synthesis toward multi-write and high-write tasks, and these targeted slices improve after training.

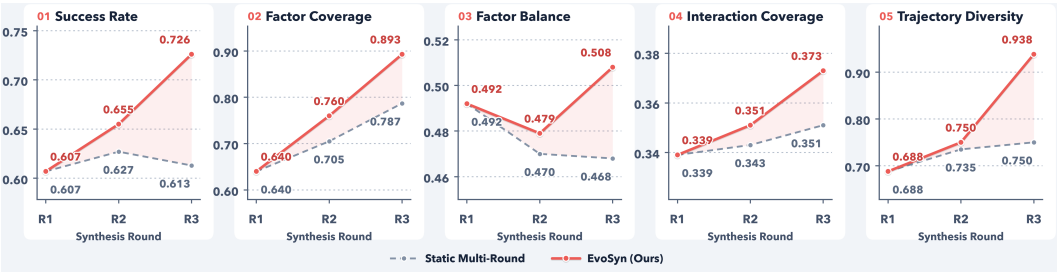


Figure 4: Closed-loop evolution of task performance and synthetic-data quality on CAR-Bench. Unlike static accumulation, EVOSYN progressively improves both downstream success and the structural quality of synthesized experience across successive synthesis rounds.

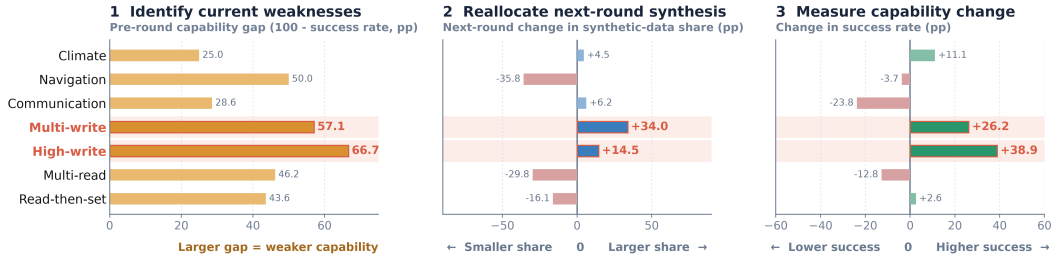


Figure 5: Capability-gap-driven synthesis adaptation on CAR-Bench. The three panels trace the closed-loop transition from pre-round capability gaps to next-round synthesis reallocation and subsequent capability changes, showing how EVOSYN redirects synthesis toward learner weaknesses.

After R2, the residual weakness moves from execution-heavy writing to read-heavy decision making; EVOSYN redirects R3 synthesis toward multi-read, navigation, and read-then-set interactions, which also improve in the next evaluation. The corresponding R2→R3 transition and detailed statistics are provided in Appendix B.5, showing that the system reallocates synthesis toward the current capability boundary and pivots as that boundary evolves.

5.6 RQ5: How Does EvoSyn Differ from Existing Synthetic Corpora?

We further compare EVOSYN with publicly released synthetic corpora constructed for the same environment. As shown in Figure 7, under matched data budgets on Airline, EVOSYN provides the largest unique decisive interaction support (73.0%), compared with AReaL-SEA (69.0%), APIGen-MT (49.8%), and WRIT (25.5%). This complementarity also grows through synthesis-space evolution, with baseline-unseen structural support increasing from 13.6% in R1 to 79.4% in later rounds. Appendix B.4 further shows substantially higher order-sensitive contrastive support and more balanced synthesis across interaction structures. These results show that feedback-guided evolution changes both the support and distribution of synthesized experience, rather than merely generating additional variants within a fixed synthesis space, as learner weaknesses evolve through later rounds.

6 Conclusion

In this paper, we present EVOSYN, a closed-loop multi-agent framework for self-evolving agentic data synthesis. By coupling environment-grounded generation with task-agent learning, EVOSYN uses post-training feedback to both reallocate synthesis effort within known interaction factors and expand the synthesis space with newly discovered factors. Experiments on τ^2-Bench and CAR-Bench show consistent improvements in task-agent performance and synthetic-data quality, while demonstrating that learner feedback reveals useful interaction factors missed by feedback-free synthesis. Further analyses show that this iterative evolution progressively broadens the structural support of synthesized experience beyond that covered by existing synthesis pipelines.

AI USE STATEMENT

Generative AI tools were used in this work in two distinct capacities. (i) As components of the proposed method and experiments: since this work studies LLM-based agentic data synthesis, large language models are an integral part of our framework and experimental setup. Specifically, Claude Opus 4.5, Qwen3-30B-A3B-Thinking-2507, and Qwen3-235B-A22B-Thinking-2507 serve as synthesis backends for the role-specialized agents in EVOSYN, which synthesize the tasks and trajectories used for training. Claude Opus 4.5 is additionally used as the user simulator during both synthesis and evaluation, and Qwen3 models serve as the downstream task agents trained on the synthesized data. The full configuration of these models is described in Section 5.1 and Appendix B.1. (ii) For manuscript preparation: we used generative AI tools for language polishing and manuscript editing. The research ideas, methodology, experimental design, and analysis were developed by the authors, and no generative AI tool was used to fabricate, modify, or interpret empirical results. LLM-assisted text was reviewed and revised by the authors to ensure accuracy, consistency with the experimental results, and appropriate academic presentation. We take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies synthetic data generation and task-agent training in benchmarked interactive environments. It does not involve human subjects or private user data. All experiments are conducted using publicly available benchmarks and simulated environments.

REPRODUCIBILITY STATEMENT

We provide the complete method specification in the main paper and additional implementation details, training hyperparameters, synthesis-space definitions, and evaluation protocols in the appendix. Anonymous source code and supporting materials for reproducing our experiments are available at `https://anonymous.4open.science/r/evosyn_repo-4DB0`.

REFERENCES

Maosong Cao, Taolin Zhang, Mo Li, Chuyu Zhang, Yunxin Liu, Haodong Duan, Songyang Zhang, and Kai Chen. Condor: Enhance llm alignment with knowledge-driven data synthesis and refinement. *arXiv preprint arXiv:2501.12273*, 2(4), 2025.

Jiaao Chen, Jingyuan Qi, Mingye Gao, Wei-Chen Wang, Hanrui Wang, and Di Jin. Eigendata: A self-evolving multi-agent platform for function-calling data synthesis, auditing, and repair. *arXiv preprint arXiv:2603.05553*, 2026.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, et al. Scaling agent learning via experience synthesis. *arXiv preprint arXiv:2511.03773*, 2025.

Yuhao Cheng, Liang Tang, Shuxian Li, Yukang Huo, Tiaonan Duan, Kaer Huang, Yanzhe Jing, and Yiqiang Yan. Evolving in tasks: Empowering the multi-modality large language model as the computer use agent. *arXiv preprint arXiv:2508.04037*, 2025.

Yuqin Dai, Ning Gao, Wei Zhang, Jie Wang, Zichen Luo, Jinpeng Wang, Yujie Wang, Ruiyuan Wu, and Chaozheng Wang. Sead: Self-evolving agent for multi-turn service dialogue. *arXiv preprint arXiv:2602.03548*, 2026.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026.

Jiaxuan Gao, Jiaao Chen, Chuyi He, Wei-Chen Wang, Shusheng Xu, Hanrui Wang, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward rl: Post-training multi-turn interactive tool-using agents. *arXiv preprint arXiv:2601.22607*, 2026.

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. In *International Conference on Learning Representations*, volume 2024, pp. 48362–48395, 2024.

Hengrui Gu, Xiaotian Han, and Kaixiong Zhou. Writ: Write-read intensive trajectory synthesis for multi-turn user-facing agents. *arXiv preprint arXiv:2606.02908*, 2026.

Junliang He, Ziyue Fan, Shaohui Kuang, Li Xiaoqing, Kai Song, Yaqian Zhou, and Xipeng Qiu. Fine: Filtering and improving noisy data elaborately with large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8686–8707, 2025.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

Wai-Chung Kwan, Aryo Pradipta Gema, Joshua Ong Jun Leang, and Pasquale Minervini. Scope: Self-play via co-evolving policies for open-ended tasks. *arXiv preprint arXiv:2605.31433*, 2026.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. Self-alignment with instruction backtranslation. In *The twelfth international conference on learning representations*, 2023.

Zhongzhi Li, Xuansheng Wu, Yijiang Li, Lijie Hu, and Ninghao Liu. Less is enough: synthesizing diverse data in feature space of llms. *arXiv e-prints*, pp. arXiv–2602, 2026.

Xiao Liang, Zhong-Zhi Li, Yeyun Gong, Yang Wang, Hengyuan Zhang, Ying Nian Wu, Weizhu Chen, et al. Sws: Self-aware weakness-driven problem synthesis in reinforcement learning for llm reasoning. *Advances in Neural Information Processing Systems*, 38:56801–56839, 2026.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jian-Guang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, et al. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. In *International Conference on Learning Representations*, volume 2025, pp. 49573–49609, 2025.

Yuanjie Lyu, Chengyu Wang, Haonan Zheng, Yuanhao Yue, Junbing Yan, Ming Wang, and Jun Huang. Agenticqwen: Training small agentic language models with dual data flywheels for industrial-scale tool use. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pp. 535–551, 2026.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Hassan. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6300–6323, 2025.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Advances in Neural Information Processing Systems*, 38, 2026.

Ram Ramrakhya, Andrew Szot, Omar Attia, Yuhao Yang, Anh Nguyen, Bogdan Mazoure, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling synthetic task generation for agents via exploration. *arXiv preprint arXiv:2509.25047*, 2025.

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.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, 2026.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, 2025.

Zexu Sun, Bokai Ji, Hengyi Cai, Shuaiqiang Wang, Lei Wang, Guangxia Li, and Xu Chen. Agentskiller: Scaling generalist agent intelligence through semantically integrated cross-domain data synthesis. *arXiv preprint arXiv:2602.09372*, 2026.

Shuo Tang, Xianghe Pang, Zexi Liu, Bohan Tang, Rui Ye, Tian Jin, Xiaowen Dong, Yanfeng Wang, and Siheng Chen. Synthesizing post-training data for llms through multi-agent simulation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23306–23335, 2025.

Zeao Tu, Xiangdi Meng, Yu He, Zihan Yao, Tianyu Qi, Jun Liu, and Ming Li. Resofilter: Fine-grained synthetic data filtering for large language models through data-parameter resonance analysis. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5414–5428, 2025.

Qi Wang, Hongzhi Zhang, Jia Fu, Kai Fu, Yahui Liu, Tinghai Zhang, Chenxi Sun, Gangwei Jiang, Jingyi Tang, Xingguang Ji, et al. Klear-agentforge: Forging agentic intelligence through post-training scaling. *arXiv preprint arXiv:2511.05951*, 2025a.

Rui Wang, Ce Zhang, Jun-Yu Ma, Jianshu Zhang, Hongru Wang, Yi Chen, Boyang Xue, Tianqing Fang, Zhisong Zhang, Hongming Zhang, et al. Webaggregator: Enhancing compositional reasoning capabilities of deep research agent foundation models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24486–24517, 2026a.

Shaobo Wang, Zhengbo Jiao, Zifan Zhang, Yilang Peng, Xu Ze, Boyu Yang, Wei Wang, Hu Wei, and Linfeng Zhang. Socratic-zero: Bootstrapping reasoning via data-free agent co-evolution. *arXiv preprint arXiv:2509.24726*, 2025b.

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, 2023.

Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Pei Chen, Ziwei Dong, Jing Huang, Jiri Gesi, Xianfeng Tang, Chen Luo, Qun Liu, et al. Trajectory2task: Training robust tool-calling agents with synthesized yet verifiable data for complex user intents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44021–44044, 2026b.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, volume 2024, pp. 30745–30766, 2024.

Chen Yang, Ran Le, Yun Xing, Zhenwei An, Zongchao Chen, Wayne Xin Zhao, Yang Song, and Tao Zhang. Toolmind technical report: A large-scale, reasoning-enhanced tool-use dataset. *arXiv preprint arXiv:2511.15718*, 2025.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. Coevolve: Training llm agents via agent-data mutual evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23015–23036, 2026.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. In *International Conference on Learning Representations*, volume 2024, pp. 40320–40341, 2024.

Zhenrui Yue, Kartikeya Upasani, Xianjun Yang, Suyu Ge, Shaoliang Nie, Yuning Mao, Zhe Liu, and Dong Wang. Dr. zero: Self-evolving search agents without training data. *arXiv preprint arXiv:2601.07055*, 2026.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

Shuo Zhang, Chaofa Yuan, Ryan Guo, Xiaomin Yu, Rui Xu, Zhangquan Chen, Zinuo Li, Zhi Yang, Shuhao Guan, Zhenheng Tang, et al. Evofsm: Controllable self-evolution for deep research with finite state machines. *arXiv preprint arXiv:2601.09465*, 2026.

APPENDIX

A METHOD

A.1 PROBLEM FORMULATION

Interactive Agentic Synthesis. We consider synthetic data generation for improving language-model agents in interactive tool-use environments. Let $\mathcal{E}$ denote an environment that specifies domain policies, tool interfaces, state constraints, and an automatic task evaluator, and let π_{θ_r} denote the task agent at synthesis round r. A synthesized training example consists of a task x and an interaction trajectory τ, where τ contains the sequence of language responses, tool calls, and environment observations produced while solving x. We denote by

$$V_{\mathcal{E}}(x, \tau) \in \{0, 1\}$$

the environment verifier, which determines whether the task–trajectory pair is executable, policy-consistent, state-consistent, and successful under the environment evaluator.

At round r, a synthesis process generates a candidate set $\widetilde{\mathcal{D}}_r = \{(x_i, \tau_i)\}_{i=1}^{N_r}$, from which verified supervision is retained as

$$\mathcal{D}_r = \left\{ (x_i, \tau_i) \in \widetilde{\mathcal{D}}_r \mid V_{\mathcal{E}}(x_i, \tau_i) = 1 \right\}. \tag{12}$$

The task agent is trained on accumulated verified experience,

$$\pi_{\theta_{r+1}} = \mathrm{Train}\left(\pi_{\theta_r}, \mathcal{D}_{\le r}\right), \qquad \mathcal{D}_{\le r} = \bigcup_{t=1}^{r} \mathcal{D}_t. \tag{13}$$

We measure the resulting agent by its held-out task success $J(\pi_{\theta_{r+1}})$ in the target environment. Unlike one-pass synthesis, however, the value of newly synthesized data depends on the current learner: an example can be fully valid while providing little useful experience for an agent that has already mastered the corresponding interaction pattern.

Revisable Synthesis State. To make this learner dependence explicit, we characterize synthesis at round r by a state

$$S_r = (\mathcal{O}_r, \alpha_r), \tag{14}$$

which separates two complementary decisions. The *synthesis space* $\mathcal{O}_r$ specifies *what interaction properties can be deliberately synthesized*, while α_r specifies *where synthesis effort is allocated within that space*. We use $\mathcal{Z}(\mathcal{O}_r)$ to denote the set of synthesis directions expressible under $\mathcal{O}_r$. A direction $z \in \mathcal{Z}(\mathcal{O}_r)$ corresponds to a controllable region of the synthesis space, whose concrete structure is introduced in Section 4.

Given a synthesis budget B_r, we view

$$\alpha_r : \mathcal{Z}(\mathcal{O}_r) \to \mathbb{R}_{\ge 0}, \qquad \sum_{z \in \mathcal{Z}(\mathcal{O}_r)} \alpha_r(z) = B_r, \tag{15}$$

as an allocation of generation effort across synthesis directions. The synthesis process can therefore be written abstractly as

$$\widetilde{\mathcal{D}}_r \sim \mathcal{G}\left(\mathcal{E}, \mathcal{O}_r, \alpha_r\right),$$

where $\mathcal{G}$ realizes high-level synthesis decisions as concrete agentic tasks and trajectories. This decomposition distinguishes changing the distribution of experience within a known space from changing the space of experience that can be deliberately generated in the first place.

Feedback-to-Decision Revision. After training, the updated task agent is evaluated to obtain learner feedback

$$F_r = \mathrm{Feedback}\left(\pi_{\theta_{r+1}}, \mathcal{D}_{\le r}, \mathcal{O}_r, \mathcal{E}\right), \tag{16}$$

which summarizes evidence about the learner's current capabilities and the synthesized experience it has received. The central problem is not merely to observe F_r, but to translate it into an actionable revision of future synthesis. We formalize this step through a synthesis revision operator

$$S_{r+1} = \Psi(S_r, F_r), \qquad (\mathcal{O}_{r+1}, \alpha_{r+1}) = \Psi(\mathcal{O}_r, \alpha_r, F_r). \tag{17}$$

This revision map captures the **feedback-to-decision gap**: learner feedback is expressed through observed behavior, whereas future synthesis requires decisions over controllable experience.

Importantly, feedback can imply two fundamentally different forms of revision. First, when a remaining weakness is already expressible by the current synthesis coordinates, the mismatch is *distributional*. The synthesis space remains fixed while generation effort is redistributed:

$$\underbrace{\mathcal{O}_{r+1} = \mathcal{O}_r}_{\text{same synthesis space}}, \qquad \underbrace{\alpha_{r+1} \neq \alpha_r}_{\text{revised allocation}}. \tag{18}$$

We refer to this case as **within-space adaptation**. It captures settings in which the appropriate interaction factor is already controllable, but the learner has received insufficient, imbalanced, or poorly targeted experience along that factor.

Second, learner feedback may expose a systematic weakness that cannot be adequately represented by any existing synthesis coordinate. In this case, reallocating α_r within the current space cannot directly target the underlying capability gap. The synthesis representation itself must be revised:

$$\mathcal{O}_{r+1} = \mathcal{O}_r \cup \Delta\mathcal{O}_r, \qquad \Delta\mathcal{O}_r \neq \varnothing, \tag{19}$$

where $\Delta\mathcal{O}_r$ contains newly introduced controllable interaction factors. We refer to this case as **synthesis-space expansion**. The allocation α_{r+1} is then defined over the expanded set of synthesis directions $\mathcal{Z}(\mathcal{O}_{r+1})$. Conventional adaptation over a fixed synthesis space is a special case in which $\mathcal{O}_{r+1} = \mathcal{O}_r$ for all rounds.

Closed-Loop Objective. The overall synthesis–learning process can be summarized as

$$S_r \xrightarrow{\text{synthesize}} \mathcal{D}_r \xrightarrow{\text{train}} \pi_{\theta_{r+1}} \xrightarrow{\text{evaluate}} F_r \xrightarrow{\text{revise}} S_{r+1}. \tag{20}$$

Given a total synthesis budget B, our goal is to construct a sequence of feedback-conditioned synthesis states that maximizes the final task-agent performance,

$$\max_{S_{1:R}} \quad J\big(\pi_{\theta_{R+1}}\big) \qquad \text{s.t.} \qquad \sum_{r=1}^{R} B_r \leq B, \quad S_{r+1} = \Psi(S_r, F_r). \tag{21}$$

The resulting problem therefore requires deciding, at each round, whether learner feedback should change *where* synthesis effort is placed within the current space or *what* the synthesis process is capable of explicitly controlling. Section 4 introduces EvoSyn, which instantiates this revision-and-realization process through a role-specialized multi-agent architecture.

A.2 Comparison with Existing Data Synthesis Frameworks

Table 3 reveals a clear progression in existing data-synthesis frameworks, but also a persistent separation between environment grounding, learner adaptation, and synthesis-space evolution. Reasoning-oriented methods such as R-Zero, Socratic-Zero, and SwS make generation learner-aware by targeting solver failures or capability boundaries, yet operate outside executable, stateful agent environments. Agentic synthesis methods such as AReaL-SEA, Explorer, AutoPlay, WebAggregator, AgentSkiller, and EnvScaler address the complementary problem: they generate executable and verifiable interaction experience, but synthesis is largely driven by environment structure, coverage, or data quality rather than by the trained learner's post-training response. More tightly coupled systems, including Agent-World, DreamGym, SEAD, Agent0, and AgenticQwen, further incorporate task-agent feedback and closed-loop resynthesis. However, their adaptation remains confined to already available task representations, predefined dimensions, or free-form generation, without explicitly revising the synthesis space when the learner exposes a previously unrepresented capability gap.

EvoSyn closes this gap by making the synthesis representation itself part of the learning loop. Under the conservative criteria in Table 3, it is the only framework that jointly supports all eight capabilities, and in particular the only one that combines *learning-effect feedback* with explicit *new-factor discovery*. Post-training behavior is mapped into a structured and revisable synthesis space that serves simultaneously as a diagnostic representation and a generation-control interface. When a weakness is already expressible in the current space, EvoSyn performs known-factor adaptation

Table 3: Comparison of representative data-synthesis methods for post-training LLMs and agents. ✓ indicates that the capability is explicitly supported and demonstrated by the method, while – indicates that it is absent or not demonstrated. Partial or implicit support is conservatively marked as –.

Method	Interactive Setting	Executable & Verifiable	Task-Agent Feedback	Learning-Effect Feedback	Structured Task Space	Known-Factor Adaptation	New-Factor Discovery	Closed-Loop Evolution
Reasoning and instruction synthesis								
MATRIX / MATRIX-Gen	–	–	–	–	✓	–	–	–
Socratic-Zero	–	–	✓	–	–	✓	–	✓
R-Zero	–	–	✓	–	–	✓	–	✓
SwS	–	–	✓	–	✓	✓	–	–
Interactive and agentic data synthesis								
AReaL-SEA / EigenData	✓	✓	–	–	–	–	–	–
SCOPE	✓	–	✓	–	–	✓	–	✓
Agent-World	✓	✓	✓	–	✓	✓	–	✓
WebAggregator	✓	✓	–	–	✓	–	–	–
DreamGym	✓	✓	✓	–	–	✓	–	✓
Explorer	✓	✓	–	–	–	–	–	–
AutoPlay	✓	✓	–	–	✓	–	–	–
SEAD	✓	✓	✓	–	✓	✓	–	✓
Agent0	✓	✓	✓	–	–	✓	–	✓
AgentSkiller	✓	✓	–	–	✓	–	–	–
EnvScaler	✓	✓	–	–	✓	–	–	–
Klear-AgentForge	✓	✓	✓	–	–	–	–	–
AgenticQwen	✓	✓	✓	–	✓	✓	–	✓
Ours	✓	✓	✓	✓	✓	✓	✓	✓

Criterion definitions. *Interactive Setting* denotes synthesis for multi-turn interaction with users, tools, or stateful environments. *Executable & Verifiable* requires synthesized tasks or trajectories to be executable in an environment and automatically verifiable. *Task-Agent Feedback* means that synthesis is informed by the current task agent's behavior, performance, or failures. *Learning-Effect Feedback* further evaluates how previously synthesized data changes the task agent after training and uses this learning effect to guide subsequent synthesis. *Structured Task Space* denotes an explicit representation based on task factors, schemas, states, or an ontology. *Known-Factor Adaptation* adjusts difficulty, coverage, proportions, or generation budgets over already represented factors. *New-Factor Discovery* requires discovering a previously unrepresented reusable synthesis factor and incorporating it into future synthesis; generating new instances, trajectories, or environment states alone does not satisfy this criterion. *Closed-Loop Evolution* requires repeated synthesis, learner training, evaluation, and feedback-driven resynthesis.

by reallocating synthesis effort toward weak or under-covered regions. When the weakness cannot be represented by existing coordinates, it instead expands the space by promoting a new reusable and controllable interaction factor. This distinction is critical: rather than merely generating harder instances, repairing failed samples, or resampling weak categories, EvoSyn can change both *where* synthesis is concentrated and *what* the synthesis process is capable of explicitly controlling. The resulting loop therefore connects executable data generation, task-agent training, post-training diagnosis, synthesis-space revision, and targeted resynthesis within a unified closed-loop framework.

A.3 Closed-Loop Role-Specialized Synthesis

Agentic data synthesis spans multiple levels of abstraction. The system must reason about what experience the current learner needs, allocate synthesis effort accordingly, translate abstract synthesis targets into feasible tasks under environment constraints, realize successful tool-use trajectories, and verify that the resulting supervision is valid. We therefore factorize EvoSyn by *decision abstraction*, assigning specialized agents to distinct roles in the synthesis–learning loop rather than forcing a single generator to jointly solve these heterogeneous decisions.

At round r, task-agent feedback F_r first enters a *backward synthesis-revision path*. A Reflector Agent reasons over learner performance, data coverage, and failure structure to revise the synthesis state,

$$F_r \xrightarrow{\text{Reflector}} S_{r+1} = (\mathcal{O}_{r+1}, \alpha_{r+1}), \tag{22}$$

where $\mathcal{O}_{r+1}$ determines which interaction factors can be explicitly controlled and α_{r+1} determines how synthesis effort is allocated over them. Importantly, the Reflector revises the synthesis process itself rather than directly repairing individual failed examples.

The revised synthesis state is then realized through a *forward grounded-synthesis path*. A Designer Agent operates at the strategic level, translating the synthesis state into generation directions and budgets. A Planner Agent grounds each abstract direction into an environment-consistent task blueprint, bridging synthesis-level intent and executable task construction. The Generator Agent and rollout teacher instantiate these blueprints into concrete tasks and successful interaction trajectories, while a Verifier Agent filters invalid supervision according to environment and task constraints. The resulting verified experience is used to train and evaluate the task agent, producing new feedback that initiates the next round of synthesis revision.

This decomposition gives the major components of EVOSYN distinct but complementary roles. The Reflector performs *meta-level synthesis revision*, the Designer performs *round-level synthesis planning*, and the Planner performs *instance-level grounding*, while the Generator, rollout teacher, and Verifier realize and validate the resulting experience in the environment. Together, these components form a closed loop in which learner behavior reshapes the synthesis process, and the revised synthesis process determines the experience from which the learner evolves.

B EXPERIMENTS

This appendix provides additional experimental details and analyses supporting the main results. We first describe the complete experimental setup, including datasets, synthesis backends, verification, task-agent training, controlled baselines, intrinsic data-quality metrics, initial synthesis spaces, and representative agent prompts. We then examine the robustness and properties of EVOSYN through task-agent scaling, Designer ablations, and comparative structural analyses against existing synthetic corpora. A series of mechanism-focused experiments further investigates capability-gap-driven reallocation, feedback-dependent factor discovery, controllability of evolved synthesis factors, structural-support expansion, residual learner weaknesses, and observed learning gains across rounds. Finally, detailed CAR-Bench and τ^2-Bench Airline case studies trace how learner feedback is converted into synthesis-space revisions and subsequent changes in the generated experience.

B.1 EXPERIMENTAL DETAILS

Datasets and Evaluation. We evaluate EVOSYN on τ^2-Bench and CAR-Bench, both of which provide executable interactive environments and automatic task-level evaluators. For τ^2-Bench, we consider the Airline, Retail, and Telecom domains, while CAR-Bench is evaluated under the Base setting. We follow the official dataset splits and evaluate on held-out benchmark tasks using the benchmark-native binary task-success reward. Unless otherwise specified, reported results are averaged over three random seeds.

Synthesis Backends. We instantiate EVOSYN with three synthesis backends: Claude Opus 4.5, Qwen3-30B-A3B-Thinking-2507, and Qwen3-235B-A22B-Thinking-2507. The open-source Qwen3 models are served with vLLM. Within each synthesis run, the Designer, Planner, Generator, Repair module, and Reflector share the selected backend. Claude Opus 4.5 is used as the user simulator during both synthesis and evaluation, with thinking disabled to match the benchmark-standard simulator configuration. Unless otherwise specified, the Designer, Planner, and Generator use a sampling temperature of 1.0. For brevity, we refer to Qwen3-30B-A3B-Thinking-2507 and Qwen3-235B-A22B-Thinking-2507 as Qwen3-30B and Qwen3-235B throughout the paper.

Synthesis and Verification. Each round generates a fixed budget of synthesis blueprints within the corresponding experimental arm and retains only environment-grounded trajectories that pass deterministic replay and achieve the benchmark reward of 1. Main synthesis runs typically use approximately 1,200–1,500 blueprint proposals per round, with domain-specific adjustments where necessary. When a proposed pseudo-ground-truth trajectory fails replay, the Repair module is invoked for at most two attempts, with every repaired trajectory replayed from the original environment state. The synthesis process runs for at most three rounds and terminates early after two consecutive rounds without a synthesis-space update. To keep synthesis-space evolution gradual and interpretable, the Reflector introduces at most one new synthesis factor per round. To construct learning feedback, we sample diagnostic examples from each synthesis round and accumulate them

across rounds, allowing the Reflector to analyze the learner against both newly generated experience and the evolving synthesis history.

Task-Agent Training. Qwen3-8B serves as the default downstream task agent, while Qwen3-14B and Qwen3-30B are additionally evaluated for task-agent scaling. We perform full-parameter supervised fine-tuning with LLaMA-Factory and DeepSpeed ZeRO-3 using the same training recipe across model scales and synthesis rounds. We use a maximum sequence length of 16,384, six epochs, a learning rate of 1×10^{-5} with cosine decay, a warm-up ratio of 0.1, BF16 precision, and FlashAttention-2. The per-device batch size is 1 with two gradient-accumulation steps over eight GPUs, yielding an effective global batch size of 16. Training examples are exported in ShareGPT format from verified successful trajectories, and multi-round EvoSyn trains on the accumulated verified experience synthesized up to the corresponding round.

Controlled Comparisons. For synthesis variants and ablations, we reuse the same environment, tool schema, domain policy, verifier, task-agent initialization, SFT recipe, and evaluation harness, changing only the synthesis mechanism under study. All generated trajectories are subject to the same environment replay and reward-based verification procedure, enabling controlled comparisons of feedback, synthesis-space evolution, and role specialization.

Evaluation Protocol. Learner feedback used for synthesis revision is computed on a held-out diagnostic set that is disjoint from the official test set. The official held-out benchmark tasks are reserved for final evaluation and reporting, using a rollout temperature of 1.0 and the corresponding native evaluator. Test labels are never used to drive synthesis or training. Analyses involving newly promoted synthesis factors use the learner state before the corresponding factor is introduced, preventing post-hoc leakage from subsequent factor-guided synthesis or training.

Baselines. We compare against four recent agentic data synthesis and post-training methods. **AgentSkiller** (Sun et al., 2026) constructs executable environments and synthesizes high-fidelity multi-turn interaction trajectories across semantically connected domains, combining structured domain modeling, cross-domain composition, and execution-based validation to train generalist agents. **EnvScaler** (Song et al., 2026) programmatically synthesizes tool-interactive environments and further generates scenarios, tasks, verification functions, and interaction trajectories for supervised fine-tuning and reinforcement learning. **Klear-AgentForge** (Wang et al., 2025a) builds general-purpose agentic models through large-scale supervised fine-tuning followed by multi-turn reinforcement learning, using trajectories distilled or synthesized from strong LLMs together with filtering and verification. **AgenticQwen** (Lyu et al., 2026) adopts iterative data flywheels to progressively increase training difficulty, including an agentic flywheel that expands workflows into multi-branch behavior structures and converts the resulting branches into new executable tasks for subsequent reinforcement learning. For these baselines, we evaluate the reported synthetic data or training pipeline using Qwen3-8B as the downstream task agent for a consistent comparison.

We additionally construct four synthesis-variant baselines under the same environment, verifier, task-agent initialization, training recipe, and evaluation harness. **Monolithic Synthesis** removes the role-specialized decomposition and directly generates tasks and trajectories with a single synthesis agent. **Fixed-State Synthesis** freezes the initial synthesis space and allocation, disabling feedback-driven revision across rounds. **R-Zero Style** (Huang et al., 2025) uses learner failures as unstructured synthesis guidance without an explicit revisable synthesis space. **AutoPlay Style** (Ramrakhya et al., 2025) uses environment-grounded trajectory exploration without learner-feedback-driven synthesis-space evolution.

Intrinsic Data-Quality Metrics. We evaluate the verified synthetic data using four deterministic intrinsic metrics, all normalized to $[0, 1]$. Metrics are computed only on samples whose ground-truth trajectories successfully replay to full environment reward. Importantly, all structural factors are re-derived from the realized ground-truth tool actions rather than from the synthesis plan, avoiding inflation from intended but unrealized generation targets. We consider eight observed factors, including task family, tool-use pattern, expected outcome, state-change type, computation requirement, numbers of write and read actions, and policy constraints.

Factor Coverage. Let $\mathcal{V}_k$ denote the fixed reference value set of factor k, and $\mathcal{O}_k$ the distinct values observed in the verified data. We measure the fraction of each factor's reference space that is realized:

$$\text{FactorCov} = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \frac{|\mathcal{O}_k \cap \mathcal{V}_k|}{|\mathcal{V}_k|}. \tag{23}$$

Higher values therefore indicate broader marginal coverage of controllable interaction factors. For CAR-Bench, where marginal factor values saturate across methods, we instead use combinatorial coverage over the joint factor configuration space, computed as the fraction of distinct observed factor cells relative to the union of cells across the compared conditions.

Factor Balance. Coverage alone does not reveal whether synthesis is concentrated on only a few values. For each factor k, let p_k denote the empirical distribution over values in $\mathcal{V}_k$. We compute normalized Shannon entropy,

$$\text{FactorBal} = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \frac{H(p_k)}{\log |\mathcal{V}_k|}, \tag{24}$$

where $H(\cdot)$ is Shannon entropy. A larger value indicates that synthesis mass is distributed more evenly across the factor values.

Interaction Coverage. To capture compositional structure beyond individual factors, we measure coverage over predefined pairs of interaction factors. For each template $(a, b) \in \mathcal{P}$, let $\mathcal{O}_{a,b}$ denote the distinct observed value pairs. We compute

$$\text{InteractCov} = \frac{1}{|\mathcal{P}|} \sum_{(a,b) \in \mathcal{P}} \frac{|\mathcal{O}_{a,b}|}{|\mathcal{V}_a|\,|\mathcal{V}_b|}. \tag{25}$$

The templates capture complementary interaction structure such as task family $\times$ tool pattern, task family $\times$ outcome, tool pattern $\times$ state change, and computation $\times$ write complexity. Higher values indicate richer joint coverage of interaction configurations.

Trajectory Diversity. We further characterize the structural diversity of executable trajectories independently of surface language. Each ground-truth tool action is mapped to a functional symbol corresponding to *read*, *write*, *compute*, or *transfer*, and consecutive identical symbols are collapsed to form an abstract trajectory pattern. Let p_{traj} denote the empirical distribution of these patterns and $\mathcal{P}_{\text{union}}$ the union of patterns observed across all compared conditions. We define

$$\text{TrajDiv} = \frac{H(p_{\text{traj}})}{\log |\mathcal{P}_{\text{union}}|}. \tag{26}$$

All compared methods and rounds are evaluated against the same $\mathcal{P}_{\text{union}}$, ensuring that the normalized entropy is directly comparable. Higher values indicate a broader and more evenly distributed set of execution structures.

Initial Task Ontologies We initialize each domain with a compact, domain-specific task ontology. Airline, Retail, and Telecom share the same six ontology axes but use different domain-specific values. CAR-Bench uses seven native axes to better capture vehicle-state execution, infeasibility, and ambiguity.

Airline Initial Ontology $\mathcal{O}_0$

```
{
"task_family":  [
"flight_booking",
"flight_change",
"flight_cancellation",
"refund_or_compensation",
"seat_or_baggage_service",
"loyalty_or_membership_service"
],
"expected_outcome":  [
```

```
"fulfill",
"refuse"
],
"policy_or_constraint_focus": [
"fare_rule_constraint",
"change_or_cancellation_fee",
"refund_eligibility",
"compensation_eligibility",
"passenger_or_identity_constraint",
"time_window_constraint",
"payment_or_credit_constraint",
"loyalty_status_constraint"
],
"tool_use_pattern": [
"search_compare_then_book",
"lookup_verify_then_modify",
"multi_entity_retrieval",
"policy_check_before_action",
"price_or_fee_recalculation",
"state_change_with_confirmation",
"partial_update_or_split_action"
],
"interaction_challenge": [
"missing_information",
"ambiguous_preference",
"user_false_claim",
"preference_change",
"emotional_pressure",
"multi_intent_interleaving",
"constraint_hidden_until_late"
],
"difficulty_profile": ["easy", "medium", "hard"]
}
```


Retail Initial Ontology $\mathcal{O}_0$


```
{
"task_family": [
"order_cancellation",
"order_item_modification",
"order_logistics_modification",
"return_delivered_order",
"exchange_delivered_order",
"profile_or_inquiry"
],
"expected_outcome": ["fulfill", "refuse"],
"policy_or_constraint_focus": [
"order_status_precondition",
"identity_authentication",
"once_per_order_action_lock",
"product_variant_match",
"payment_or_giftcard_balance",
"confirmation_requirement",
"cancellation_reason_validity",
"scope_or_escalation_boundary"
],
"tool_use_pattern": [
"authenticate_then_act",
"lookup_verify_then_modify",
"multi_entity_retrieval",
"policy_check_before_action",
"price_or_refund_calculation",
```

```
"state_change_with_confirmation",
"batched_multi_item_action"
],
"interaction_challenge": [
"missing_information",
"ambiguous_request",
"user_false_claim",
"preference_change",
"emotional_pressure",
"multi_intent_interleaving",
"constraint_hidden_until_late"
],
"difficulty_profile": ["easy", "medium", "hard"]
}
```


Telecom Initial Ontology $\mathcal{O}_0$


```
{
"task_family": [
"plan_change",
"billing_issue",
"device_upgrade",
"line_management",
"service_troubleshooting",
"account_profile_update"
],
"expected_outcome": ["fulfill", "refuse"],
"policy_or_constraint_focus": [
"eligibility_check",
"contract_or_fee_constraint",
"billing_or_payment_constraint",
"identity_or_authorization_constraint",
"service_compatibility_constraint",
"time_window_constraint"
],
"tool_use_pattern": [
"lookup_then_update",
"multi_hop_retrieval",
"compare_options_then_apply",
"verify_before_action",
"diagnose_then_resolve",
"state_change_with_confirmation"
],
"interaction_challenge": [
"missing_information",
"ambiguous_request",
"user_false_claim",
"preference_change",
"emotional_pressure",
"multi_intent_interleaving"
],
"difficulty_profile": ["easy", "medium", "hard"]
}
```


CAR-Bench Initial Ontology $\mathcal{O}_0$


```
{
"task_family": [
"climate_control",
"window_and_body",
"navigation",
```

```
"charging",
"communication",
"info_query"
],
"task_nature": [
"base_execute",
"hallucination_missing_tool",
"hallucination_missing_param",
"hallucination_missing_result",
"disambiguation_internal",
"disambiguation_user"
],
"expected_behavior": [
"fulfill_reach_goal",
"decline_infeasible",
"resolve_from_preferences",
"clarify_with_user"
],
"tool_use_pattern": [
"single_set",
"read_then_set",
"multi_step_sequence",
"cross_domain_compose",
"confirmation_required"
],
"state_precondition": [
"no_precondition",
"ordering_dependency",
"policy_guarded",
"occupancy_or_context_gated"
],
"interaction_challenge": [
"cooperative_clear",
"persona_indirect",
"underspecified_value",
"distractor_context"
],
"difficulty_profile": ["easy", "medium", "hard"]
}
```


For Airline, Retail, and Telecom, each task selects exactly one `task_family`, one `expected_outcome`, one or two `policy_or_constraint_focus` values, one to three `tool_use_pattern` values, zero to two `interaction_challenge` values, and one `difficulty_profile`. The difficulty levels correspond to increasing tool-use and compositional requirements: easy tasks typically require 2–3 tool calls, medium tasks 4–5 calls, and hard tasks 6 or more calls. For the CAR-Bench Base experiments, `task_nature` is restricted to `base_execute`, while `expected_behavior` is restricted to `fulfill_reach_goal` and `resolve_from_preferences`.

Representative Agent Prompts. To keep the experimental-details section concise, we defer the representative prompts for the Designer, Planner, and Reflector to Appendix D.

B.2 Additional Overall Performance

We further examine whether the benefits of EvoSyn persist as the downstream learner scales and how the resulting task agents compare with directly evaluated foundation models. As shown in Table 4, scaling the base Qwen3 backbone from 8B to 14B and 30B raises the average success rate from 39.4% to 48.6% and 57.1%, respectively, while leaving substantial room for post-training. Training on evolved EvoSyn experience consistently closes this gap, with Final-SFT reaching 77.8%, 80.9%, and 86.0% across the three task-agent scales. Crucially, multi-round synthesis remains beneficial even as the learner becomes substantially stronger: relative to R1, feedback-guided evolution

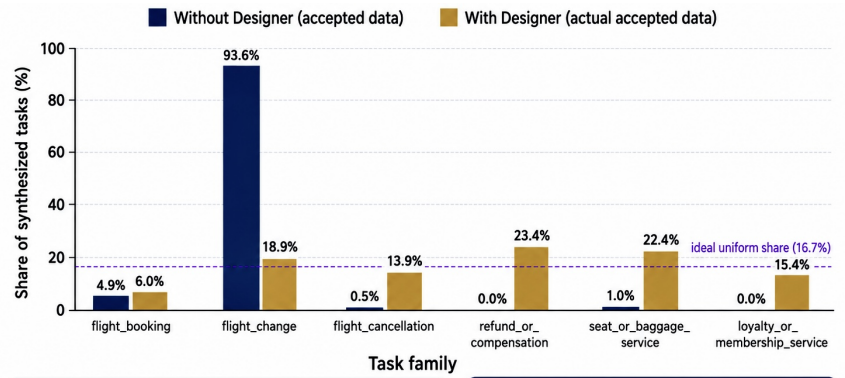
Table 4: Direct foundation-model performance and task-agent scaling results of EVOSYN. For EVOSYN, Claude Opus 4.5 is used as the synthesis LLM, while Qwen3-8B, Qwen3-14B, and Qwen3-30B serve as downstream task agents. EVOSYN-R1 uses first-round synthesized data, whereas EVOSYN-Final-SFT trains on accumulated multi-round data.

Method	Synthesis LLM	Task Agent	τ^2-Bench			CAR-Bench	Avg.
			Airline	Retail	Telecom		
Frontier Closed-source Models							
Claude Haiku 4.5	–	–	60.0	75.1	42.4	53.3	57.7
Claude Sonnet 4.5	–	–	83.3	88.3	62.0	71.3	76.2
Claude Opus 4.5	–	–	79.3	91.8	88.0	82.0	85.3
GPT-5.2	–	–	83.0	81.6	89.7	74.0	82.1
Gemini-3 Flash	–	–	82.5	76.8	91.2	69.3	80.1
Gemini-3 Pro	–	–	80.5	75.9	91.0	67.3	78.7
Open-Source Foundation Models							
Qwen3-8B	–	–	37.3	40.1	40.3	40.0	39.4
Qwen3-14B	–	–	40.7	61.4	43.0	49.3	48.6
Qwen3-30B	–	–	62.0	75.4	41.8	49.3	57.1
Qwen3-235B	–	–	64.7	83.3	65.8	65.3	69.8
EVOSYN *Task-Agent Scaling*							
EVOSYN-R1	Opus 4.5	Qwen3-8B	64.8	78.3	79.8	60.7	70.9
EVOSYN-Final-SFT	Opus 4.5	Qwen3-8B	71.3	83.6	83.9	72.6	77.8
EVOSYN-R1	Opus 4.5	Qwen3-14B	64.5	83.3	82.5	62.7	73.3
EVOSYN-Final-SFT	Opus 4.5	Qwen3-14B	73.3	86.5	90.4	73.3	80.9
EVOSYN-R1	Opus 4.5	Qwen3-30B	75.3	87.7	89.5	68.7	80.3
EVOSYN-Final-SFT	Opus 4.5	Qwen3-30B	**84.0**	**89.8**	**93.3**	**76.7**	**86.0**

provides additional gains of 6.9, 7.6, and 5.7 points for Qwen3-8B, Qwen3-14B, and Qwen3-30B, respectively. These results show that task-agent scaling and synthesis evolution are complementary rather than interchangeable: larger backbones improve the learner itself, while evolving the synthesized experience continues to provide additional supervision beyond model scale alone. Most notably, the Qwen3-30B task agent trained with EVOSYN reaches an average success rate of 86.0%, surpassing the strongest directly evaluated foundation-model average in the table, Claude Opus 4.5 at 85.3%. This demonstrates that the utility of EVOSYN is not confined to compensating for a weak learner, but persists across substantially different model capacities and can elevate open-source task agents to frontier-level performance.

B.3 ADDITIONAL ABLATION: WHY IS A DESIGNER NECESSARY?

Many agentic synthesis pipelines primarily rely on a Planner–Generator decomposition, where the Planner determines how a task should be constructed and the Generator realizes it in the environment. We therefore further investigate whether this conventional decomposition is sufficient for self-evolving synthesis, or whether an explicit Designer is necessary to separate *where to synthesize* from *how to instantiate* each task. Without the Designer, the Planner must simultaneously perform global task-space allocation and local task construction. As shown in Figure 6, this coupling causes a severe distributional collapse on Airline: `flight_change` accounts for 93.6% of accepted data, while two of the six task families are not covered at all. Introducing the Designer explicitly separates high-level synthesis allocation from environment-grounded task realization. The resulting data covers all six task families, while the largest family share drops to 23.4%, yielding substantially more balanced exploration of the task space. This result clarifies why the Designer is important beyond ordinary Planner–Generator synthesis: it provides the global allocation mechanism required to intentionally explore selected regions of the synthesis space, which is especially important in a self-evolving pipeline where learner feedback continually changes what should be synthesized next.



Coverage summary

Metric	Without Designer (accepted data)	With Designer (actual accepted data)
Distinct task families	4	6
Top-family share	93.6%	23.4%
Missing families	2	0
Status	Severe collapse	Full coverage

Without the Designer, synthesis collapses to ***flight_change*** and completely misses ***refund_or_compensation*** and ***loyalty_or_membership_service***.

With the Designer, ***all six task families*** are covered, with natural variation around the *ideal uniform share*, demonstrating that high-level planning is necessary for reliable task-space exploration.

Figure 6: Role of the Designer in task-space exploration on τ^2-Bench Airline. A Planner–Generator pipeline without explicit high-level allocation collapses toward `flight_change`, whereas introducing the Designer separates global synthesis allocation from local task construction and restores coverage across all six task families.

B.4 Comparative Structural Analysis of Synthetic Data

Beyond downstream task performance, we directly examine how the experience synthesized by EvoSyn differs from existing synthetic corpora at the data-structure level. We conduct three complementary analyses under matched data budgets: *unique structural support* measures whether EvoSyn reaches interaction structures absent from existing corpora, *order-sensitive contrastive support* evaluates local structural variation among similar action compositions, and *balanced structural diversity* examines how broadly and evenly synthesis mass is distributed across the resulting interaction space. Together, these analyses characterize not only whether EvoSyn generates different data, but whether synthesis-space evolution progressively expands, enriches, and redistributes the structural support of the training experience.

B.4.1 Unique and Expanding Structural Support

We further compare the structural support of EvoSyn with publicly released synthetic corpora constructed for the same environment. We focus on Airline, where EvoSyn, AReaL-SEA (Gao et al., 2026), APIGen-MT (Prabhakar et al., 2026), and WRIT (Gu et al., 2026) all provide directly domain-matched synthetic data, and perform all comparisons under matched data budgets. To account for differences in read-action logging across released datasets, we canonicalize each example by its *decisive* tool-interaction sequence, retaining write, compute, and transfer actions. We define the *unique-support fraction* as the proportion of a method's distinct structural signatures that are absent from the pooled support of all other methods. As shown in Figure 7, EvoSyn contributes the largest unique structural support at 73.0%, compared with 69.0% for AReaL-SEA, 49.8% for APIGen-MT, and 25.5% for WRIT. Its pairwise overlap with these corpora is also small, with Jaccard similarities of only 0.107–0.128. This indicates that EvoSyn is not simply reproducing structural patterns already present in existing corpora, but contributes a largely complementary set of executable interaction structures under the same data budget.

More importantly, this complementarity emerges progressively through synthesis-space evolution. Relative to the union of all matched baseline corpora, the fraction of EvoSyn's support that is baseline-unseen increases sharply from 13.6% in R1 to 79.1% in R2 and remains consistently high through R3 and R4 (76.5% and 79.4%). Meanwhile, the fraction of individual examples realizing baseline-unseen structures increases from 1.7% to 15.0%. The sharp transition after the initial

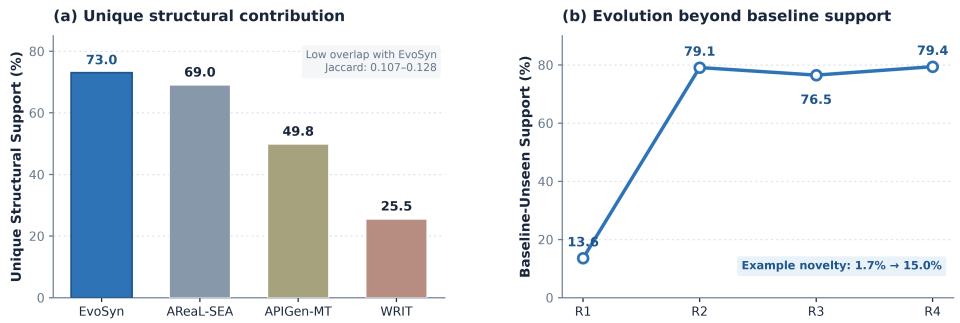


Figure 7: Structural complementarity of EVOSYN relative to domain-matched synthetic-data baselines on Airline. **Left:** fraction of each method's distinct decisive interaction support that is absent from the pooled support of all other methods under matched data budgets. EVOSYN provides the largest unique structural contribution. **Right:** fraction of EVOSYN's structural support that is absent from the union of all matched baseline corpora across synthesis rounds. Iterative synthesis-space evolution progressively moves generation toward interaction structures not represented by existing synthesis pipelines.

round suggests that the gain is not explained by the initial synthesis prior alone: once learner feedback begins revising the synthesis space, subsequent rounds increasingly move generation toward interaction structures absent from existing pipelines. These results provide direct data-side evidence that feedback-guided evolution does more than generate additional variants within a fixed interaction distribution. Instead, it expands the realized structural support of the training experience and continually redirects synthesis toward previously underrepresented regions of the agentic interaction space.

B.4.2 CONTRASTIVE SUPPORT FOR ORDER-SENSITIVE INTERACTION STRUCTURE

Structural coverage alone does not reveal whether a synthetic corpus captures meaningful variation within similar interaction compositions. In agentic tasks, examples involving the same or overlapping sets of actions may still require different execution orders, making order-sensitive contrast an important source of training signal. We therefore measure whether structurally similar examples exhibit distinct decisive-action orderings. Figure 8 reports the clearest domain-matched comparison on Telecom, where decisive-action traces are directly comparable across EVOSYN and AReaL-SEA without the read-logging asymmetry present in other domains. EVOSYN achieves a tool-order contrastive support rate of 53.1%, compared with only 9.6% for AReaL-SEA, corresponding to a $5.5\times$ increase. This indicates that EVOSYN is substantially more likely to realize alternative order-sensitive interaction structures rather than repeatedly reproducing a single canonical ordering over similar action compositions. Complementing the unique-support analysis above, this result shows that synthesis-space evolution improves not only the global breadth of structural support, but also the local structural variation within that support, exposing execution distinctions that cannot be captured by tool-set coverage alone.

B.4.3 BALANCED STRUCTURAL DIVERSITY ACROSS INTERACTION PATTERNS

Broad structural support does not necessarily imply a useful synthetic distribution: a corpus may contain many distinct interaction patterns while still concentrating most of its mass on a small number of dominant modes. We therefore complement the support-based analyses above with distributional measures that quantify how evenly synthesized experience is spread across factors and trajectory structures under matched data budgets. As shown in Figure 9, EVOSYN exhibits a particularly strong advantage on Retail, achieving a trajectory-shape evenness of 0.460, compared with 0.342 for ToolMind (Yang et al., 2025), 0.148 for AReaL-SEA, and 0.088 for both APIGen-MT and WRIT. This represents a $1.35\times$ improvement over the strongest competing method and a $3.12\times$ improvement over AReaL-SEA, with EVOSYN's 95% interval ($[0.416, 0.500]$) remaining disjoint from those of all four baselines. This result indicates that the additional structures introduced by EVOSYN receive substantial synthesis mass rather than appearing only as isolated long-tail examples.

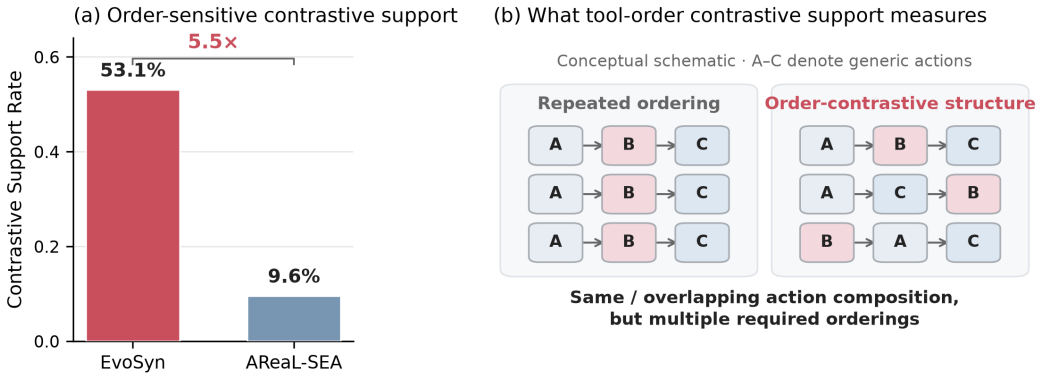


Figure 8: Order-sensitive contrastive structure in synthetic data. **Left:** On Telecom, EvoSyn exhibits substantially higher tool-order contrastive support than the domain-matched AReaL-SEA baseline (53.1% vs. 9.6%, 5.5×). **Right:** Conceptual illustration of the metric. Repeated use of the same ordering provides little order-level contrast, whereas multiple orderings over the same or overlapping action composition create richer order-sensitive structural variation.

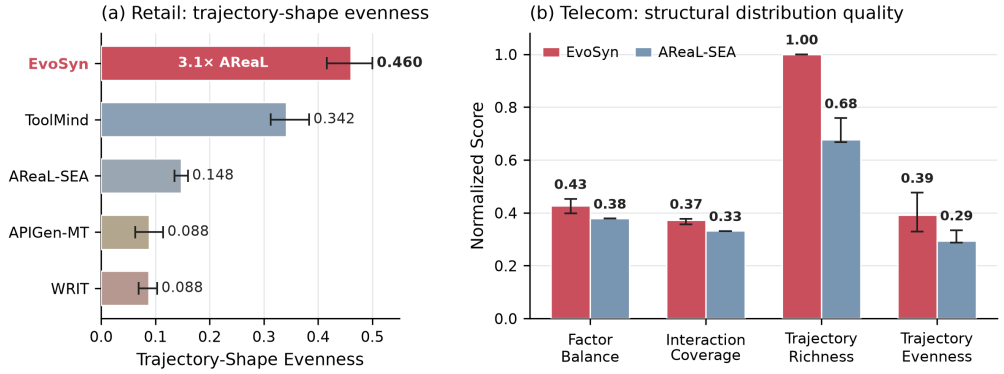


Figure 9: Balanced structural diversity of synthetic agent data. **Left:** On Retail, EvoSyn achieves the highest trajectory-shape evenness among five domain-matched synthetic corpora under the same data budget, exceeding the strongest competing method by 1.35× and AReaL-SEA by 3.12×. **Right:** On Telecom, where the compared ground-truth traces have directly comparable decisive-action structure, EvoSyn provides higher factor balance, interaction coverage, trajectory richness, and trajectory evenness than AReaL-SEA. Together, these results show that EvoSyn distributes synthesis mass across a broader and more balanced set of interaction structures rather than concentrating on a small number of dominant trajectory patterns.

The same distributional advantage is reflected on Telecom, where the compared ground-truth trajectories admit directly comparable decisive-action structure. We additionally report trajectory richness, defined as the fraction of distinct trajectory patterns covered relative to the shared pattern union. Relative to AReaL-SEA, EvoSyn achieves higher factor balance (0.427 vs. 0.378), interaction coverage (0.372 vs. 0.331), trajectory richness (1.000 vs. 0.678), and trajectory evenness (0.392 vs. 0.293).

B.5 Targeted Slice Statistics for Capability-Gap Adaptation

Table 5 provides the slice-level statistics underlying the two capability-gap transitions visualized in Figure 5 and Figure 10. For each transition, we examine interaction slices associated with the promoted synthesis factor or explicitly emphasized by the next-round Designer allocation. During R1→R2, the learner exhibits large gaps on write-heavy interactions: `multi_write` and `high_write` have capability gaps of 0.571 and 0.667. Their verified-data proportions subsequently

Table 5: Targeted capability evolution on CAR-Bench Base. We report the promoted or directly targeted slices in each transition. ΔProp measures the change in verified synthetic-data proportion from the current round to the next round, and Gain measures the corresponding held-out success-rate change after training.

Transition	Targeted Slice	Pre-round SR	Capability Gap	ΔProp	Post-round Gain	Targeted Improvement
R1 → R2	`multi_write`	0.429	0.571	+0.340	+0.262	✓
	`high_write`	0.333	0.667	+0.145	+0.389	✓
R2 → R3	`navigation`	0.463	0.537	+0.690	+0.148	✓
	`multi_read`	0.410	0.590	+0.804	+0.205	✓
	`read_then_set`	0.590	0.410	+0.354	+0.103	✓
R1 → R2 aggregate		Allocation–Gain $\rho = 0.571$			Targeted Improvement Rate $= 2/2$	
R2 → R3 aggregate		Gap–Allocation $\rho = 0.964$			Allocation–Gain $\rho = 0.750$, TIR $= 3/3$	

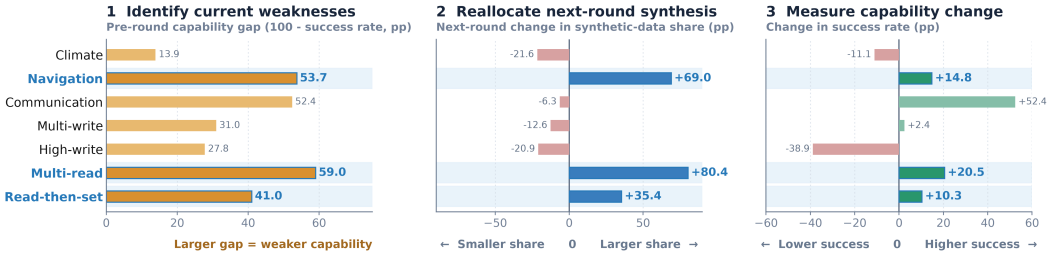


Figure 10: Capability-gap-driven synthesis adaptation on CAR-Bench from R2 to R3. As the learner's residual weakness shifts toward read-intensive decision making, EvoSyn substantially increases the verified-data share of `navigation`, `multi_read`, and `read_then_set`, all of which subsequently improve after training.

increase by 0.340 and 0.145, followed by success-rate gains of 0.262 and 0.389, respectively. After this write-heavy weakness is reduced, R2→R3 shifts toward read-intensive decision making, as shown in Figure 10. `navigation`, `multi_read`, and `read_then_set` receive substantial increases in synthesis allocation and all improve after training, yielding gains of 0.148, 0.205, and 0.103.

The aggregate statistics further quantify this feedback-to-decision alignment. In R2→R3, pre-round capability gaps are strongly aligned with next-round synthesis shifts (Spearman $\rho = 0.964$), indicating that synthesis effort is systematically redirected toward regions where the learner remains weak. All directly upweighted slices improve after the corresponding training round, giving targeted improvement rates of $2/2$ for R1→R2 and $3/3$ for R2→R3. The positive allocation–gain association in R2→R3 ($\rho = 0.750$) provides additional descriptive evidence that these targeted shifts coincide with subsequent capability improvement. These slice-level results show that EvoSyn follows a moving capability boundary rather than repeatedly oversampling a fixed notion of task difficulty.

B.6 No-Feedback Control for Synthesis-Space Expansion

We further test whether the factors introduced during synthesis-space expansion are genuinely driven by learner feedback or could be recovered through generic reasoning over the synthesis space and environment alone. For each CAR-Bench transition, we compare the feedback-conditioned Reflector with a no-feedback control that receives the same synthesis interface, environment description, and tool schema, but no post-training failures, coverage statistics, diagnostic probes, or learner-feedback summaries. As shown in Table 6, the feedback-conditioned Reflector identifies both factors subsequently adopted by EvoSyn: `num_write` after R1 and `num_read` after R2. These factors are also operationally reflected in the next-round data, with the verified `multi_write` share increasing from 0.232 to 0.571 and the `multi_read` share from 0.056 to 0.860.

In contrast, the no-feedback control proposes six semantically plausible dimensions, including temporal reasoning, entity resolution, constraint interaction, information synthesis, and target resolution, but none recovers the write- or read-volume factors that are actually exposed by the learner's measured failures. The resulting adopted-factor recovery is therefore $2/2$ with feedback versus $0/6$

Table 6: Detailed no-feedback control for Reflector breadth on CAR-Bench Base. The feedback-conditioned Reflector uses empirical failure and coverage evidence, while the no-feedback control receives only the ontology, environment description, and tool schema.

Transition	Empirical Feedback Signal	Feedback-Conditioned Discovery	Adoption Evidence	No-Feedback Proposals	Control Outcome
R1 → R2	Tasks with 5+ write actions have high failure, indicating that write-action quantity is a missing difficulty coordinate beyond the coarse tool_use_pattern dimension.	Promotes num_write with buckets {0, 1, 2, 3–4, 5+}. The factor is novel, evidence-grounded, actionable, and adopted.	num_write is added to the next synthesis interface. The verified multi_write share increases from 0.232 to 0.571.	temporal_reasoning_depth, entity_resolution_complexity, constraint_interaction.	Misses the adopted write-volume factor; 0/3 proposals recover the adopted write-volume factor.
R2 → R3	Tasks with 3–4 read actions have high failure and low coverage, exposing an under-sampled read-heavy decision region.	Promotes num_read with buckets {0, 1, 2, 3–4, 5+}. The factor is novel, evidence-grounded, actionable, and adopted.	num_read is added to the next synthesis interface. The verified multi_read share increases from 0.056 to 0.860.	temporal_reasoning, information_synthesis_depth, target_resolution.	Misses the adopted read-volume factor; 0/3 proposals recover the adopted read-volume factor.
Overall	Feedback-conditioned Reflector: 2 proposed factors, both evidence-grounded and both adopted.		No-feedback control: 6 proposed factors, none grounded in observed failure or coverage evidence, and none matching the adopted write/read-volume factors.		Adopted-factor recovery: 2/2 vs. 0/6.

without feedback. This control shows that synthesis-space expansion is not equivalent to asking a strong LLM to invent additional task dimensions: learner feedback identifies the specific missing coordinate along which the current task agent fails, allowing the Reflector to convert that weakness into a reusable and actionable generation control.

B.7 Controllability of Evolved Synthesis Factors

A revisable synthesis space is useful only if its factors serve as actionable generation controls rather than post-hoc descriptors of generated data. We therefore test whether a factor value specified before generation is faithfully realized in the resulting environment-grounded trajectory. We use `num_write` as a representative directly measurable factor: for each blueprint–task pair, we recover the requested value from the synthesis plan and independently recompute the realized value from the ground-truth tool trajectory. To isolate generation controllability from round-wise evolution, we evaluate R2 synthesis runs across multiple domains and synthesis backbones. As shown in Table 7, requested and realized values exhibit near-perfect agreement in every setting, with exact-match accuracy of 99.6–100% and Spearman $\rho = 0.997$–1.000 across thousands of blueprint–task pairs. Importantly, this agreement is not an artifact of the Verifier filtering non-conforming generations: the explicit axis-conformance check rejects only 0.15% of candidates in Retail and 1.0% in CAR-Bench, while candidates rejected for unrelated reasons still retain requested–realized agreement of 100% and 98.3%, respectively. These results show that the synthesis pipeline can reliably translate an abstract factor specification into the intended interaction structure, supporting the synthesis space as an operational control interface for feedback-driven allocation and evolution.

B.8 Capability-Gap Alignment of Synthesis Reallocation

We further examine whether feedback-driven synthesis reallocation is systematically aligned with the learner's residual capability gaps rather than being driven by a small number of selected cases. We focus on the CAR-Bench R2→R3 transition and evaluate all seven measurable interaction slices. For each slice, we define the pre-round capability gap as $1 - \mathrm{SR}_{R2}$ and quantify the next-round

Table 7: Controllability of the synthesis coordinate across R2 synthesis runs. We compare the value requested in the synthesis plan with the value independently recomputed from the realized ground-truth trajectory.

Domain	Synthesis LLM	Exact Match	Spearman ρ
CAR-Bench	Claude Opus 4.5	99.6%	0.998
Retail	Claude Opus 4.5	100.0%	1.000
Airline	Qwen3-30B	100.0%	1.000
CAR-Bench	Qwen3-235B	99.7%	0.997

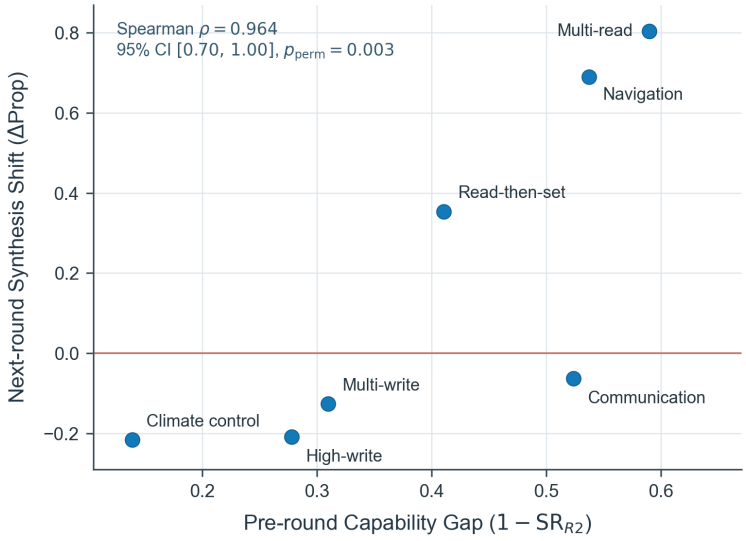


Figure 11: Capability-gap-driven synthesis reallocation on CAR-Bench from R2 to R3. Each point denotes an interaction slice. The pre-round capability gap ($1 - \mathrm{SR}_{R2}$) is strongly aligned with the subsequent change in synthesis allocation (Spearman $\rho = 0.964$, 95% CI $[0.70, 1.00]$, permutation $p = 0.003$), showing that EvoSyn shifts synthesis effort toward regions where the current learner remains weak.

synthesis shift by the change in its proportion of verified synthetic data. As shown in Figure 11, the two quantities are strongly correlated (Spearman $\rho = 0.964$, 95% CI $[0.70, 1.00]$, permutation $p = 0.003$). Interaction slices with larger residual gaps, such as `multi_read` and `navigation`, receive substantially greater synthesis emphasis, whereas regions in which the learner is already stronger are generally downweighted. In contrast, under the Fixed-State Synthesis baseline, the same learner-side capability gaps are negatively correlated with subsequent changes in data proportion ($\rho = -0.964$), indicating that simply accumulating additional synthetic data does not reproduce this learner-aligned redistribution. The observed reallocation is also consistent with subsequent learning outcomes. All three slices that receive increased synthesis emphasis in R3, namely `multi_read`, `navigation`, and `read_then_set`, improve after training by $+0.205$, $+0.148$, and $+0.103$, respectively. These results provide direct evidence that EvoSyn translates residual learner weaknesses into targeted synthesis decisions and continuously redirects future experience as the learner's capability boundary changes.

B.9 Structural Support Expansion Across Synthesis Rounds

We next examine whether synthesis-space evolution expands the structural support of the generated experience or simply produces additional samples within an already covered interaction regime. To abstract away surface-level linguistic variation, we represent each verified example by its ground-truth tool-call sequence and measure the cumulative number of distinct interaction sequences. As shown in Table 8, EvoSyn exhibits sustained structural growth across synthesis rounds, whereas Static synthesis increasingly saturates. On CAR-Bench, the number of distinct sequences grows

Table 8: Structural-support growth across synthesis rounds. We report the cumulative number of distinct ground-truth tool-call sequences covered by the verified synthetic data.

Domain	Method	R1	R2	R3
CAR-Bench	EvoSyn	270	497	**952**
	Static	276	398	415
Airline	EvoSyn	28	111	**122**
	Static	23	27	36

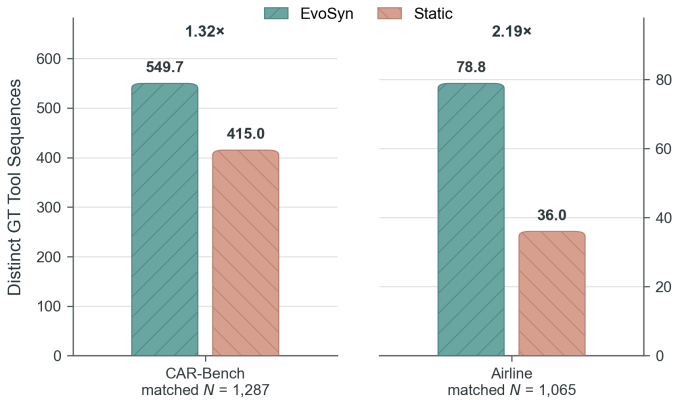


Figure 12: Size-matched structural support at the final synthesis round. The cumulative EvoSyn pool is subsampled to match the number of accepted examples in the Static baseline, and structural support is measured by the number of distinct ground-truth tool-call sequences. After controlling for data volume, EvoSyn still covers $1.32\times$ more interaction sequences on CAR-Bench and $2.19\times$ more on Airline.

from 270 in R1 to 497 in R2 and 952 in R3 under EvoSyn, while Static reaches only 415 by R3. The contrast is even sharper on Airline, where EvoSyn expands from 28 to 111 and 122 sequences across rounds, compared with only 36 for Static at R3. This divergence indicates that later EvoSyn rounds continue to introduce new executable interaction structures rather than repeatedly sampling from the support established in earlier rounds.

Because the cumulative EvoSyn pools contain more accepted examples, we further control for data volume through size-matched subsampling at the final round. For each domain, we subsample the EvoSyn pool to exactly match the number of accepted examples in Static and recompute structural support. Figure 12 shows that the advantage remains substantial: at the same $N = 1{,}287$, EvoSyn covers 549.7 distinct sequences on CAR-Bench compared with 415.0 for Static, a $1.32\times$ increase. On Airline, it covers 78.8 sequences versus 36.0 at the same $N = 1{,}065$, corresponding to a $2.19\times$ increase. The persistence of this gap after controlling for sample count shows that the structural expansion cannot be explained by data accumulation alone. Instead, revising the synthesis state changes the support of the generated experience, enabling controllable interaction factors to be realized through a broader set of executable behaviors.

B.10 Do Promoted Factors Capture Previously Unmodeled Learner Weakness?

We next examine whether factors introduced during synthesis-space expansion capture genuine learner weaknesses that are not adequately represented by the current synthesis space. For each promoted factor, we retrospectively annotate the corresponding interaction property on held-out evaluation tasks and analyze the learner *before* that factor is added, so that the diagnosis is not confounded by subsequent factor-guided synthesis or training. To quantify the additional failure signal carried by the promoted coordinate, we compute the conditional mutual information $I(Y_{\text{fail}}; d_{\text{new}} \mid \mathcal{O}_r)$, where $\mathcal{O}_r$ denotes the synthesis space before promotion. This quantity measures how much information

Table 9: Residual learner-failure information captured by promoted synthesis factors. All statistics are computed using the learner before the corresponding factor is introduced. Conditional mutual information measures the additional failure information provided by the promoted factor after conditioning on the existing synthesis space $\mathcal{O}_r$.

Domain	**Promoted Factor**	$\mathbf{I}(\mathbf{Y}_{\text{fail}}; \mathbf{d}_{\text{new}} \mid \mathcal{O}_\mathbf{r})$ **(nats)**
Airline	`num_write`	0.122
CAR-Bench	`num_write`	**0.168**
CAR-Bench	`num_read`	0.134

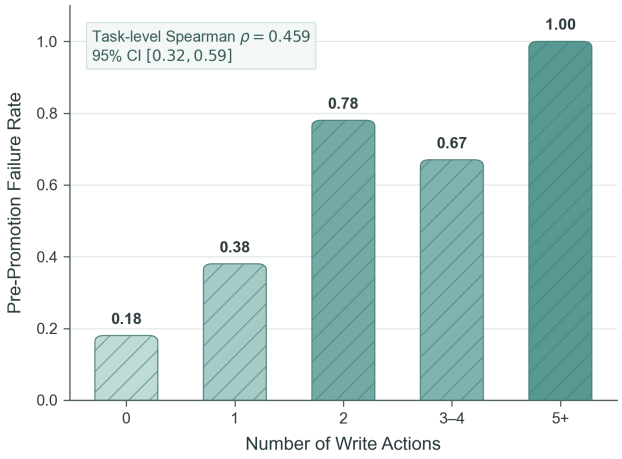


Figure 13: Pre-promotion learner failure rate across `num_write` buckets on Airline. Tasks requiring more state-changing write actions are substantially more failure-prone before `num_write` is introduced as an explicit synthesis coordinate, with a positive task-level association between write complexity and learner failure (Spearman $\rho = 0.459$, 95% CI $[0.32, 0.59]$).

about learner failure remains attributable to the new factor after conditioning on the interaction factors already represented in the current synthesis space.

As shown in Table 9, the promoted factors retain substantial additional failure information across representative expansion cases. On Airline, `num_write` contributes 0.122 nats beyond the existing synthesis coordinates. On CAR-Bench, the corresponding values are 0.168 nats for `num_write` and 0.134 nats for `num_read`. These positive values indicate that the promoted factors are not redundant with the pre-existing synthesis representation. Instead, they capture residual failure-relevant structure that remains visible even after accounting for the factors already available to the synthesizer.

Figure 13 provides an intuitive illustration on Airline. Before `num_write` is introduced as an explicit synthesis coordinate, the learner's failure rate rises sharply with the number of required write actions, increasing from 0.18 for tasks with no write action to 1.00 for tasks with five or more, with a positive task-level rank association of $\rho = 0.459$. Because all measurements are taken from the pre-promotion learner, this pattern reflects a capability gap that precedes and motivates the later synthesis-space revision. The result therefore supports a stronger interpretation than post-hoc relabeling: promoted factors expose previously unmodeled failure structure and convert it into actionable synthesis coordinates for subsequent rounds.

B.11 Do Promoted Factors Capture Failure Structure Beyond Generic Complexity?

A natural alternative explanation for synthesis-space expansion is that the promoted factors merely recover generic task difficulty. For example, tasks requiring more reads or writes may simply involve longer tool-use trajectories and therefore be harder for the learner. We test this hypothesis by using

Table 10: Promoted synthesis factors capture learner-failure structure beyond generic interaction complexity. We use the total number of tool calls as a domain-agnostic complexity variable g. All information quantities are reported in nats.

Domain	**Promoted Factor** z	$\mathrm{NMI}(\mathbf{z}, \mathbf{g})$	$\mathbf{I}(\mathbf{Y}_{\mathrm{fail}}; \mathbf{g})$	$\mathbf{I}(\mathbf{Y}_{\mathrm{fail}}; \mathbf{z} \mid \mathbf{g})$
Airline	`num_write`	0.216	0.032	**0.241**
CAR-Bench	`num_read`	0.420	0.109	**0.075**

Table 11: Observed marginal learning gain on CAR-Bench. Round-wise gain is normalized by the number of newly added verified examples; endpoint efficiency is measured relative to the base Qwen3-8B agent ($\mathrm{SR} = 0.360$).

Transition	**ΔSR**	**Added Examples**	**ΔSR / 1K**
R1→R2	+0.020	0.97K	+0.021
R2→R3	**+0.120**	0.78K	**+0.153**
Final endpoint efficiency relative to the base agent			
EvoSyn R3	+0.400	1.996K	**+0.200**
Static R3	+0.210	1.287K	+0.163

the total number of tool calls, discretized into tertile-based buckets, as a domain-agnostic complexity variable g. For each promoted factor z, we measure its dependence on generic complexity using $\mathrm{NMI}(z, g)$ and quantify the residual failure information it provides after conditioning on g through $I(Y_{\mathrm{fail}}; z \mid g)$.

As shown in Table 10, Airline `num_write` is only weakly associated with overall tool count ($\mathrm{NMI} = 0.216$), yet retains 0.241 nats of failure information after generic complexity is known, compared with only 0.032 nats carried by tool count itself. CAR-Bench `num_read` shows a moderate association with tool count ($\mathrm{NMI} = 0.420$), but still contributes 0.075 nats of residual failure information beyond it. These results show that the promoted factors are not redundant proxies for trajectory length or overall interaction complexity. Instead, they identify more specific failure-relevant interaction properties, supporting synthesis-space expansion as a mechanism for exposing structured capability gaps rather than merely generating globally harder tasks.

B.12 Observed Marginal Learning Gain Across Synthesis Rounds

We further examine how downstream improvement changes as new verified experience is introduced across synthesis rounds. Because both the learner state and training-data composition evolve over time, this analysis is descriptive rather than a causal estimate of per-example utility. We normalize the observed change in task success by the amount of newly added verified training data. As shown in Table 11, the CAR-Bench R1→R2 transition improves success rate by $+0.020$ with approximately 0.97K additional examples, corresponding to an observed gain of $+0.021$ per 1K examples. In comparison, R2→R3 improves success by $+0.120$ with only 0.78K additional examples, yielding $+0.153$ per 1K examples. This is approximately seven times the normalized gain observed in the preceding round. Notably, R2→R3 is also the transition in which synthesis is redirected toward the learner's newly exposed read-heavy capability gaps, linking the strongest synthesis-space revision with the largest observed learning return from newly added experience.

The endpoint comparison provides a complementary view after accounting for cumulative training-set size. Relative to the base Qwen3-8B agent, EvoSyn R3 gains 0.400 success-rate points using 1,996 verified examples, corresponding to $+0.200$ per 1K examples, compared with $+0.210$ and $+0.163$ per 1K for Static. While these ratios should not be interpreted as intrinsic sample-level utility, they provide additional evidence that feedback-guided synthesis can improve not only the amount and structural coverage of training experience, but also the observed learning return associated with later synthesis rounds.

Table 12: Intrinsic quality of synthetic data over rounds on CAR-Bench Base. Each row is computed on the cumulative data up to the corresponding round. All intrinsic metrics are normalized to $[0, 1]$, where higher is better.

Setting	Round	Factor Coverage	Factor Balance	Interaction Coverage	Trajectory Pattern Diversity	Success Rate
Static Multi-Round	R1	0.640	0.492	0.339	0.688	0.607
	R2	0.705	0.470	0.343	0.735	0.627
	R3	0.787	0.468	0.351	0.750	0.613
EvoSyn	R1	0.640	0.492	0.339	0.688	0.607
	R2	0.760	0.479	0.351	0.750	0.655
	R3	0.893	0.508	0.373	0.938	0.726

B.13 Evolution Case Study

B.13.1 CAR-Bench

Table 12 quantitatively verifies the multi-round evolution illustrated in Figure 4. Both Static Multi-Round and EvoSyn start from the same R1 data, isolating the effect of feedback-driven synthesis decisions in later rounds. Static Multi-Round continues to add data under a fixed synthesis interface and allocation, but its downstream performance does not improve consistently, moving from 0.607 to 0.627 and then dropping to 0.613. In contrast, EvoSyn improves monotonically from 0.607 to 0.655 and finally to 0.726, suggesting that later rounds become increasingly useful for task-agent learning rather than merely larger in size. The intrinsic metrics show the same pattern. By R3, EvoSyn achieves substantially higher factor coverage than Static Multi-Round (0.893 vs. 0.787) and much higher trajectory-pattern diversity (0.938 vs. 0.750), while also improving factor balance and interaction coverage. These results support the mechanism behind EvoSyn: post-training feedback does not only evaluate the current synthetic pool, but changes the controllable synthesis interface and redirects subsequent generation toward interaction regions that static sampling fails to cover effectively.

Table 13 provides the detailed evidence behind the CAR-Bench evolution trace shown in Figure 3. The table expands each round transition into four components: the post-training feedback observed after the previous round, the Reflector's ontology decision, the Designer's next-round synthesis focus, and the resulting synthesis change. In the R1→R2 transition, the key signal is a failure concentration in write-heavy vehicle-control tasks: the 5+ write bucket has a failure rate of 0.92, while the initial interface only represents coarse tool-use patterns and cannot directly control the number of state-changing actions. The Reflector therefore promotes `num_write`, allowing the Designer to allocate synthesis budget toward high-write trajectories. In the R2→R3 transition, feedback shows that the write dimension has become meaningful, but a new gap appears in read-heavy tasks: the `num_read`=3–4 bucket has a failure rate of 0.63 while covering only 3.9% of the data. The Reflector consequently promotes `num_read`, shifting the next round toward read-heavy trajectories. This detailed trace supports the interpretation that EvoSyn evolves by converting post-training feedback into new controllable synthesis dimensions, rather than merely resampling more tasks from a fixed interface.

B.13.2 Tau-Bench Airline

The Airline domain provides a complementary case of learning-guided synthesis evolution in a customer-service environment with richer policy constraints and state-changing operations. As shown in Table 15, EvoSyn first identifies a strong distributional and capability mismatch after R1: the verified data is dominated by `flight_change` tasks, while high-write interactions remain under-represented and difficult. The Reflector therefore promotes `num_write` into a controllable dimension, allowing the Designer to explicitly allocate R2 synthesis budget toward multi-write trajectories. After this axis becomes operational, the next round reveals a different source of difficulty: numerical or derived tool arguments that require computation before execution. The Reflector then promotes `computation` as a new synthesis dimension, and R3 shifts toward more balanced coverage of computation-aware interactions. This trace supports the same mechanism observed in CAR-

Table 13: Mechanistic trace of learning-guided synthesis evolution on CAR-Bench Base. Each row reports the concrete feedback evidence used by the Reflector, the resulting ontology decision, the Designer's budget shift, and the observed synthesis change.

Transition	Feedback Evidence	Reflector Decision	Designer Focus	Observed Synthesis Change
R1 → R2	R1 evaluation shows that the hardest cases concentrate in high-write vehicle-control tasks: the 5+ write bucket has failure rate 0.92. The R1 data is skewed toward lower write counts (1 write: 54%; 2 writes: 14%; 3–4 writes: 16%; 5+ writes: 16%). The existing `tool_use_pattern` dimension captures action-sequence shape, but not the quantity of state-changing writes.	Promote `num_write_actions` into a new controllable dimension `num_write`. This separates write-action quantity from the existing tool-pattern dimension and introduces a new axis for execution-heavy tasks.	Increase synthesis budget for high-write regions, especially `num_write`=3–4 and 5+, under the base-only CAR environment with a weak macro prior.	R2 generates multi-write vehicle-control tasks that the initial interface could not directly target. The round accepts 971 verified tasks, bringing the cumulative verified set to 1212.
R2 → R3	After R2, the `num_write` axis is validated: failure on 5+ write tasks drops from 0.92 to 0.53, and the failure gradient becomes monotonic across write-count buckets. New feedback shows that read-heavy tasks remain both difficult and under-covered: `num_read`=3–4 has failure rate 0.63 but only 3.9% coverage.	Promote `num_write` from experimental to standard. Promote `num_read` into a new controllable axis with buckets 0/1/2/3–4/5+. The new axis is considered orthogonal to `num_write`, since read actions correspond to information gathering and decision preparation, while write actions correspond to execution.	Shift macro budget toward the under-covered high-failure read region, especially `num_read`=3–4, while keeping `num_write` as a standard seed dimension.	R3 produces read-heavy trajectories: the average number of reads increases to 5.02 per task, compared with 0.97 in R2. The round accepts 784 verified tasks and expands the data toward read–decide–act interactions.

Bench: feedback does not merely filter generated tasks, but changes the coordinates along which future tasks can be deliberately synthesized.

Table 14 further shows that this mechanism translates into stronger multi-round behavior than Static Multi-Round synthesis. Both methods start from the same R1 data, but Static Multi-Round only accumulates more samples under a fixed interface and allocation, leading to unstable performance from 0.648 to 0.633 and then 0.660. In contrast, EvoSyn improves steadily from 0.648 to 0.693 and finally to 0.713. The intrinsic data-quality metrics show a similar divergence: by R3, EvoSyn achieves higher factor coverage, factor balance, interaction coverage, complexity coverage, and trajectory-pattern diversity than the static baseline. The largest gap appears in trajectory-pattern diversity, where EvoSyn reaches 0.769 compared with 0.538 for Static Multi-Round, indicating that feedback-guided evolution expands not only the labels attached to synthetic tasks but also the underlying tool-use and state-transition structures. Together with the CAR-Bench results in the main text, this Airline case suggests that the self-evolution loop generalizes across domains: later rounds become more useful because the synthesis process itself is revised according to observed failures, coverage gaps, and newly exposed interaction factors.

C Related Work

C.1 Synthetic Data for Post-Training and Self-Evolution

Synthetic data has become a major mechanism for expanding the supervision available to large language models beyond manually curated datasets. A broad line of work studies how to bootstrap

Table 14: Intrinsic quality of synthetic data over rounds on τ^2-Bench Airline. Each row is computed on the cumulative data up to the corresponding round. All metrics are normalized to $[0, 1]$, where higher is better.

Setting	Round	Factor Coverage	Factor Balance	Interaction Coverage	Trajectory Pattern Diversity	Success Rate
Static Multi-Round	R1	0.742	0.264	0.311	0.462	0.648
	R2	0.760	0.309	0.330	0.538	0.633
	R3	0.778	0.368	0.403	0.538	0.660
EvoSyn	R1	0.742	0.264	0.311	0.462	0.648
	R2	0.785	0.453	0.399	0.692	0.693
	R3	0.891	0.450	0.455	0.769	0.713

Table 15: Mechanistic trace of learning-guided synthesis evolution on τ^2-Bench Airline. Each row reports the concrete feedback evidence used by the Reflector, the resulting ontology decision, the Designer's budget shift, and the observed synthesis change.

Transition	Feedback Evidence	Reflector Decision	Designer Focus	Observed Synthesis Change
R1 → R2	R1 data is highly skewed: `flight_change` accounts for 93.6% of verified tasks. Failure probes further show that the 3–4 write-action bucket has failure rate 1.00, indicating that high-write state-changing interactions are difficult but not explicitly controllable in the initial interface.	Promote `num_write_actions` into a new controllable axis `num_write` with buckets 0/1/2/3–4/5+. The axis is introduced as a failure-driven experimental dimension because the probe shows strong failure correlation and is generator-realizable.	Increase macro budget for the previously undercovered high-write buckets, especially `num_write`=3–4 and 5+. Lower write-count buckets remain medium priority.	R2 blueprints shift strongly toward high-write synthesis: the 3–4 and 5+ buckets together account for 57% of realized blueprints. Accepted tasks span all five write-count buckets, showing that `num_write` becomes an operational synthesis control rather than only a diagnostic label.
R2 → R3	After R2, `num_write` is validated as a meaningful factor: the 5+ write bucket remains difficult with failure rate 0.86, while write-count coverage becomes healthier. A new probe, `num_numeric_fields`, shows failure rate 1.00 at value 1, revealing difficulty with numerical or derived tool arguments before execution.	Promote `num_write` from experimental to standard. Then promote `num_numeric_fields` into a new semantic axis `computation`, with values `none`, `aggregate`, `constrained_opt`, and `conditional`.	Tilt macro budget toward all four buckets of the new `computation` axis. The goal is to systematically explore different forms of computation-before-action behavior while retaining `num_write` as a standard dimension.	R3 blueprints become approximately balanced across computation types, with each computation bucket covering about one quarter of the realized blueprint distribution. The synthesis focus therefore shifts from only controlling write-action complexity to also covering numerical decision-making and conditional tool execution.

instruction-following data from seed tasks, teacher models, unlabeled corpora, or model-generated responses, including Self-Instruct Wang et al. (2023), WizardLM Xu et al. (2024), and instruction backtranslation Li et al. (2023). Subsequent methods improve the diversity, realism, and usefulness of synthetic corpora through multi-agent simulation, feature-targeted generation, filtering, or autonomous data engineering Tang et al. (2025); Li et al. (2026); Cao et al. (2025); He et al. (2025); Tu et al. (2025). These approaches establish synthetic data as a central ingredient in LLM post-training. However, most of them treat data generation as an offline construction process: a corpus is

generated, filtered, and then used to train a learner, while the synthesis process itself is only weakly coupled to the learner's post-training behavior.

Reasoning domains provide a particularly favorable setting for scalable synthetic supervision because many examples admit verifiable answers or checkable solution traces. STaR Zelikman et al. (2022) bootstraps rationales from successful model attempts, while WizardMath Luo et al. (2025), MetaMath Yu et al. (2023), MAmmoTH Yue et al. (2024), and ToRA Gou et al. (2024) expand mathematical or tool-integrated reasoning supervision through instruction evolution, problem transformation, and rationale construction. Large-scale mathematical post-training pipelines such as DeepSeekMath Shao et al. (2024) and NuminaMath Li et al. (2024) further demonstrate the effectiveness of combining synthesized or curated reasoning data with outcome verification. More recent learner-aware methods use model behavior to guide synthesis: for example, SwS Liang et al. (2026) identifies persistent weaknesses and generates new problems around the corresponding concepts. Together, these works show that verifiability and learner feedback can substantially improve synthetic data quality. Nevertheless, their feedback signals are usually mapped to harder problems, corrected rationales, or concept-level variants within a relatively self-contained problem space.

A related line of work studies self-evolving LLMs, where task generation and task solving are placed in a feedback loop. R-Zero Huang et al. (2025) co-evolves a Challenger and Solver through generated problems and solution feedback, and Socratic-Zero Wang et al. (2025b) introduces Teacher, Solver, and Generator roles to construct an increasingly challenging curriculum from diagnosed learner weaknesses. Agent0 Xia et al. (2025), Dr. Zero Yue et al. (2026), and SCOPE Kwan et al. (2026) extend this paradigm to tool-integrated reasoning, multi-turn search, and document-grounded task construction. These methods demonstrate that data generation can be coupled with learner improvement rather than treated as a fixed preprocessing step. Our work shares this closed-loop perspective, but focuses on a different challenge: in interactive agentic environments, useful synthesis must not only generate harder examples, but also discover and control the interaction factors that determine feasibility, tool execution, state transitions, and downstream learning utility.

C.2 Feedback-Driven Agentic Data Synthesis

Synthetic data for interactive agents introduces constraints that are less prominent in self-contained reasoning tasks. A generated example must correspond to a feasible environment state, admit an executable sequence of tool calls, satisfy domain policies, and lead to a verifiable outcome. Recent environment-grounded synthesis methods address these requirements by using exploration, trajectory reconstruction, or state replay to build executable agentic experience. Explorer Pahuja et al. (2025) synthesizes diverse web trajectories through large-scale environment exploration, OS-Genesis Sun et al. (2025) derives executable GUI tasks from successful interactions, AutoPlay Ramrakhya et al. (2025) explores application states before producing grounded tasks, and Trajectory2Task Wang et al. (2026b) converts valid tool trajectories into user-facing tasks. WebAggregator Wang et al. (2026a) further constructs search tasks requiring compositional evidence aggregation. These methods make important progress on feasibility and executability, but their feedback is primarily used to obtain valid instances or broaden trajectory coverage, rather than to revise an explicit synthesis control space according to the trained agent's learning response.

Other work improves the quality of tool-use and multi-turn synthetic data through checkers, validators, and data-centric refinement. EigenData Chen et al. (2026) synthesizes multi-turn tool-use dialogues with executable checkers, while APIGen, APIGen-MT, and TOUCAN-style pipelines construct function-calling or tool-use supervision with varying degrees of automatic validation. DreamGym Chen et al. (2025) scales synthetic tasks and interactions through a learned experience model. These approaches show that agentic data generation benefits from structured procedures, executable verification, and iterative data-quality control. However, validation and repair mainly answer whether a generated sample is well formed, executable, or diverse. They do not by themselves determine which interaction factors should receive more synthesis budget after training, or whether the current set of factors is sufficient to explain the remaining weaknesses of the task agent.

Several recent frameworks further explore continual improvement in agentic settings. CoEvolve Yang et al. (2026) uses trajectory-derived signals such as forgetting and uncertainty to guide subsequent tool-use task synthesis, while Agent-World Dong et al. (2026), SEAD Dai et al. (2026), AgentEvolver Zhai et al. (2025), EvolveR Wu et al. (2025), and EvoFSM Zhang et al. (2026) in-

vestigate the evolution of environments, curricula, workflows, memories, or interaction experiences across different agent tasks. This literature establishes feedback-driven agent improvement as a promising direction. The remaining gap is how to translate feedback into reusable synthesis decisions. In particular, existing methods often adapt by generating more valid instances, repairing failures, or upsampling weak categories, while leaving the underlying synthesis interface largely fixed. EVOSYN instead treats the synthesis interface itself as an evolving object. Its multi-agent loop uses feedback not only to filter or repair samples, but also to support synthesis depth, which reallocates effort across known interaction factors, and synthesis breadth, which promotes previously implicit factors into explicit controls for future generation.

D REPRESENTATIVE AGENT PROMPTS

We provide representative prompt templates for the three agents that directly operate on the evolving synthesis state: the Designer, Planner, and Reflector. The Designer and Reflector are domain-agnostic, while domain-specific tool schemas, policies, environment information, synthesis-space values, and round-specific feedback are injected at runtime. We use Airline as the representative instantiation because it exposes the complete structured synthesis interface used in our τ^2-Bench experiments. For readability, we omit implementation-only scaffolding while retaining the runtime inputs, decision rules, and output structure that determine each agent's behavior.

Designer Prompt

```
You are the Designer for an interactive tool-use task synthesizer.
You decide which synthesis directions the next batch should cover
and how much generation budget each direction should receive.  You
do not construct concrete tasks or instantiate specific entities.
A downstream Planner grounds your decisions into executable task
blueprints.

RUNTIME INPUTS

DOMAIN: airline

DOMAIN CONTEXT SUMMARY:
<environment and task context>

AVAILABLE TOOLS SUMMARY:
<tool names, arguments, and functional descriptions>

POLICY SUMMARY:
<domain policies and execution constraints>

CURRENT SYNTHESIS SPACE:
{
"task_family":  [
"flight_booking",
"flight_change",
"flight_cancellation",
"refund_or_compensation",
"seat_or_baggage_service",
"loyalty_or_membership_service"
],
"expected_outcome":  ["fulfill", "refuse"],
"policy_or_constraint_focus":  [...],
"tool_use_pattern":  [...],
"interaction_challenge":  [...],
"difficulty_profile":  ["easy", "medium", "hard"],
<newly promoted factors, e.g.,
"num_write":  ["0", "1", "2", "3-4", "5+"]>
}
```

```
STEERING SIGNAL: one of the following is provided at each round.

  (1) NEWLY ADDED FACTOR:
<new factor, admissible values, and currently under-covered
regions>

  (2) LEARNER FEEDBACK:
<coverage and post-training task-agent performance statistics>

  (3) COLD-START PRIOR:
<initial exploration prior when no learner feedback is yet
available>

INSTRUCTIONS

For each synthesis direction:
-- specify the capability or interaction structure to target;
-- assign an objective type such as under-covered,
failure-targeted, exploration, or repair;
-- optionally specify preferred values as soft preferences;
-- optionally specify allowed values as hard constraints;
-- optionally specify forbidden values as hard exclusions;
-- identify undesirable cross-dimension patterns to avoid;
-- assign a generation budget level;
-- provide a concise rationale for why the direction is useful.

All referenced keys must belong to the current synthesis space.
A proposed direction should describe a capability or interaction
structure, rather than a concrete action script on a particular
entity.  Directions must remain realizable under the available
tools, domain policy, and environment constraints.

When learner feedback is available, use performance and coverage
signals to prioritize weak or under-covered regions.  When a
new synthesis factor has been introduced, deliberately allocate
synthesis budget across its value range, particularly toward values
that remain empty or under-covered.

OUTPUT

{
"conditions":  [
{
"condition_id":  "...",
"objective_type":
"undercovered | over_mastered | failure_targeted | exploration |
repair",
"objective":  "...",
"preferred_values":  {"<key>":  ["<value>"]},
"allowed_values":  {"<key>":  ["<value>"]},
"forbidden_values":  {"<key>":  ["<value>"]},
"avoid_patterns":  ["..."],
"budget_level":  "zero | low | medium | high | explore",
"rationale":  "..."
}
]
}

Output strict JSON only.
```

Planner Prompt

```
You are the Planner for interactive tool-use task synthesis. Given
the current synthesis space and an optional generation direction
from the Designer, construct one structured synthesis blueprint.
The blueprint defines a task type rather than a fully instantiated
task. A downstream Generator uses the blueprint together with the
domain policy, tool schemas, and environment state to realize a
concrete executable task.

RUNTIME INPUTS

DOMAIN: airline

DOMAIN CONTEXT SUMMARY:
<environment and task context>

AVAILABLE TOOLS SUMMARY:
<tool schemas and functional descriptions>

DATABASE / ENVIRONMENT SUMMARY:
<available entities and environment state information>

POLICY SUMMARY:
<domain policies and execution constraints>

DIMENSION VALUES:
{
"task_family": [
"flight_booking",
"flight_change",
"flight_cancellation",
"refund_or_compensation",
"seat_or_baggage_service",
"loyalty_or_membership_service"
],
"expected_outcome": ["fulfill", "refuse"],
"policy_or_constraint_focus": [...],
"tool_use_pattern": [...],
"interaction_challenge": [...],
"difficulty_profile": ["easy", "medium", "hard"],
<additional evolved factors>
}

GENERATION CONDITION FROM DESIGNER:
{
"objective": "...",
"preferred_values": {...},
"allowed_values": {...},
"forbidden_values": {...},
"avoid_patterns": [...]
}

INSTRUCTIONS

Select mutually compatible values from the provided synthesis
space:
-- exactly one task_family;
-- exactly one expected_outcome;
-- one or two policy_or_constraint_focus values;
-- one to three tool_use_pattern values;
-- zero to two interaction_challenge values;
```

```
-- exactly one difficulty_profile;
-- one value for each additional evolved factor when specified.

If a Designer condition is provided, obey all hard constraints
and strongly prefer the specified soft preferences.  Complete
the remaining dimensions so that the full selection is mutually
compatible and realizable with the available tools and policy.

The blueprint should encourage genuine multi-step interaction
rather than surface-level variation.  It should specify a task type
rather than fixed entities or a memorized instance.

For policy-compliant tasks, the intended trajectory should verify
the relevant conditions and carry out the required state-changing
action.  For policy-violating tasks, the intended trajectory should
verify the violation and decline the request without an invalid
state change.

OUTPUT

{
"domain":  "airline",
"plan_id":  "...",
"selected_dimensions":  {
"task_family":  "...",
"expected_outcome":  "...",
"policy_or_constraint_focus":  [...],
"tool_use_pattern":  [...],
"interaction_challenge":  [...],
"difficulty_profile":  "...",
<additional evolved factors>
},
"generation_intent":  "...",
"success_focus":  ["..."],
"diversity_notes":  ["..."],
"expected_difficulty_controls":  {
"expected_tool_calls":  "...",
"num_policy_constraints":  "...",
"num_entities":  "...",
"requires_clarification":  true/false,
"allows_user_false_claim":  true/false,
"requires_state_change":  true/false
}
}

Output strict JSON only.
```

Reflector Prompt

```
You are the Reflector for a self-evolving task-synthesis pipeline.
You do not generate tasks or allocate synthesis budget.  Your
role is to revise the synthesis space using post-training learner
feedback.

RUNTIME INPUT: SYNTHESIS AUDIT PACKAGE

CURRENT SYNTHESIS SPACE:
{
"k_1":  ["v_1", "v_2", ...],
"k_2":  [...],
...
```

```
}

COVERAGE AND PERFORMANCE REPORT:
{
"<factor/value>": {
"coverage": "...",
"task_agent_performance": "...",
"before_after_training_change": "..."
},
...
}

DIAGNOSTIC PROBE REPORT:
{
"<probe>": {
"distribution": "...",
"failure_correlation": "...",
"coverage_note": "...",
"suggested_factor": {
"name": "...",
"values": [...]
}
},
...
}

EXPERIMENTAL FACTOR CARDS:
<previously introduced factors and accumulated evidence>

PREVIOUS REFLECTOR OUTPUTS:
<previous promotion / retention decisions>

INSTRUCTIONS

Review the current synthesis space and determine whether the
learner's residual weaknesses are adequately represented by
existing factors. For previously introduced experimental factors,
decide whether each should be promoted, retained for further
evaluation, or deprecated.

A new synthesis factor may be justified in two ways:

  (A) FAILURE-DRIVEN EXPANSION:
A measurable interaction property occurs in the data and is
systematically associated with learner failure. This indicates a
capability gap not adequately represented by the current synthesis
space.

  (B) COVERAGE-GAP-DRIVEN EXPANSION:
A measurable interaction property exposes an important but severely
under-covered region. Empty or nearly empty high-value regions may
indicate that the current synthesis process has not explored the
corresponding interaction structure sufficiently.

When the per-round factor budget forces a choice, prioritize
failure-driven evidence when a strong learner-side signal is
available. Use coverage-gap evidence to expose important
regions that are systematically absent from the current synthetic
experience.

Prefer factors supported by measurable diagnostic evidence
over arbitrary semantic dimensions. A new factor must be
```

```
interpretable, low-cardinality, non-redundant with existing
factors, and realizable by the downstream synthesis pipeline.
When a diagnostic probe provides a generator-realizable suggested
factor, prefer promoting that factor rather than inventing an
unrelated coordinate.

Introduce at most one new synthesis factor in each round.  Do not
rename, merge, delete, or change the semantics of existing seed or
promoted factors.

OUTPUT

{
"round_id":  "...",
"synthesis_space_version_in":  "...",
"dimension_reviews":  [
{
"dimension":  "...",
"decision":  "promote | keep_experimental | deprecate",
"reason":  "...",
"evidence":  "..."
}
],
"new_proposals":  [
{
"action":  "no_change | promote_probe_to_dimension | add_value",
"dimension_name":  "...",
"values":  [...],
"reason":  "...",
"evidence":  "..."
}
],
"summary":  "..."
}

Output strict JSON only.
```

The complete runtime prompts additionally include full machine-readable schemas, round-specific domain policies, tool specifications, environment summaries, and feedback statistics. The representative templates above preserve the actual runtime inputs, constraints, and decision logic while omitting implementation-only formatting and defensive scaffolding. Full executable prompts are included in the anonymous code release.