GRADUS: DISCOVERING AND RECOVERING AGENT LESSONS FROM THE ENVIRONMENT'S LATENT SYLLABUS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training LLM agents for policy-constrained tool environments, such as an airline backend with booking, refund, and cancellation policies, requires tasks that exercise the environment's rules individually and in combination. Existing synthesis pipelines let a model propose tasks and retain only those it immediately solves, biasing training data toward easier behaviors. We introduce GRADUS (**G**raded **R**ungs of **A**scending **D**ifficulty from an **U**nderlying **S**ystem), which decouples what a task should teach from the concrete task and solution used to teach it. Without seed tasks, GRADUS treats a fixed environment as a latent syllabus, extracting guarded behavioral operators that capture how actions depend on current conditions and how their effects reshape subsequent decisions. It composes these operators into increasingly demanding tasks while using corpus-level feedback to target missing outcome combinations. Crucially, GRADUS jointly calibrates tasks and solutions against the environment: execution feedback identifies which side to refine while preserving the intended behavioral requirements. Verification thus becomes a recovery signal rather than a terminal filter, coupling coverage-guided exploration with the preservation of difficult examples. From off-the-shelf backbones, the resulting curriculum lifts a 30B model from 59% to 89% on τ^2-Bench and from 47% to 83% on CAR-Bench with an Opus 4.5 proposer, and to 76% and 62%, respectively, with an open-weight Qwen3-235B proposer. These results show the value of jointly controlling which behaviors are explored and which survive into verified training data.

1 INTRODUCTION

Training LLMs to act as agents requires data spanning diverse tool use, reasoning demands, and multi-turn interactions. Recent synthesis pipelines build generalist agents by generating tasks, workflows, and even entire environments at scale (Lyu et al., 2026; Wang et al., 2025; Song et al., 2026). These efforts broaden the range of situations agents encounter. A complementary problem arises when the environment is already given: how can we construct a curriculum that prepares an agent for the full range of behaviors required by that particular environment?

A fixed environment contains substantially more training structure than its list of APIs suggests. Depending on the current state and operating policies, the same action may be permitted, denied, or require additional steps. For example, in an airline booking environment, whether a cancellation is permitted depends on the cabin class, the time since booking, and whether travel insurance covers the stated reason, so the same request may end up with different agent behaviors in different states. Moreover, an earlier action may change the conditions governing later decisions. Constructing training data from this environment structure presents two coupled challenges: deciding which behaviors to teach and producing valid tasks and solutions that teach them.

A high-coverage curriculum must therefore identify these behavioral distinctions, compose them systematically, and track which remain underrepresented. Existing methods structure synthesis through task blueprints and workflows (Prabhakar et al., 2025), explicit constraints (Chen et al., 2026b), domain-entity graphs (Sun et al., 2026), or predefined behavioral and difficulty dimensions (Gu et al., 2026). These structures effectively guide individual proposals, but individually

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.

diverse or difficult tasks do not necessarily produce broad corpus-level coverage of policy outcomes and their interactions.

A second challenge is preserving the intended coverage during verification. Existing synthesis pipelines commonly use successful execution, constraint satisfaction, or exact state agreement as an acceptance gate, retaining verified trajectories while filtering unsuccessful candidates (Sun et al., 2026; Chen et al., 2026b; Lyu et al., 2026; Zeng et al., 2026). Although this improves data precision, it also makes verification a hidden curriculum selector. As tasks combine more rules and state changes, errors in both task construction and solution generation become more likely, causing difficult behaviors to be disproportionately dropped.

We introduce GRADUS (**G**raded **R**ungs of **A**scending **D**ifficulty from an **U**nderlying **S**ystem), which separates what agent behaviors to teach and producing valid tasks and solutions that teach them. Starting from a fixed executable environment without seed tasks, GRADUS organizes each action according to the conditions governing its use and the consequences of its possible outcomes. These reusable descriptions, which we call *guarded operators*, expose how one action can enable, invalidate, or otherwise alter the requirements of another. GRADUS composes these relationships into increasingly demanding task blueprints, while corpus-level feedback directs generation toward underrepresented outcomes and combinations. Additional cross-space interaction dimensions vary how the same underlying requirements are surfaced by introducing ambiguity, indirect references, social pressure, or reasoning demands.

Crucially, as blueprints become more difficult, some fail to yield a valid task and solution on the first attempt, and neither the proposed task nor its solution is assumed correct. Using the executable environment as a common anchor, GRADUS lets task design and solution construction calibrate each other: failures in solution construction feed back into task revision, flagging unsolvable tasks, hallucinated states or conditions, and ambiguous objectives, while the task and its oracle proposal anchor the solution search, steering it to fix departures from policy. Both sides are refined against environment execution, helping difficult compositions survive verification rather than being discarded.

From off-the-shelf backbones, the resulting curriculum lifts a 30B model from 59% to 89% on τ^2-Bench and from 47% to 83% on CAR-Bench with an Opus 4.5 proposer, and to 76% and 62%, respectively, with an open-weight Qwen3-235B proposer. These results demonstrate the value of jointly controlling which behaviors are explored and which survive into verified training data. Our contributions are:

- A compositional representation of environment behavior that connects action conditions, policy-dependent outcomes, and evolving state.
- Coverage-guided task construction with explicit controls over structural complexity and interaction demands.
- Environment-grounded joint task-solution refinement that recovers difficult examples while preserving their intended behavioral requirements, connecting proposed coverage to verified training data.

2 METHODOLOGY

2.1 THE LATENT SYLLABUS

A fixed environment implicitly specifies what an agent must learn through its tools, states, policies, and executable transitions. In an airline system, for example, the same cancellation request may require a refund or a refusal depending on the booking state and policy; earlier booking changes may alter which options remain available. These conditional behaviors and their interactions form the environment's *latent syllabus*. They are distributed across system artifacts rather than supplied as an explicit collection of training tasks.

GRADUS turns this implicit syllabus into verified training data through

$$E \xrightarrow{\text{Organize}} \mathcal{G} \xrightarrow{\text{Coverage-guided search}} \mathcal{Z} \underset{\text{Diagnose and re-design}}{\overset{\text{Construct and verify}}{\rightleftarrows}} \mathcal{C} \xrightarrow{\text{Accept}} \mathcal{D}. \tag{1}$$

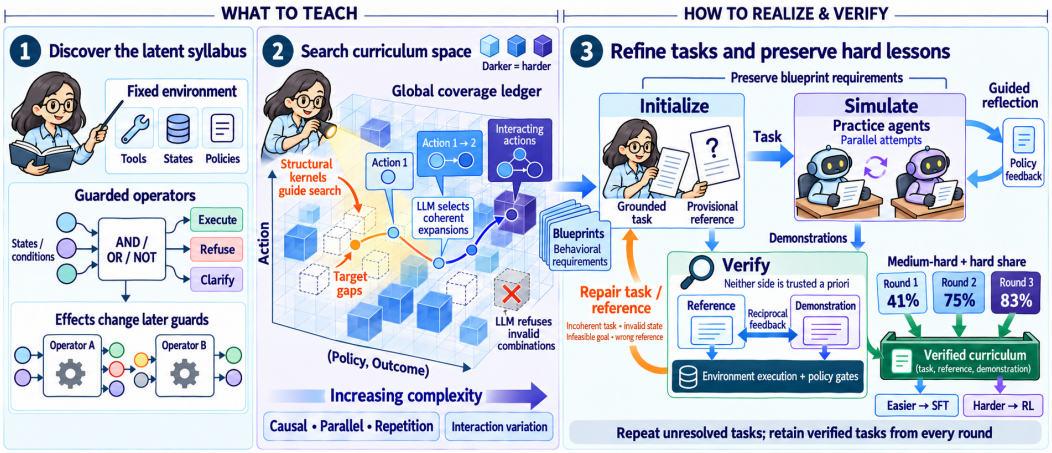


Figure 1: **Overview of GRADUS. (1)** Environment artifacts are compiled into guarded operators. **(2)** Coverage-guided evolution searches uncovered syllabus regions. **(3)** Joint task-solution refinement recovers unresolved candidates, preserving difficult lessons while accumulating verified examples.

where G contains reusable descriptions of environment behaviors, Z contains task blueprints, C contains the realized curriculum, and D is the resulting accepted training corpus. A *guarded operator* in G describes an action together with the conditions governing its possible outcomes, such as when cancellation is allowed and what refund follows. *Coverage-guided search* composes these descriptions into blueprints that specify which behaviors a task should exercise, prioritizing under-represented outcomes and interactions. A blueprint might combine a booking modification with a subsequent cancellation; additional interaction patterns can make the request ambiguous without changing its underlying requirements.

Each blueprint is then instantiated as a concrete task with a provisional reference solution and tested through agent interaction. *Verification* uses execution evidence and policy checks to assess the proposed reference and observed solution, guiding revision of either side when conflicts arise. Throughout refinement, the blueprint preserves what the task is intended to teach, connecting targeted behavioral coverage to verified examples.

2.2 COMPILING ENVIRONMENT ARTIFACTS TO GUARDED OPERATORS

To make the environment's behavioral requirements reusable during synthesis, we ask an LLM to organize the supplied policies, tool implementations, and state schemas into a persistent library. Each entry centers on one atomic action and records its purpose, parameters, relevant state fields, guarded-condition branches and conditional consequences. We call this a *guarded operator*:

$$g(a) = \{r_{a,i} \Rightarrow q_{a,i}\}_{i=1}^{m_a}, \tag{2}$$

where each guard $r_{a,i}$ combines applicable conditions using conjunction, disjunction, or negation, and $q_{a,i}$ describes the corresponding effects or required behavior, including state updates, follow-up actions, refusals, or failures. For example, a cancellation operator distinguishes eligible bookings that trigger refunds from ineligible requests requiring refusal.

These descriptions expose how an action's effects can establish or invalidate another action's conditions. They serve as an LLM-readable representation for composing behavioral requirements.

2.3 COVERAGE-GUIDED SCENARIO EVOLUTION

GRADUS uses the operator library to construct symbolic scenario blueprints. A blueprint specifies the user goal, relevant context, known and unknown information, and intended operator outcomes, while leaving concrete entities and database values unbound. Construction proceeds through the three stages below: at each stage, code governs the evolution schedule and samples the transformation instructions, and the LLM realizes each instruction into a contextually coherent blueprint.

Atomic blueprints. For each guarded operator, the LLM receives the full operator catalog and generates k initial blueprints exercising its conditional behaviors. A shared coverage ledger tracks all the (policy, outcome) pairs of all operators represented in generated blueprints. Generation prompts include this ledger to prioritize underrepresented outcomes, both during atomic scenario generation and subsequent structural evolution.

Structural evolution. For each parent blueprint, code requests up to K children by independently sampling a structural kernel for each attempt. It supplies the LLM with the parent, kernel instruction, operator catalog, and current coverage ledger. The kernel specifies how to extend the parent: CAUSAL connects actions through a state change, PARALLEL adds an independent request, and REPETITION applies an action across a user-described set. The LLM chooses compatible operators and outcomes, retaining the parent's requirements while adding new ones, or declines an incoherent extension. For example, causal evolution may add a booking modification whose effects change the conditions of a subsequent cancellation. Repeating this procedure builds a lineage tree whose branching factor and depth control the breadth and extent of composition.

Interaction variation. After structural evolution, code samples interaction kernels and asks the LLM to rewrite each blueprint accordingly. These kernels introduce indirect references, ambiguity, user pressure, or additional reasoning while preserving the intended actions and outcomes. Thus, structural kernels change what the task requires, whereas interaction kernels change how those requirements will be surfaced and interpreted. New interaction dimensions can be added through additional kernel instructions.

Across all stages, the programmatic controller allocates search across structural kernels, and interaction variations, while the LLM performs model-guided pruning: it realizes an assigned expansion, selects a compatible alternative, or declines when the scenario-kernel combination is unrealistic. This avoids enumerating the full Cartesian product while preventing generation from collapsing onto the model's preferred patterns.

2.4 PRESERVING COVERAGE THROUGH TASK AND SOLUTION VALIDITY

A blueprint specifies what a task should teach, but its task, reference, and demonstration must still be constructed, with harder blueprints more likely to fail on the first attempt. GRADUS therefore jointly refines both: *task validity* concerns whether the grounded task is coherent and feasible, while *solution validity* concerns whether a solution fulfills the task under the environment's policies. The two processes take each other's results as reciprocal feedback: Task design cross-checks simulated solution outcomes to diagnose task design defects and guide its next revision; solution search cross-checks the task and reference to produce reflection feedback for subsequent attempts. Environment execution anchors both directions. The process follows an initialization $\rightarrow$ simulation $\rightarrow$ verification loop, while preserving the blueprint's behavioral requirements.

Initialization: ground blueprints to environment states. An LLM agent receives the blueprint, tools, policies, state schemas, and an environment-probing tool. It inspects the state and constructs a concrete *task* and a candidate *reference* action sequence. Code immediately replays the reference and returns execution errors for local revision. This first check establishes executability; whether the task is coherent and the reference fulfills it remains subject to cross-checking through simulation.

Simulation: task-guided solution refinement. A user simulator receives the task and interacts with a solving agent that sees only the conversation and tool responses, never the hidden task specification or reference, and their interaction produces a full *demonstration* candidate. Code launches K parallel search branches with at most m attempts each. A trajectory evaluator scores each attempt for action correctness, policy compliance, and communication quality, judging against the proposed task and reference, the policies, and the execution evidence, and returns feedback that guides the next attempts. The task and reference thus steer simulation reflection while remaining hidden from the solving agent. By the time an outcome reaches verification (next phase), parallel sampling and feedback-guided retries have already searched for the strongest demonstration the task admits.

Verification: demonstration-guided task refinement. This phase decides which (task, reference, demonstration) tuples to accept and, for the rest, arbitrates the refinement direction for the next

loop: repair a flawed task, or keep searching for a better demonstration. To do so, code replays the reference and the demonstrated actions from the same initial state and compares their outcomes. Acceptance is decided by: when the two outcomes agree and the programmatic policy checks pass, the whole tuple is accepted as gold-standard supervision. When they do not, a diagnostic LLM reads the task, reference, demonstration, outcome differences, and check results and attributes the failure. Demonstration errors route the task back to further solution search, whereas evidence of a flawed task or reference (an incoherent or unsolvable task, hallucinated states, an incorrect reference or outcome) is summarized as feedback and returned to initialization. The language model only interprets the replay evidence and never certifies correctness itself, so a mutually agreeable but environment-inconsistent tuple cannot be accepted.

GRADUS thus treats task design and solution search as two provisional halves refined against each other, with execution in the environment as the shared arbiter that accepts a pair only when both hold. By recovering difficult blueprints rather than discarding them at their first failure, this loop carries the coverage produced by scenario evolution through to the training data, where the hardest lessons would otherwise be lost.

3 Experiments

3.1 Experimental Setup

Environments and evaluation. We evaluate curriculum quality by downstream success on two customer-facing agent benchmarks: τ^2-Bench airline, retail, and telecom domains (Barres et al., 2025) and CAR-Bench (Kirmayr et al., 2026), an in-car voice assistant, with base, disambiguation, and hallucination task splits. Each task is scored by its benchmark's executable evaluator as a pass or fail, and per-split success is reported as pass@1 averaged over five independent runs, $\overline{\text{pass}}@1 = \frac{1}{5}\sum_{i=1}^{5}\mathbb{I}[y_i = 1]$. Variance and consistency metrics are reported in Appendix A.8. Benchmark tasks and references are held out from training curriculum construction, training, and checkpoint selection.

Curriculum construction and training. GRADUS uses either Claude Opus 4.5 (Anthropic, 2025) or Qwen3-235B-A22B-Thinking-2507 (Qwen Team, 2025b) for all model-mediated synthesis roles and trains Qwen3-8B (Qwen Team, 2025d), Qwen3-14B (Qwen Team, 2025a), and Qwen3-30B-A3B students (Qwen Team, 2025c). Shallow lessons (evolution depths 1-3) are routed primarily to SFT, whereas deeper lessons (depths 4-5) are routed primarily to RL. Full-parameter SFT is followed by GRPO-based RL (Shao et al., 2024), whose reward is the same executable acceptance signal used during construction, so RL optimizes verified targets rather than a learned or heuristic score.

Model references. Table 1 reports proprietary frontier models (Opus 4.5 (Anthropic, 2025) and GPT-5.2 (OpenAI, 2025)), open-weight frontier-scale models: Kimi-K2-Thinking (Moonshot AI, 2025) and DeepSeek-V3.2 (DeepSeek-AI, 2025), and off-the-shelf Qwen3 backbones, including Qwen3-235B-A22B (Qwen Team, 2025b) as our open-weight proposal model. We also evaluate released 8B generalist-agent checkpoints to measure how agents broadly trained with generalist data synthesis recipes transfer to these policy-constrained environments (Song et al., 2026; Wang et al., 2025; Sun et al., 2026; Lyu et al., 2026).

Synthesis baselines. We train the same Qwen3-8B student on the released Simia (Li et al., 2025), CoVe (Chen et al., 2026b), and WRIT (Gu et al., 2026) datasets using our full-parameter SFT recipe, matching the number of tasks and maximum tokens per example across the baselines and our methods; these comparisons are limited to airline and retail, where their data are available. We further compare three paradigms under the same Opus 4.5 teacher, Qwen3-8B student, and data budget: *Model-prior synthesis* provides the teacher with the environment policy, tools, and state schema but no explicit task structure or coverage guidance; the teacher freely proposes a blueprint, grounds it into a task and oracle, and retains it only when simulation produces the designed outcome. *Graph-based synthesis* replaces free generation with a set of ontology constructed from tool graphs, each generating multiple blueprints before applying the same grounding and simulation-based rejection procedure. *Trajectory-first synthesis* instead lets the LLM explore the live environment to obtain executable trajectories, uses a global tracker to discourage repeated action sequences, and derives a user scenario and oracle backward from each trajectory, followed by final replay. None of these baselines uses GRADUS's adaptive coverage expansion or verification-guided task recovery.

Table 1: Task success on τ^2-Bench and CAR-Bench, with the six-split average (Avg) at right. GRADUS (ours) rows are shaded; **bold** marks the best result among all rows outside the *Frontier models* block.

		τ^2-Bench			CAR-Bench			
Model	Proposer	Air	Ret	Tel	Base	Disamb	Halluc	Avg
Frontier models								
Opus 4.5	–	0.88	0.89	0.92	0.85	0.68	0.64	0.81
GPT-5.2	–	0.82	0.88	0.89	0.79	0.52	0.88	0.80
Kimi-K2-Thinking	–	0.84	0.88	0.79	0.56	0.42	0.67	0.69
DeepSeek-V3.2	–	0.78	0.91	0.76	0.64	0.48	0.57	0.69
Open-weight backbones								
Qwen3-8B	–	0.38	0.41	0.40	0.40	0.32	0.40	0.39
Qwen3-14B	–	0.42	0.60	0.42	0.49	0.35	0.42	0.45
Qwen3-30B	–	0.63	0.77	0.38	0.52	0.43	0.45	0.53
Qwen3-235B	–	0.61	0.80	0.50	0.64	0.46	0.46	0.58
Generalist agent training (released checkpoints)								
EnvScaler (8B)	–	0.36	0.57	0.33	0.48	0.34	0.34	0.40
Klear-AgentForge (8B)	–	0.39	0.70	0.26	0.46	0.29	0.36	0.41
AgentSkiller (8B)	–	0.34	0.64	0.65	0.49	0.34	0.33	0.47
AgenticQwen (8B)	–	0.44	0.64	0.45	0.55	0.32	0.56	0.49
Environment-targeted synthesis (matched student & budget)								
Simia (8B)	GPT-5/o4-mini	0.36	0.60	–	–	–	–	–
CoVe (8B)	Qwen3-235B	0.31	0.66	–	–	–	–	–
WRIT (8B)	Qwen3.6-Plus	0.58	0.78	–	–	–	–	–
Environment-targeted synthesis (matched proposer, student & budget)								
Model-Prior Synthesis (8B)	Opus-4.5	0.46	0.53	0.57	0.40	0.44	0.31	0.45
Graph-Based Synthesis (8B)	Opus-4.5	0.53	0.59	0.62	0.49	0.36	0.46	0.51
Trajectory-First Synthesis (8B)	Opus-4.5	0.59	0.70	0.71	0.56	0.48	0.52	0.59
GRADUS (ours)								
Qwen3-8B	Qwen3-235B	0.71	0.74	0.64	0.69	0.45	0.51	0.62
Qwen3-8B	Opus-4.5	0.71	0.85	0.92	0.76	0.58	0.74	0.76
Qwen3-30B-A3B	Qwen3-235B	0.69	0.85	0.75	0.72	0.60	0.55	0.69
Qwen3-30B-A3B	Opus-4.5	0.75	0.86	0.90	0.80	0.74	0.74	0.80
Qwen3-30B-A3B +RL	Opus-4.5	**0.80**	**0.89**	**0.97**	**0.83**	**0.75**	**0.91**	**0.86**

3.2 Main Results

Table 1 reports the main results across τ^2-Bench and CAR-Bench.

Curriculum construction matters under matched controls. With the Opus 4.5 proposer, Qwen3-8B student, and data budget fixed, GRADUS outperforms all three controlled synthesis baselines on every split. Compared with the strongest baseline, success increases from 59% to 71% on airline, from 71% to 92% on telecom, and from 52% to 74% on CAR hallucination. The resulting 8B student also surpasses Kimi-K2-Thinking and DeepSeek-V3.2 across all CAR-Bench splits. **GRADUS remains effective with a weaker proposer.** Using Qwen3-235B and fewer than 1000 examples per domain, GRADUS raises the 8B student's average success from 40% to 70% on τ^2-Bench and from 37% to 55% on CAR-Bench. It leads the generalist-agent baselines on four of six splits and outperforms Simia and CoVe on both available domains. The only environment-targeted exception is WRIT on retail (78% versus 74%), where WRIT uses a different synthesis model, Qwen3.6-Plus. **The curriculum transfers across architectures and supports effective RL.** Without resynthesizing the curriculum, transferring from the dense Qwen3-8B student to the Qwen3-30B-A3B mixture-of-experts model maintains or improves performance on five of six splits, with the largest gain on CAR disambiguation (58% to 74%). Executable RL then improves all six splits, most notably increasing CAR hallucination success from 74% to 91%. The final model reaches 80%, 89%, and 97% on the three τ^2-Bench domains, matching Opus 4.5 on retail and achieving the best result on

telecom. On CAR-Bench, its 83% average surpasses that of every frontier model, while also achieving the best overall results on disambiguation and hallucination. Additional comparisons across 8B, 14B, and 30B student models are provided in Appendix Table 13 and Table 14.

3.3 Ablations on Curriculum Components

Table 2 isolates each curriculum component. Removing guarded operators lowers performance on both domains, as the proposer must then infer guards and outcomes from raw tool schemas, policies, and state definitions implicitly. The global ledger matters most for telecom, whose operators contain many policy-outcome pairs: corpus-wide coverage tracking is what steers the agent from skipping a required diagnostic step. Structural evolution matters as well, as it steers the proposer from overproducing familiar scenario structures. Interaction variation matters much more for CAR disambiguation, because varying how requests are surfaced demands different levels of interpretation, reasoning, and clarification. The validity ablations show a complementary effect. Removing verification causes a substantial drop, although the resulting curriculum remains useful because coverage-guided search can still produce difficult tasks whose solutions happen to succeed on the first attempt. Parallel search improves performance by combining outcome-based matching with multiple solution attempts. Adding policy-grounded reflection yields a particularly large gain on CAR disambiguation, where the proposer often fails on its first attempt but improves after receiving targeted feedback. Finally, task recovery uses simulation feedback to redesign failed tasks, preserving difficult yet valid examples and producing a more complete curriculum.

Table 2: Ablation of GRADUS components with Qwen3-235B-A22B as proposal model and Qwen3-30B-A3B as student backbone. SFT task success (mean ± std) on τ^2-Bench telecom and Car-Bench disambiguation. Left: leave-one-out ablation; right: cumulative validity loop.

Remove one component	Tel.	Dis.	Add validity cumulatively	Tel.	Dis.
Full (Ours)	0.75 ±0.02	0.60 ±0.05	No validity	0.65 ±0.05	0.40 ±0.04
− Guarded operators	0.64 ±0.03	0.45 ±0.03	+ Solution: parallel	0.68 ±0.02	0.43 ±0.03
− Global ledger	0.54 ±0.03	0.52 ±0.04	+ Solution: reflection	0.71 ±0.04	0.55 ±0.05
− Structural evolution	0.63 ±0.03	0.55 ±0.05	+ Task recovery (full)	0.75 ±0.02	0.60 ±0.05
− Interaction variation	0.67 ±0.02	0.46 ±0.04			

3.4 Curriculum Coverage

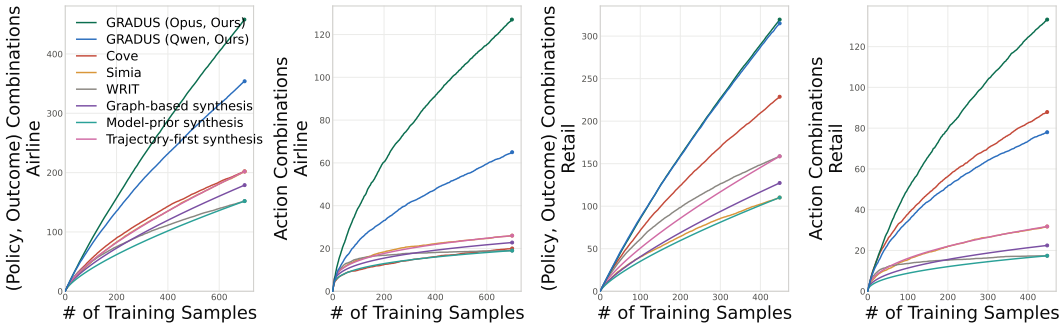


Figure 2: **Dataset Coverage by Environment Dimensions** (1,2: Airline; 3,4: Retail). **Panel-1,3**: Policy-Outcome Coverage. **Panel-2,4**: Action Combinations.

Because GRADUS targets coverage gaps, it explores substantially more of the task space on both airline and retail (Figure 2). Panels (1,3) measure distinct *(policy, outcome) combinations*; coverage-guided search discovers 1.4× to 2.9× more combinations compared to baseline methods, exposing the student to a broader range of interacting policies and outcomes. Panels (2,4) measure *action combinations* among state-changing tools, where structural evolution expands coverage by up to

$\sim 5\times$ on airline and $\sim 1.5\times$ on retail. In all four panels, baseline coverage plateaus as additional examples accumulate, indicating repeating familiar patterns, whereas GRADUS continues discovering new ones. Proposer strength also matters: the two proposers coincide on retail (policy, outcome) coverage but differ by roughly $1.7\times$ on action combinations, indicating that composing unseen action sets depends on proposer capability more than enumerating policy branches does.

3.5 How Curriculum Preservation Retains Difficult Lessons

Figure 3 shows how iterative recovery preserves curriculum difficulty. In the left panel, mean zero-shot success of the off-the-shelf student model falls monotonically from 0.87 to 0.17 with evolution depth, demonstrating controlled difficulty growth. The middle panel traces the full initialization $\rightarrow$ simulation $\rightarrow$ verification loop, with the red arrows marking tasks to be re-initialized in the next round. This process is not a filtering funnel: the final curriculum accumulates the post-verification outputs from every round (all blue bars), rather than retaining only Round-3. The right panel shows that recovery disproportionately retains difficult examples: medium-hard and hard tasks comprise 41% of initially accepted tasks but 75% and 83% of those recovered in Rounds-2 and-3. By recovering rather than discarding difficult proposals, the loop preserves the difficulty introduced by coverage-guided search and supports the downstream results in Table 1.

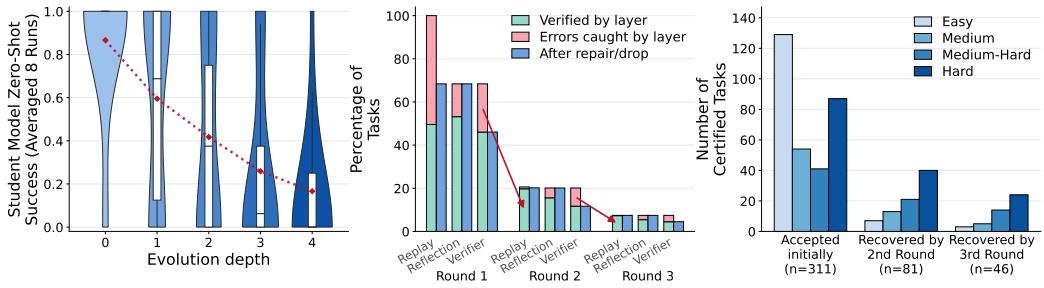


Figure 3: Curriculum preservation retains difficult lessons. **(a)** Student zero-shot success (averaged over 8 runs) versus evolution depth. **(b)** Replay, reflection and recovery dynamics per loop. **(c)** Verified tasks grouped by the round in which they were accepted, split by difficulty class (Easy/Medium/Medium-Hard/Hard).

3.6 From Intermittent Demonstrations to Reliable Learning

As discussed in Section 2.4, solution generation is cast as a search problem; the two search strategies (parallel rollouts and reflection-guided retries) recover capability that a single reference-blind proposal model would miss at its first attempt (Figure 4). The two settings compare both proposal models: proprietary proposer (Opus 4.5 on CAR: Hallucination, top) and open-weight proposer (Qwen3-235B on airline, bottom). The proposer improves from 0.64 and 0.61 at pass@1 to 0.72 and 0.84 at pass@4, and further to 0.88 and 0.88 with evaluator-guided retries. Training then converts this intermittent capability into reliable single-shot behavior: SFT lifts the student from 0.45 to 0.74 and from 0.63 to 0.69, surpassing both proposers' pass@1. RL raises this further to 0.91 on CAR-Bench, exceeding the proposer's observed search success rate of 0.88.

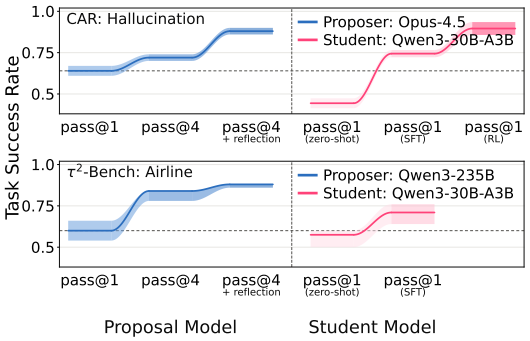


Figure 4: From intermittent frontier-model capability into reliable student performance.

3.7 FROM CURRICULUM TO FRONTIER EXAMS

Beyond synthesizing training data, we investigate whether GRADUS could construct frontier exams for the proposer, since the environment is cast as one latent syllabus for both: coverage-guided search targets uncovered *(policy, outcome)* and action combinations, which initialization, simulation, and verification realize harder tasks. Along similar lines, TASTE (Keren et al., 2026) searches tool-name sequences and falls back to simpler variants when harder evolution fails; GRADUS recovers the evolved task and its solution while preserving the difficulty requirements. With a frontier model as scout, this process therefore locates supported regions that the frontier model solves only intermittently. Using Opus 4.5 as scout, we evolve airline, retail, and CAR-Bench base tasks to depth five and retain verified tasks whose gold outcome is matched in at most five of eight rollouts. Two annotators audited all 58 τ^2-Bench and 79 CAR-Bench candidates, judging 87.5%, 100%, and 90.5% correct, with Gwet's agreement coefficient AC1 (Gwet, 2008) of 0.79, 1.00, and 0.96 (Appendix A.9). On tasks both annotators accepted as correct, Opus-4.5 (proposal model) falls from 88%, 89%, and 85% to 52%, 60%, and 43%, and three other held-out models decline by 18% to 44% in every domain (Table 3), so the difficulty transfers to models held out from the exam construction process.

Table 3: Task success on the original benchmarks and the frozen frontier exams (mean $\pm$ std over five runs). Results are reported on τ^2-Bench (Airline and retail), and CAR-bench (base). Opus 4.5 serves as the scout; all other models are held out during the evolved exam synthesis.

	Airline (τ^2)		Retail (τ^2)		Base (CAR)	
Model	Original	Evolved	Original	Evolved	Original	Evolved
GPT-5.2	0.82 ± 0.05	0.64 ± 0.08	0.88 ± 0.02	0.62 ± 0.08	0.79 ± 0.02	0.44 ± 0.04
Opus 4.5	0.88 ± 0.01	0.52 ± 0.08	0.89 ± 0.01	0.60 ± 0.02	0.85 ± 0.02	0.43 ± 0.03
DeepSeek-V3.2	0.78 ± 0.03	0.43 ± 0.08	0.91 ± 0.02	0.57 ± 0.04	0.64 ± 0.02	0.26 ± 0.04
Kimi-K2-Thinking	0.84 ± 0.03	0.40 ± 0.10	0.88 ± 0.01	0.49 ± 0.16	0.56 ± 0.03	0.17 ± 0.03

4 CONCLUSION

We introduced GRADUS, a framework that treats a fixed executable environment as a latent syllabus and separates the behaviors to be taught from the tasks and solutions that teach them. GRADUS organizes environment logic into guarded operators, uses a global coverage ledger to steer composition toward underrepresented policy outcomes and action combinations, and refines tasks and solutions through reciprocal feedback, so verification recovers difficult lessons rather than filtering them out. Under matched controls, the resulting curricula outperform alternative synthesis methods, transfer across student architectures, and support executable RL: the final 30B-A3B model reaches 89% average success on τ^2-Bench and 83% on CAR-Bench, surpassing all evaluated frontier model on the latter. The same machinery turns advanced syllabus regions into frontier exams that remain challenging for held-out frontier models. Thus, the agent-data synthesis could be framed as controlled curriculum search, requiring control over both which behaviors are explored and which survive verification. Extending this machinery beyond bounded environments with explicit policies and executable outcomes remains important future work.

AI USE STATEMENT

Large language models are components of the proposed data-synthesis method. Language models compile guarded operators, construct and evolve task blueprints, initialize tasks and references, simulate interactions, provide reflection feedback, and diagnose failed verification attempts. Language models are also the students trained and evaluated in our experiments. These model-mediated operations are distinguished from programmatic search, environment execution, replay, and outcome checking throughout the paper, as described in Section 2. Model choices are reported in Section 3.1. During manuscript preparation, generative AI tools also assisted with drafting and language editing, figure preparation, literature discovery, data analysis and software development and debugging. The

authors retain responsibility for the research decisions and the accuracy and originality of the final manuscript and accompanying artifacts.

Ethics Statement

Our experiments use sandboxed benchmark environments, synthetic backend states, and simulated users; they do not execute actions on real customer accounts. The human annotation study is limited to evaluating synthetic tasks, and no annotator identities or personal information are reported. Improving agents in policy-constrained environments may reduce errors, but the same techniques could be used to generate persuasive, adversarial, or social-engineering interactions, or to identify behavioral weaknesses in deployed systems. Real-world applications require domain-specific safety evaluation, access controls, privacy protections, and appropriate human oversight.

Reproducibility Statement

Section 2 and Algorithm 1 describe the curriculum-construction procedure. Section 3.1 specifies the experimental setup and baseline comparisons, while Appendices A.5 and A.3 report curriculum statistics and training configurations. Appendix A.8 provides detailed evaluation results, and Appendix A.9 documents frontier-exam construction and human auditing. Code for reproducing the method and experiments is provided through the anonymous repository: `https://anonymous.4open.science/r/gradus_review-739B/`.

References

Anthropic. Claude Opus 4.5 System Card. Model card, 2025. URL https://www-cdn.anthropic.com/bf10f64990cfda0ba858290be7b8cc6317685f47.pdf. Accessed 2026-09-25.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Shihao Cai, Runnan Fang, Jialong Wu, Baixuan Li, Xinyu Wang, Yong Jiang, Liangcai Su, Liwen Zhang, Wenbiao Yin, Zhen Zhang, Fuli Feng, Pengjun Xie, and Xiaobin Wang. Autoforge: Automated environment synthesis for agentic reinforcement learning, 2025. URL https://arxiv.org/abs/2512.22857.

Aili Chen, Chi Zhang, Junteng Liu, Jiangjie Chen, Chengyu Du, Yunji Li, Ming Zhong, Qin Wang, Zhengmao Zhu, Jiayuan Song, Ke Ji, Junxian He, Pengyu Zhao, and Yanghua Xiao. Dive: Scaling diversity in agentic task synthesis for generalizable tool use, 2026a. URL https://arxiv.org/abs/2603.11076.

Jinpeng Chen, Cheng Gong, Hanbo Li, Ziru Liu, Zichen Tian, Xinyu Fu, Shi Wu, Chenyang Zhang, Wu Zhang, Suiyun Zhang, Dandan Tu, and Rui Liu. Cove: Training interactive tool-use agents via constraint-guided verification, 2026b. URL https://arxiv.org/abs/2603.01940.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Ningyu Zhang, Fei Huang, Wentao Zhang, and Jingren Zhou. Towards general agentic intelligence via environment scaling. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17610–17621, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.872. URL https://aclanthology.org/2026.findings-acl.872/.

Jiaxuan Gao, Jiaao Chen, Chuyi He, Shusheng Xu, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward rl: Post-training multi-turn interactive tool-using agents, 2026. URL https://arxiv.org/abs/2601.22607.

Hengrui Gu, Xiaotian Han, and Kaixiong Zhou. Writ: Write-read intensive trajectory synthesis for multi-turn user-facing agents, 2026. URL https://arxiv.org/abs/2606.02908.

Kilem Li Gwet. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48, 2008. doi: 10.1348/000711006X126600.

Tomer Keren, Nitay Calderon, Asaf Yehudai, Yotam Perlitz, Michal Shmueli-Scheuer, and Roi Reichart. A matter of taste: Improving coverage and difficulty of agent benchmarks, 2026. URL https://arxiv.org/abs/2605.28556.

Johannes Kirmayr, Lukas Stappen, and Elisabeth Andre. CAR-bench: Evaluating the consistency and limit-awareness of LLM agents under real-world uncertainty. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 40599–40618, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1886. URL https://aclanthology.org/2026.acl-long.1886/.

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. Simulating environments with reasoning models for agent training, 2025. URL https://arxiv.org/abs/2511.01824.

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 Yunyao Li, Georg Rehm, and Mei Tu (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 535–551, San Diego, California, USA, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-394-4. doi: 10.18653/v1/2026.acl-industry.37. URL https://aclanthology.org/2026.acl-industry.37/.

Moonshot AI. Kimi-K2-Thinking. Hugging Face model card, 2025. URL https://huggingface.co/moonshotai/Kimi-K2-Thinking.

OpenAI. Update to GPT-5 System Card: GPT-5.2. Model card, 2025. URL https://deploymentsafety.openai.com/gpt-5-2. Accessed 2026-09-25.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay, 2025. URL https://arxiv.org/abs/2504.03601.

Qwen Team. Qwen3-14B. Hugging Face model card, 2025a. URL https://huggingface.co/Qwen/Qwen3-14B.

Qwen Team. Qwen3-235B-A22B-Thinking-2507. Hugging Face model card, 2025b. URL https://huggingface.co/Qwen/Qwen3-235B-A22B-Thinking-2507.

Qwen Team. Qwen3-30B-A3B-Thinking-2507. Hugging Face model card, 2025c. URL https://huggingface.co/Qwen/Qwen3-30B-A3B-Thinking-2507.

Qwen Team. Qwen3-8B. Hugging Face model card, 2025d. URL https://huggingface.co/Qwen/Qwen3-8B.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

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 Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.407. URL https://aclanthology.org/2026.findings-acl.407/.

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, 2026. URL https://arxiv.org/abs/2602.09372.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, Qi Gu, Hui Su, and Xunliang Cai. Scaleenv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training, 2026. URL https://arxiv.org/abs/2602.06820.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, Dayiheng Liu, Que Shen, Junyang Lin, and Tao Yu. Cua-gym: Scaling verifiable training environments and tasks for computer-use agents, 2026a. URL https://arxiv.org/abs/2605.25624.

Qi Wang, Hongzhi Zhang, Jia Fu, Kai Fu, Yahui Liu, Tinghai Zhang, Chenxi Sun, Gangwei Jiang, Jingyi Tang, Xingguang Ji, Yang Yue, Jingyuan Zhang, Fuzheng Zhang, Kun Gai, and Guorui Zhou. Klear-agentforge: Forging agentic intelligence through posttraining scaling, 2025. URL https://arxiv.org/abs/2511.05951.

Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Pei Chen, Ziwei Dong, Jing Huang, Jiri Gesi, Xianfeng Tang, Chen Luo, Qun Liu, Yisi Sang, Hanqing Lu, Manling Li, Jin Lai, and Dakuo Wang. Trajectory2task: Training robust tool-calling agents with synthesized yet verifiable data for complex user intents, 2026b. URL https://arxiv.org/abs/2601.20144.

Siyuan Xu, Shiyang Li, Xin Liu, Tianyi Liu, Yixiao Li, Zhan Shi, Zixuan Zhang, Zilong Wang, Qingyu Yin, Jianshu Chen, Tuo Zhao, and Bing Yin. Controllable and verifiable tool-use data synthesis for agentic reinforcement learning, 2026a. URL https://arxiv.org/abs/2604.09813.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. Toucan: Synthesizing 1.5m tool-agentic data from real-world mcp environments, 2025. URL https://arxiv.org/abs/2510.01179.

Zhihao Xu, Rumei Li, Jiahuan Li, Rongxiang Weng, Jingang Wang, Xunliang Cai, and Xiting Wang. Unlocking implicit experience: Synthesizing tool-use trajectories from text. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9961–9980, San Diego, California, United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.452. URL https://aclanthology.org/2026.acl-long.452/.

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, 2025. URL https://arxiv.org/abs/2511.15718.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Yucheng Zeng, Weipeng Lu, Linyun Liu, Shupeng Li, Zitian Qu, Chenghao Zhu, Shaofei Li, Zhengdong Tan, Mengyue Liu, Haotian Zhao, Zhe Zhou, and Jianmin Wu. Logigen: Logic-driven generation of verifiable agentic tasks, 2026. URL https://arxiv.org/abs/2603.00540.

A Appendix

Limitations

First, our experiments focus on policy-constrained, stateful tool environments, where operating rules and executable transitions provide concrete structure for curriculum construction. Extending this approach to multimodal interfaces and environments with less explicit operating rules is a direction for future work. Further, settings without a decidable success criterion, such as open-web browsing or open-ended research assistance, fall outside the method as formulated. Second, our strongest results depend on a proprietary model whose behavior may change in the future, which limits exact reproducibility; the open-weight proposer results should be read as the reproducible floor. Finally, our experiments cover the curriculum sizes reported in Table 5, three student backbones, and one RL configuration; substantially larger-scale settings remain unexplored.

A.1 Related Work

Scaling data and environments for generalist agents. A growing line of work scales agent training by synthesizing executable environments together with their tasks, rewards, or interaction trajectories (Fang et al., 2026; Cai et al., 2025; Tu et al., 2026; Song et al., 2026; Wang et al., 2026a). Complementary efforts construct broad tool-use corpora spanning many interfaces and domains, including ToolMind (Yang et al., 2025), GEM (Xu et al., 2026b), AgentSkiller (Sun et al., 2026), AgenticQwen (Lyu et al., 2026), Klear-AgentForge (Wang et al., 2025), and TOUCAN (Xu et al., 2025). These methods primarily seek breadth and transfer across many worlds. GRADUS addresses the complementary problem of achieving systematic behavioral depth within a fixed target environment: it seeks systematic depth within that environment by expanding coverage over its guarded operations and their compositions, and by validating every lesson through deterministic replay in the system itself.

Diverse and verifiable task synthesis for interactive agents. Existing methods obtain diversity through seed amplification or blueprint generation (Li et al., 2025; Prabhakar et al., 2025), explicit constraints or predefined dimensions (Chen et al., 2026b; Gu et al., 2026), domain-entity-graph traversal (Sun et al., 2026), iterative workflow expansion (Lyu et al., 2026), or execution-first exploration (Wang et al., 2026b; Chen et al., 2026a; Zeng et al., 2026), and assess solvability by exact state equivalence (Zeng et al., 2026), state-difference and action matching (Sun et al., 2026), generated programmatic checkers (Song et al., 2026), constraint checklists (Chen et al., 2026b), or LLM rubrics (Lyu et al., 2026); others transform tasks under instance-specific or oracle-preserving checks (Gao et al., 2026; Xu et al., 2026a). GRADUS differs in what it synthesizes and how it is controlled. It makes the system's *guarded operations* (policy-conditioned actions and their contrasting outcomes) the reusable unit of synthesis, and expands them within operations, across operations and into cross-space interaction challenges with ascending difficulty, a space broader than fixed dimensions, tool-graph paths, or behavior-tree branches. An adaptive coverage ledger steers generation toward under-covered targets globally, while the task verification is decided by executable environment replay with the model diagnosing rather than judging, and a selected lesson that first fails is recovered through task revision or expanded demonstration search instead of being discarded.

Table 4 compares related methods along the design choices central to GRADUS: the source of candidate lessons, adaptive corpus-level coverage, the treatment of candidates that fail validation, added interaction challenges, and construction in or outside the target system. The comparison concerns curriculum design rather than overall system capability; partial marks denote mechanisms that apply only to a subset of data or operate at a different level, as detailed below.

Table 4: Curriculum-design comparison. ✓ present; ∼ partial; × absent. **Direct** and **Xfer** indicate construction in the target system and transfer from other systems.

			Failed-lesson handling			Interaction challenges			
Method	Source	Adaptive coverage	Drop	Revise	Demo	Human	Reason.	Cross-dom.	Transfer
Simia (Li et al., 2025)	Seed	×	∼	×	×	×	×	×	Xfer
CoVe (Chen et al., 2026b)	System	×	✓	×	×	∼	×	×	Direct
WRIT (Gu et al., 2026)	System	×	✓	×	×	✓	∼	×	Direct
AgentSkiller (Sun et al., 2026)	World	×	✓	∼	×	∼	×	✓	Xfer
EnvScaler (Song et al., 2026)	World	×	∼	×	×	×	×	×	Xfer
Klear-AgentForge (Wang et al., 2025)	Mixed	×	∼	×	∼	×	×	×	Xfer
AgenticQwen (Lyu et al., 2026)	Seed	×	✓	×	×	∼	×	×	Xfer
LOGIGEN (Zeng et al., 2026)	World	×	✓	∼	×	∼	×	×	Xfer
GRADUS (ours)	System	✓	∼	✓	✓	✓	✓	✓	Direct

A.2 Problem Formulation

Agent-environment interaction. We study an agent π_θ that communicates with a user and acts in a stateful tool environment. Given a user policy μ, the interaction with the environment from the agent's perspective is a partially observable Markov decision process

$$\mathcal{M}_\mu(E) = \big(\mathcal{X}, \mathcal{A}_{\text{ag}}, \mathcal{O}_{\text{ag}}, P_\mu, Z_\mu, \mathsf{r}, \rho_0, \gamma\big). \tag{3}$$

The hidden state $x_t \in \mathcal{X}$ includes the backend state, task state, and information private to the user. An agent action $a_t \in \mathcal{A}_{\text{ag}}$ is either a message or a tool call, while its observations include user messages and tool responses. The agent chooses actions from the interaction history h_t according to $\pi_\theta(a_t \mid h_t)$. Its standard objective is

$$J_E(\pi_\theta; \mu) = \mathbb{E}_{\tau \sim (\pi_\theta, \mu, E)} \left[\sum_{t=0}^{H} \gamma^t \mathsf{r}_t \right], \tag{4}$$

where the reward measures successful task completion under the environment's operating policies.

Fixed executable environment. The agent-environment interaction process should be distinguished from the fixed environment specification consumed by GRADUS. We represent this specification as

$$E = \big(\mathcal{S}, \mathcal{A}_{\text{tool}}, \mathcal{P}_{\text{op}}, F\big), \tag{5}$$

where $\mathcal{S}$ is the backend state space described by the state schema, $\mathcal{A}_{\text{tool}}$ is the set of parameterized tool actions, $\mathcal{P}_{\text{op}}$ is the written operating policy, and

$$F : \mathcal{S} \times \mathcal{A}_{\text{tool}} \to \mathcal{S} \times \mathcal{O}_{\text{tool}}$$

is the executable tool semantics that returns a new state and tool response. For example, in the airline domain, $\mathcal{S}$ contains reservation and payment records, $\mathcal{A}_{\text{tool}}$ includes actions such as `cancel_reservation`, and $\mathcal{P}_{\text{op}}$ specifies when that action should be executed, refused, or preceded by additional steps.

The environment specification E is fixed: GRADUS neither generates nor modifies the definitions of states, tools, policies, or implementation. Combined with a task and user policy μ, however, E induces the environment interaction process $\mathcal{M}_\mu(E)$ in which the agent acts.

Budgeted Curriculum Construction Equation 4 assumes a task distribution; the role of GRADUS is to construct one with broad behavioral coverage. Let $\mathcal{U}_E$ contain the coverage units derived from E, including atomic blueprints, structural evolutions, and interaction variations. Each blueprint z covers a subset $\phi(z) \subseteq \mathcal{U}_E$. For a curriculum $\mathcal{D}$, define

$$n_{\mathcal{D}}(u) = \sum_{d \in \mathcal{D}} \mathbb{I}[u \in \phi(z_d)] \tag{6}$$

as the number of examples covering unit u.

We formulate curriculum construction as

$$\begin{aligned} \max_{\mathcal{D}} \quad & F(\mathcal{D}) = \sum_{u \in \mathcal{U}_E} v(n_{\mathcal{D}}(u)) \\ \text{subject to} \quad & C(\mathcal{D}) \le B, \\ & V_E(d) = 1, \qquad \forall d \in \mathcal{D}, \end{aligned} \tag{7}$$

where v is increasing with diminishing returns, $C(\mathcal{D}) \le B$ imposes the data or construction budget, and V_E requires every retained task and solution to pass environment-grounded validation.

GRADUS approximates this objective rather than solving it exactly. Its coverage ledger prioritizes underrepresented units, while joint task-solution refinement helps selected behaviors survive construction and enter the verified curriculum. Coverage serves as a controllable data-design objective; its effect on the trained policy is evaluated empirically.

Supervised fine-tuning. SFT establishes the behavioral prior: it teaches the student the tool-calling format, the turn-taking discipline of the environment, and the policy-compliant action sequences that the validity loop has already certified, giving RL a starting policy whose success rate is high enough for group-relative comparisons to be informative. Given a verified curriculum $\mathcal{D}_{\mathrm{SFT}} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i contains the task and interaction context and y_i is the verified assistant trajectory, we optimize the standard autoregressive negative log-likelihood:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \frac{1}{\sum_t m_{i,t}} \sum_{t=1}^{|y_i|} m_{i,t} \log \pi_\theta(y_{i,t} \mid x_i, y_{i,<t}), \tag{8}$$

where $m_{i,t} \in \{0, 1\}$ masks non-assistant tokens. Thus, loss is computed only over the target assistant responses while the complete preceding interaction remains available as context.

Reinforcement learning with GRPO. Starting from the SFT checkpoint, we apply Group Relative Policy Optimization (GRPO). For each task x, the old policy π_{old} samples a group of G trajectories $\{y_i\}_{i=1}^{G}$ by interacting with the live environment, which receive executable rewards $\{r_i\}_{i=1}^{G}$ from the same environment-grounded checks used during verification. GRPO forms group-normalized advantages $\hat{A}_i = (r_i - \bar{r})/(\mathrm{std}(r) + \delta)$, with δ a small constant for numerical stability, and optimizes

$$\mathcal{J}_{\mathrm{GRPO}}(\theta) = \mathbb{E}_{x, \{y_i\}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \min\Big(\rho_{i,t} \hat{A}_i,\ \mathrm{clip}(\rho_{i,t}, 1 - \epsilon_{\mathrm{low}}, 1 + \epsilon_{\mathrm{high}}) \hat{A}_i \Big) \right], \tag{9}$$

where $\rho_{i,t} = \pi_\theta(y_{i,t} \mid x, y_{i,<t}) / \pi_{\mathrm{old}}(y_{i,t} \mid x, y_{i,<t})$ and the token index t again ranges only over agent-generated tokens.

A.3 TRAINING DETAILS

SFT configuration. We fine-tune all parameters for three epochs with learning rate 1.0×10^{-5} under a cosine schedule with warmup ratio 0.1 and maximum sequence length $32{,}768$, instantiating the mask $m_{i,t}$ so that loss falls on the final assistant turn of each multi-turn example. The checkpoint with the lowest loss on a held-out 5% validation split is selected and, for the RL stage, initializes the policy. This recipe is held fixed throughout: the same optimization settings, sequence length, epoch count, and checkpoint-selection rule are applied to every student size in Table 13 and Table 14 and to every baseline curriculum in Table 1, with no per-method or per-size tuning. Differences in downstream success therefore reflect the training data under comparison rather than differences in how each dataset was consumed.

RL configuration. RL uses GRPO advantage estimation with DAPO-style refinements (Yu et al., 2025): asymmetric clip thresholds $\epsilon_{\mathrm{low}} = 0.2$ and $\epsilon_{\mathrm{high}} = 0.25$, token-level loss aggregation, and KL regularization with coefficient 0.01 under a low-variance estimator. We train with learning rate 1×10^{-6}, batch size 8, and $G = 16$ rollouts per task, under a budget of 32k length per rollout. Rollouts are collected by interacting with the live environment, so every reward reflects an executed outcome.

A.4 ALGORITHM

Algorithm 1 follows the three stages in Equation 1: constructing guarded operators, searching for lesson blueprints, and realizing them as verified training examples. The algorithm marks whether each operation is performed by an LLM or by code. Code controls search and execution, while the LLM constructs the semantic content.

Algorithm 1 GRADUS: coverage-guided curriculum construction from a fixed executable system.

Require: system $E = (\mathcal{S}, \mathcal{A}, \mathcal{P}, T)$; blueprints per operator k; children K; depth D; rollouts R; attempts m; budget B

Ensure: curriculum $\mathcal{D}$ of accepted (task, reference, demonstration) tuples

Stage 1: Construct guarded operators
$\mathcal{G} \leftarrow \{\, \textsc{ConstructOperator}(a, \mathcal{P}, \textsc{Code}(a), \textsc{Schema}(a)) : a \in \mathcal{A} \,\}$ ▷ LLM: branches $r_{a,i} \Rightarrow q_{a,i}$, touched state, failures
Stage 2: Coverage-guided lesson search ($\mathcal{Z} \leftarrow \varnothing$; ledger $L \leftarrow \mathbf{0}$)
for each $g(a) \in \mathcal{G}$ **do**
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{\, k$ atomic blueprints $\textsc{Atomic}(g(a), \mathcal{G}, L) \,\}$; $L \leftarrow \textsc{UpdateLedger}(L, \mathcal{Z})$ ▷ LLM generates; CODE records (a, ω)
end for
for depth $d = 1 \ldots D$ **do** ▷ structural expansion
  for each parent z at depth $d-1$ **do**
    for $c = 1 \ldots K$ **do**
      $z' \leftarrow \textsc{Expand}(z, \kappa, \mathcal{G}, L)$, $\kappa \sim \{\text{causal}, \text{parallel}, \text{repetition}\}$ ▷ CODE samples κ; LLM adds operator or DECLINEs
      if $z' \neq$ DECLINE **then** $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{z'\}$
      end if
    end for
    $L \leftarrow \textsc{UpdateLedger}(L, \mathcal{Z})$ ▷ CODE: update coverage after this parent's children
  end for
end for
for each $z \in \mathcal{Z}$ **do**
  $z \leftarrow \textsc{Decorate}(z, \sigma)$, $\sigma \sim \mathcal{K}_{\text{sem}}$ ▷ CODE samples σ; LLM rewrites to same actions, added surface
end for
Stage 3: Realization with the validity loop ($\mathcal{D} \leftarrow \varnothing$)
for each blueprint $z \in \mathcal{Z}$ **do**
  $b \leftarrow 0$; *fb* $\leftarrow \varnothing$
  repeat
    (*task*, *ref*, s_0) $\leftarrow \textsc{Initialize}(z, E, \textsc{Probe}, \textit{fb})$ ▷ LLM: probe state, ground task and reference
    if $\neg\, \textsc{Replay}(\textit{ref}, s_0)$ **then** *fb* $\leftarrow$ *err*; $b \leftarrow b+1$; **continue**
    end if ▷ CODE: reference pre-check
    $\hat{d} \leftarrow \textsc{Simulate}(\textit{task}, \pi_{\text{blind}}, U)$ ▷ LLM: R rollouts $\times$ m evaluator-guided retries
    match $\leftarrow \textsc{ReplayCompare}(\textit{ref}, \hat{d}, s_0)$; *pol* $\leftarrow \textsc{PolicyCheck}(\hat{d}, \mathcal{P})$ ▷ CODE: cross-examine by execution
    if *match* $\wedge$ *pol* **then**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\textit{task}, \textit{ref}, \hat{d})\}$; **break** ▷ accept gold-standard tuple
    else
      side $\leftarrow \textsc{Diagnose}(\textit{task}, \textit{ref}, \hat{d}, \textit{diff}, \textit{checks})$ ▷ LLM: attribute defect
      fb $\leftarrow$ feedback to INITIALIZE if *side* $=$ TASK, else to SIMULATE
    end if
    $b \leftarrow b+1$
  until accepted **or** $b \geq B$
end for
return $\mathcal{D}$ ▷ route by depth: shallow $\rightarrow$ SFT, deep $\rightarrow$ RL

A.5 Training Data

Data statistics. Table 5 reports the size of the final curriculum per domain and proposer, and Table 6 breaks each total down by evolution depth. SFT rows are per-assistant-step rows expanded from the SFT tasks; RL items are one per task. Shallow lessons (depths 1-3) supply most SFT tasks and deep lessons (depths 4-5) most RL tasks.

	Opus 4.5 proposer			Qwen3-235B proposer		
Domain	SFT tasks	SFT rows	RL tasks	SFT tasks	SFT rows	RL tasks
Airline	698	8,468	341	699	6,848	267
Retail	585	7,971	381	448	5,096	249
Telecom	414	6,145	367	370	5,162	168
CAR (combined)	1,901	11,091	1,227	986	5,117	1,018

Table 5: Final curriculum size per domain and proposer. SFT tasks and RL tasks are verified task–reference pairs; a task supplies either supervised demonstrations or an RL reference.

Depth	Airline	Retail	Telecom	CAR (combined)
Opus 4.5 proposer				
1	156	128	178	718
2	352	278	363	900
3	267	223	192	660
4	156	190	48	458
5	108	147	–	392
Total	1,039	966	781	3,128
Qwen3-235B proposer				
1	466	162	139	605
2	314	266	208	480
3	147	187	141	410
4	39	82	50	509
Total	966	697	538	2,004

Table 6: Verified curriculum tasks by evolution depth. Each entry is the total number of accepted tasks at that depth (supervised and RL tasks combined). The Opus proposer evolves to depth 4 on airline, retail, and CAR; telecom saturates at depth 4. The Qwen proposer is reported over depths 1-4. CAR combines the base, disambiguation, and hallucination splits.

A.6 Ablation Details

All ablations fix the proposer, student, data budget, and training procedure. − **Guarded operators** replaces them with raw policies, tool schemas, and state schemas, leaving the proposer to infer constraints and scenarios. − **Global ledger** retains evolution but removes corpus-level *(policy, outcome)* tracking and gap selection. − **Structural evolution** instead generates scenarios in one shot, without structural kernels or iterative expansion. − **Interaction variation** sends structurally evolved scenarios directly to task realization. **No validity** trains on the first rollout without verification. + **Solution: parallel** samples K rollouts and accepts one whose outcome matches the reference. + **Solution: reflection** additionally gives each rollout up to two evaluator-guided retries based on policy compliance, outcome agreement, and communication quality. + **Task recovery (full)** : task- or reference-side errors trigger re-initialization, while solution-side errors trigger further demonstration search, until verification succeeds or the recovery budget is exhausted.

A.7 End-to-End Example: Evolving and Recovering a Policy-Interaction Lesson

This section traces one airline example through guarded-operator extraction, structural evolution, task initialization, reflection-guided simulation, and task-side recovery. The artifacts below are

lightly abridged for readability; all policy conditions, task requirements, and state-changing actions are preserved.

Latent lesson from guarded operators. The extracted operators reveal a causal interaction that is not apparent from the tool names alone. An old economy reservation without insurance cannot be cancelled for a change of plans. However, changing its cabin to business makes the business-cabin cancellation guard true.

Guarded-operator artifact (abridged raw specification)

Source citations and low-level state paths are omitted.

`action_spec: cancel_reservation`

- `description:` Cancel the whole reservation.
- `parameters:` `reservation_id: string, required`.
- `returns:` the updated reservation.
- `precondition: AND`
  - obtain the user ID, reservation ID, and cancellation reason;
  - list the cancellation and refund details and obtain explicit confirmation;
  - verify that the reservation exists;
  - verify that no flight has status `flying` or `landed`;
  - verify that at least one cancellation trigger holds:

$$\begin{aligned} &\texttt{created_within_24h} \\ \vee\ &\texttt{airline_cancelled_flight} \\ \vee\ &\texttt{cabin=business} \\ \vee\ &(\texttt{insurance=yes} \wedge \texttt{covered_reason}). \end{aligned}$$

  - if a certificate was used, disclose that its remaining value is not refundable.
- `effects:`
  - append a negative refund entry for every original payment;
  - set `reservation.status` ← `cancelled`;
  - leave flight-level seat availability unchanged.
- `postcondition:`
  - the reservation is cancelled;
  - refunds mirror the original payments and produce zero net payment;
  - refunds settle within 5–7 business days.
- `ambiguity:` the API does not enforce the flown-portion, cancellation-trigger, reason, or confirmation requirements; the agent must enforce these policy guards.

`action_spec: update_reservation_flights` (relevant fields only)

- a cabin change is permitted if no segment has already been flown;
- the cabin must be uniform across all segments, sufficient seats must exist, and the user pays any fare increase;
- on success, `reservation.cabin` ← `requested cabin`.

Given the guarded operators, we can directly identify policy interactions between flight modification and cancellation. For example, cancellation is blocked when an economy reservation is uninsured, was booked more than 24 hours ago, and is being cancelled for a change of plans. Upgrading the reservation to business class activates a separate cancellation trigger, making the same reservation eligible for cancellation:

$$\underbrace{\text{economy} \wedge \text{uninsured} \wedge \text{old booking}}_{\text{cancellation blocked}} \xrightarrow{\text{upgrade cabin}} \underbrace{\text{business}}_{\text{trigger enabled}}$$

$$\underbrace{\text{business}}_{\text{trigger enabled}} \xrightarrow{\text{cancel}} \text{status} \leftarrow \text{cancelled}.$$

Coverage-guided structural evolution. The following trace shows how each evolution step preserves the existing requirements while adding one new structural demand.

Depth 0: Atomic cancellation lesson

The user asks to cancel an uninsured economy reservation booked more than 24 hours ago because their plans changed. None of the four cancellation triggers holds, so the agent must refuse the cancellation.

⇓

Depth 1: Add a logical chain

Retained: the original cancellation request is initially blocked.
Added: The user asks any workarounds to get the flight canceled. The agent must discover an intermediate action that changes the governing policy condition. It first suggests to upgrade the reservation to business class, thereby activating the business-cabin cancellation trigger, and then completes the original cancellation:

$$\texttt{upgrade_to_business} \longrightarrow \texttt{cancel_reservation}.$$

⇓

Depth 2: Add a parallel sub-task

Retained: the complete upgrade-cancel chain from Depth 1.
Added: In addition to cancel the flight, the user separately asks to update baggage on another reservation. This request is independent of the cancellation, requiring the agent to manage two distinct entities and action types without mixing their state:

$$\big(\texttt{upgrade}(A) \rightarrow \texttt{cancel}(A)\big) \quad + \quad \texttt{update_baggage}(B).$$

⇓

Depth 3: Extend the causal chain

Retained: the upgrade–cancel workflow and the independent baggage update.
Added: After cancelling the original trip and updating baggages on another trip, the user requests replacement travel. The agent must search the environment and determine that the requested booking cannot be completed because the target flight lacks sufficient seats:

$$\texttt{update_baggage}(\text{B}) \quad + \quad \texttt{upgrade}(\text{A}) \rightarrow \texttt{cancel}(A) \rightarrow \texttt{replacement denied}.$$

⇓

Depth 4: Expand cancellation into an exception-set loop

Retained: the cabin-upgrade dependency, baggage update, and failed replacement booking.
Added: the single cancellation is generalized to "cancel all my reservations except B." The agent must enumerate the user's reservations, preserve the named exception, and apply the cancellation policy separately to every remaining reservation. Different iterations produce different outcomes: some reservations can be cancelled directly, one requires the upgrade–cancel chain, and another cannot be cancelled because a segment has already been flown.

⇓

Final step: Add interaction variation

Retained: all structural actions, dependencies, exceptions, and outcomes from Depth 4.
Added: the same requirements are surfaced less directly. The user buries an invalid compensation request among valid operations, accepts the cabin upgrade only if its cost is below a threshold, describes the baggage quantity relative to an unknown free allowance, and proceeds only

when the resulting fee is at most $100. The final task therefore requires policy resistance, state-dependent parameter resolution, conditional action, and boundary reasoning without changing its underlying structural lesson.

Task initialization. Initialization grounds the evolved blueprint in executable entities, prices, availability, and user state. The first realization selected a gold member with five reservations and instantiated the following task.

Round-0 initialized task

User request. "Cancel all my reservations except `6DHUJY`." For reservation `55VQNU`, upgrade to business and cancel only if the upgrade costs less than $300. After the cancellations, request compensation for the inconvenience and push back twice if it is refused. Then add two bags beyond the free allowance to `6DHUJY`, but only if the fee is at most $100. Finally, request a one-way economy replacement flight from Denver to Chicago.

Grounded state.

- membership: gold;
- `55VQNU`: economy, uninsured, upgrade cost $100;
- `U8XRWI`: business cabin and directly cancellable;
- `9FNHDR`: intended to contain a previously flown segment;
- `6DHUJY`: business cabin with four free bags; six total bags incur exactly $100;
- intended replacement outcome: no acceptable economy seats.

Intended write outcomes.

```
upgrade(55VQNU) → cancel(55VQNU),
cancel(U8XRWI),
refuse_cancel(9FNHDR; flown portion),
update_baggage(6DHUJY; 6 total, $100),
refuse_replacement_booking(insufficient seats).
```

This realization preserves the evolved lesson: cancellation is initially blocked, a cabin update changes the governing policy branch, and the agent must also maintain an exception set and several independent constraints.

Reflection-guided solution search. The first rollout did not immediately solve the task. Rather than accepting it as a demonstration or discarding the lesson, the evaluator returned policy-grounded feedback for another simulation attempt.

Simulation and reflection trace

Initial rollout. The first trajectory did not satisfy the evaluator, so its feedback was returned to the proposal model rather than accepting the demonstration.

Reflection retry 1: incomplete policy verification. The revised agent still denied cancellation of reservation `9FNHDR` by inferring that an earlier flight date meant the segment had been flown, without checking its recorded status.
Evaluator feedback: "Before denying cancellation for a flown portion, call `get_flight_status` and verify that the segment is `flying` or `landed`; do not infer its status from the date."

Reflection retry 2: corrected behavior. The agent incorporated the additional feedback and checked the flight status. It discovered that the segment was marked `cancelled`, activating the airline-cancellation trigger rather than the flown-portion prohibition. The agent then correctly completed the cancellation workflow, including the upgrade–cancel chain, named exception, compensation refusal, and baggage-price boundary.

Result. The evaluator assigned the revised trajectory a score of 1.0, but identified the reference as incorrect: the initialized environment did not contain the flown segment assumed by the proposed solution.

The two retries illustrate that reflection need not repair every error at once: each evaluation narrows the remaining failure. Once the agent-side behavior is corrected, the residual mismatch supplies evidence for routing the example back to task initialization.

Verification and task-side routing. Execution verification compared the corrected rollout with the proposed reference state and found that the disagreement was caused by initialization, not by the agent.

Round-0 verification result

Route: `initialization`

Diagnosis 1. The task intended `9FNHDR` to contain a flown segment, but flight `HAT277` was initialized as `cancelled`, not `landed` or `flying`. The agent therefore correctly treated the airline-cancelled flight as a valid cancellation trigger.

Diagnosis 2. The reference assumed that replacement travel could not be booked. However, the request did not rule out basic-economy or connecting alternatives, and the executable environment contained an alternative the user could legitimately accept.

Recovery instruction. Re-initialize the task with a genuinely flown segment and make the replacement flight constraint explicit, or update the reference to include the available alternative.

GRADUS returns it to initialization while keeping its structural lineage and interaction requirements fixed.

Recovery across initialization rounds. Table 7 summarizes how simulation feedback is converted into task redesign.

Round	Verification finding	Recovery
0	The supposed flown segment was actually airline-cancelled, and the replacement request admitted valid alternatives.	Re-ground the same blueprint with a user having a truly landed segment, sold-out direct economy inventory, and reference actions consistent with the executable state.
1	The flown-segment defect was repaired, but the user could still accept a connecting economy itinerary because only basic economy had been excluded.	Clarify that the replacement request is for a *direct* economy flight and that connecting and basic-economy alternatives must be declined.
2	The predicted and reference database hashes match: `agent_db_match=true`.	Accept the recovered task into the final task set.

Table 7: Task-side recovery preserves the evolved lesson while repairing grounding and ambiguity defects.

This trace illustrates the coupling between search and recovery. Structural evolution discovers a difficult policy interaction and expands it into a multi-constraint lesson; reflection repairs the demonstration; and verification uses residual disagreement to repair the task itself. The final example therefore retains the upgrade–cancel dependency, exception loop, policy resistance, and boundary reasoning that a one-shot filtering pipeline would have discarded.

Final recovered task

The recovered task retains the original difficulty:

- an uninsured economy reservation is upgraded to business because the $273 cost is below the user's $300 threshold, then cancelled;
- two existing business reservations are cancelled directly;
- a reservation with a genuinely landed segment must not be cancelled;
- one named reservation is excluded from the cancellation loop;
- the agent refuses compensation for user-initiated cancellations despite two misleading appeals;

- the kept reservation receives six bags: four free and two paid, placing the $100 fee exactly at the acceptance boundary; and
- the direct replacement booking is denied because no direct economy seats remain, and the user explicitly declines connecting or basic-economy alternatives.

The final execution reaches exactly the proposed state (`predicted_hash = gold_hash`).

A.8 Experiment Results (Detailed)

Tables 8 and 9 supplement the main results with five-run standard deviations and pass^3, which credits a task only when all three independent runs succeed (Barres et al., 2025; Kirmayr et al., 2026). This metric penalizes intermittent behavior. The overall ranking is preserved: on τ^2, RL decreases from 0.89 mean success to 0.78 pass^3, turning its near-parity in mean success with Opus 4.5 into a four-point reliability gap. On CAR, however, its frontier lead remains (0.71 versus 0.64). The remaining instability is concentrated on CAR disambiguation across frontier models and on airline for GRADUS.

Table 8: Detailed task success on τ^2-Bench. Cells report mean $\pm$ standard deviation over five runs. Avg averages the three domains. pass^k credits a task only when all k independent runs succeed;we report $k = 3$. Proposer names the curriculum-construction model where applicable. GRADUS rows are shaded; **bold** marks the best value outside the *Frontier models* block.

Model	Proposer	Airline	Retail	Telecom	Avg	Avg. pass^3
Frontier models						
Opus 4.5	–	0.88 ± 0.01	0.89 ± 0.01	0.92 ± 0.01	0.90	0.82
GPT-5.2	–	0.82 ± 0.05	0.88 ± 0.02	0.89 ± 0.01	0.86	0.76
Kimi-K2-Thinking	–	0.84 ± 0.03	0.88 ± 0.01	0.79 ± 0.03	0.84	0.73
DeepSeek-V3.2	–	0.78 ± 0.03	0.91 ± 0.02	0.76 ± 0.04	0.82	0.66
Open-weight backbones						
Qwen3-8B	–	0.38 ± 0.04	0.41 ± 0.02	0.40 ± 0.04	0.40	0.22
Qwen3-14B	–	0.42 ± 0.04	0.60 ± 0.03	0.42 ± 0.02	0.48	0.31
Qwen3-30B	–	0.63 ± 0.04	0.77 ± 0.01	0.38 ± 0.02	0.59	0.42
Qwen3-235B	–	0.61 ± 0.03	0.80 ± 0.02	0.50 ± 0.03	0.64	0.44
Generalist agent training (released checkpoints)						
EnvScaler (8B)	–	0.36 ± 0.04	0.57 ± 0.03	0.33 ± 0.02	0.42	0.23
Klear-AgentForge (8B)	–	0.39 ± 0.06	0.70 ± 0.04	0.26 ± 0.01	0.45	0.27
AgentSkiller (8B)	–	0.34 ± 0.06	0.64 ± 0.03	0.65 ± 0.02	0.54	0.30
AgenticQwen (8B)	–	0.44 ± 0.02	0.64 ± 0.05	0.45 ± 0.01	0.51	0.31
Environment-targeted synthesis (matched student & budget)						
Simia (8B)	GPT-5/o4-mini	0.36 ± 0.04	0.60 ± 0.03	–	–	–
CoVe (8B)	Qwen3-235B	0.31 ± 0.02	0.66 ± 0.04	–	–	–
WRIT (8B)	Qwen3.6-Plus	0.58 ± 0.03	0.78 ± 0.03	–	–	–
Environment-targeted synthesis (matched proposer, student & budget)						
Model-Prior Synthesis (8B)	Opus-4.5	0.46 ± 0.04	0.53 ± 0.03	0.57 ± 0.02	0.52	0.33
Graph-Based Synthesis (8B)	Opus-4.5	0.53 ± 0.03	0.59 ± 0.03	0.62 ± 0.03	0.58	0.41
Trajectory-First Synthesis (8B)	Opus-4.5	0.59 ± 0.03	0.70 ± 0.03	0.71 ± 0.02	0.67	0.43
GRADUS (ours)						
Qwen3-8B	Qwen3-235B	0.71 ± 0.02	0.74 ± 0.02	0.64 ± 0.01	0.70	0.51
Qwen3-8B	Opus-4.5	0.71 ± 0.05	0.85 ± 0.02	0.92 ± 0.01	0.83	0.68
Qwen3-30B-A3B	Qwen3-235B	0.69 ± 0.05	0.85 ± 0.02	0.75 ± 0.02	0.76	0.61
Qwen3-30B-A3B	Opus-4.5	0.75 ± 0.05	0.86 ± 0.03	0.90 ± 0.02	0.84	0.71
Qwen3-30B-A3B +RL	Opus-4.5	$\mathbf{0.80} \pm 0.03$	$\mathbf{0.89} \pm 0.02$	$\mathbf{0.97} \pm 0.01$	**0.89**	**0.78**

Table 9: Detailed task success on CAR-Bench. Cells report mean ± standard deviation over five runs. Avg averages the three splits. pass^k credits a task only when all k independent runs succeed; we report $k = 3$. External curriculum baselines (Simia, CoVe, WRIT) do not report CAR-Bench and are omitted. Proposer names the curriculum-construction model where applicable. GRADUS rows are shaded; **bold** marks the best value outside the *Frontier models* block.

Model	Proposer	Base	Disamb	Halluc	Avg	Avg. pass^3
Frontier models						
Opus 4.5	–	0.85 ±0.02	0.68 ±0.06	0.64 ±0.03	0.72	0.61
GPT-5.2	–	0.79 ±0.02	0.52 ±0.08	0.88 ±0.01	0.73	0.64
Kimi-K2-Thinking	–	0.56 ±0.03	0.42 ±0.05	0.67 ±0.03	0.55	0.43
DeepSeek-V3.2	–	0.64 ±0.02	0.48 ±0.07	0.57 ±0.03	0.56	0.41
Open-weight backbones						
Qwen3-8B	–	0.40 ±0.04	0.32 ±0.05	0.40 ±0.04	0.37	0.22
Qwen3-14B	–	0.49 ±0.05	0.35 ±0.06	0.42 ±0.06	0.42	0.29
Qwen3-30B	–	0.52 ±0.06	0.43 ±0.02	0.45 ±0.04	0.47	0.30
Qwen3-235B	–	0.64 ±0.05	0.46 ±0.06	0.46 ±0.05	0.52	0.39
Generalist agent training (released checkpoints)						
EnvScaler (8B)	–	0.48 ±0.05	0.34 ±0.05	0.34 ±0.02	0.39	0.27
Klear-AgentForge (8B)	–	0.46 ±0.03	0.29 ±0.06	0.36 ±0.01	0.37	0.24
AgentSkiller (8B)	–	0.49 ±0.03	0.34 ±0.07	0.33 ±0.05	0.39	0.25
AgenticQwen (8B)	–	0.55 ±0.05	0.32 ±0.04	0.56 ±0.03	0.48	0.32
Environment-targeted synthesis (matched proposer, student & budget)						
Model-Prior Synthesis (8B)	Opus-4.5	0.40 ±0.03	0.44 ±0.05	0.31 ±0.04	0.38	0.24
Graph-Based Synthesis (8B)	Opus-4.5	0.49 ±0.04	0.36 ±0.04	0.46 ±0.03	0.44	0.31
Trajectory-First Synthesis (8B)	Opus-4.5	0.56 ±0.03	0.48 ±0.04	0.52 ±0.03	0.52	0.38
GRADUS (ours)						
Qwen3-8B	Qwen3-235B	0.69 ±0.04	0.45 ±0.04	0.51 ±0.03	0.55	0.42
Qwen3-8B	Opus-4.5	0.76 ±0.02	0.58 ±0.06	0.74 ±0.04	0.69	0.55
Qwen3-30B-A3B	Qwen3-235B	0.72 ±0.04	0.60 ±0.05	0.55 ±0.04	0.62	0.51
Qwen3-30B-A3B	Opus-4.5	0.80 ±0.04	0.74 ±0.02	0.74 ±0.02	0.76	0.64
Qwen3-30B-A3B +RL	Opus-4.5	**0.83** ±0.03	**0.75** ±0.05	**0.91** ±0.02	**0.83**	**0.71**

A.9 Evolved Frontier Exams: Construction and Human Audit

Frontier Exam Construction. Starting from the original benchmark tasks, Opus 4.5 performs five depths of structural and interaction evolution. Each blueprint passes through the same initialization, simulation, and verification loop used for curriculum construction. Among candidates judged valid, we form the set of exam candidates whose reference outcome is matched in at most five of the eight rollouts. Table 10 summarizes the workflow.

Domain	Evolved	Executable	Valid	$\leq 5/8$	Both correct
Airline (τ^2)	212	84	82	32	25
Retail (τ^2)	399	225	222	26	26
CAR-Bench	512	472	324	79	70

Table 10: Construction and human-audit counts.

Annotation protocol. Two annotators independently assessed all 137 frozen candidates: 32 airline, 26 retail, and 79 CAR tasks. Before annotating each domain tasks, annotators read its policy and tool schema to understand the domain regulations and expected agent behaviors. For every task, they were given the task, initial states, and oracle action sequence (reference) and answered the following questions without conferring.

Q1: Correctness. Is the oracle outcome what a competent, policy-compliant assistant should produce?
A. Yes
B. No
C. Cannot tell

Q2: Realism. Would a real user plausibly make this request in this situation and in this way?
A. Natural
B. Plausible but unusual
C. Contrived

Q3: Oracle strictness. Could a policy-compliant assistant legitimately use different steps, parameters, or ordering while reaching the same outcome?
A. No
B. Partly
C. Yes, materially over-specified

Annotation results. Table 11 reports pooled label distributions, averaging the two annotators. Correctness is high across all domains: 87.5% for airline, 100% for retail, and 90.5% for CAR. Both annotators marked 25/32, 26/26, and 70/79 tasks correct, respectively; only these jointly accepted tasks are used for the reported frontier-exam results. CAR requests are predominantly natural and have non-strict oracles. The evolved τ^2 tasks are more often judged unusual or partly strict, reflecting greater subjectivity about scenario naturalness and interchangeable execution paths.

Domain	Correctness (YES/NO/CT)	Realism (NAT./UNUSUAL/CONTR.)	Strictness (NO/PARTLY/YES)
Airline	87.5 / 10.9 / 1.6	26.6 / 53.1 / 20.3	18.8 / 70.3 / 10.9
Retail	100.0 / 0.0 / 0.0	46.2 / 32.7 / 21.2	53.8 / 38.5 / 7.7
CAR	90.5 / 8.2 / 1.3	88.6 / 11.4 / 0.0	81.0 / 13.3 / 5.7

Table 11: Pooled annotation distributions (%).

Inter-annotator agreement. Raw agreement and Gwet's AC1 support the same conclusions (Table 12). Correctness agreement is substantial for airline (81.2%, AC1 $= 0.789$) and near-perfect for retail and CAR (96.2–100%, AC1 $= 0.958$-1.000). Agreement on oracle strictness is also strong (AC1 $= 0.728$, 0.821, and 0.986), indicating that annotators generally agree on whether an oracle is materially over-specified. Realism is more subjective for airline and retail (AC1 $= 0.399$ and 0.596), but near-perfect for CAR (AC1 $= 1.000$); the wider confidence intervals on τ^2 reflect this uncertainty and the smaller audit sets. For realism and strictness, AC1 is computed over binary acceptable-versus-problematic judgments, while Table 11 retains the full label distributions (Gwet, 2008). Overall, the audit provides strong evidence for semantic correctness and non-strictness, with exception of disagreement concentrated in subjective judgments of naturalness on Airline and Retail domain.

Domain	Dimension	Raw agreement	Gwet's AC1 [95% CI]
Airline	Correctness	81.2%	0.789 [0.586, 0.933]
	Realism	59.4%	0.399 [0.056, 0.680]
	Oracle strictness	78.1%	0.728 [0.461, 0.897]
Retail	Correctness	100.0%	1.000 [1.000, 1.000]
	Realism	73.1%	0.596 [0.239, 0.860]
	Oracle strictness	84.6%	0.821 [0.584, 0.960]
CAR	Correctness	96.2%	0.958 [0.900, 1.000]
	Realism	100.0%	1.000 [1.000, 1.000]
	Oracle strictness	98.7%	0.986 [0.954, 1.000]

Table 12: Inter-annotator agreement.

A.10 CURRICULUM TRANSFER ACROSS STUDENT SIZES

Tables 13 and 14 compare dense 8B and 14B students with the 30B-A3B mixture-of-experts student, rather than conventional model-size scaling. The same curriculum transfers across all three architectures, with the MoE model showing its clearest advantage on CAR disambiguation, while gains on τ^2-Bench are non-monotonic. Curricula proposed by Opus 4.5 generally outperform those from Qwen3-235B across students. Finally, outcome reward-based RL improves the MoE checkpoint on every split, with the largest gain on CAR hallucination.

Table 13: Student architecture and capacity comparison on τ^2-Bench.

Model	Stage	Proposer	Airline	Retail	Telecom
Qwen3-8B (Ours)	SFT	Qwen3-235B	0.71 ± 0.02	0.74 ± 0.02	0.64 ± 0.01
Qwen3-14B (Ours)	SFT	Qwen3-235B	0.69 ± 0.04	0.85 ± 0.02	0.69 ± 0.03
Qwen3-30B-A3B (Ours)	SFT	Qwen3-235B	0.69 ± 0.05	0.85 ± 0.02	0.75 ± 0.02
Qwen3-8B (Ours)	SFT	Opus 4.5	0.71 ± 0.05	0.85 ± 0.02	0.92 ± 0.01
Qwen3-14B (Ours)	SFT	Opus 4.5	0.75 ± 0.01	0.85 ± 0.01	0.85 ± 0.03
Qwen3-30B-A3B (Ours)	SFT	Opus 4.5	0.75 ± 0.05	0.86 ± 0.03	0.90 ± 0.02
Qwen3-30B-A3B (Ours)	SFT+RL	Opus 4.5	0.80 ± 0.03	0.89 ± 0.02	0.97 ± 0.01

Table 14: Student architecture and capacity comparison on Car-Bench.

Model	Stage	Proposer	Base	Disamb.	Halluc.
Qwen3-8B (Ours)	SFT	Qwen3-235B	0.69 ± 0.04	0.45 ± 0.04	0.51 ± 0.03
Qwen3-14B (Ours)	SFT	Qwen3-235B	0.68 ± 0.03	0.50 ± 0.06	0.55 ± 0.04
Qwen3-30B-A3B (Ours)	SFT	Qwen3-235B	0.72 ± 0.04	0.60 ± 0.05	0.55 ± 0.04
Qwen3-8B (Ours)	SFT	Opus 4.5	0.76 ± 0.02	0.58 ± 0.06	0.74 ± 0.04
Qwen3-14B (Ours)	SFT	Opus 4.5	0.80 ± 0.02	0.60 ± 0.03	0.74 ± 0.02
Qwen3-30B-A3B (Ours)	SFT	Opus 4.5	0.80 ± 0.04	0.74 ± 0.02	0.74 ± 0.02
Qwen3-30B-A3B (Ours)	SFT+RL	Opus 4.5	0.83 ± 0.03	0.75 ± 0.05	0.91 ± 0.02