REGE: STRONGER AGENTS THROUGH EXPERIENCE SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Repeated attempts at interactive tasks uncover facts, useful operations, and failures that can inform subsequent execution. We introduce ReGE, a framework for turning this recorded experience into guidance for a fresh attempt. Its main implementation, ReGE-Syn, uses a single coach call to jointly synthesize bounded execution records and available artifacts into self-contained guidance, connecting task-specific findings with proposed actions and relevant pitfalls. A student starts from the task's initial environment state and adapts this guidance to current observations. Source generation, synthesis, and student execution use the same base model, without parameter updates or a stronger teacher. Across four benchmarks, reporting the higher observed score of joint-synthesis or selected-trajectory guidance for each model–benchmark pair yields mean task-success gains of 9.9 percentage points on GPT and 8.8 on Claude over unguided execution. On matched BrowseComp-Plus subsets, ReGE-Syn reduces guidance-construction tokens by 98.2% on GPT and 95.5% on Claude relative to adapted AggAgent, with similar observed success. Combined construction and execution tokens decrease by 15.7% and 35.4%, respectively, counting input plus output and excluding source generation. BrowseComp-Plus ablations further associate access to process information with fewer student steps and show that joint synthesis produces substantially shorter guidance than direct history or independent summaries, with similar observed success. These results demonstrate how agents can use their own recorded experience to improve subsequent task completion.

1 INTRODUCTION

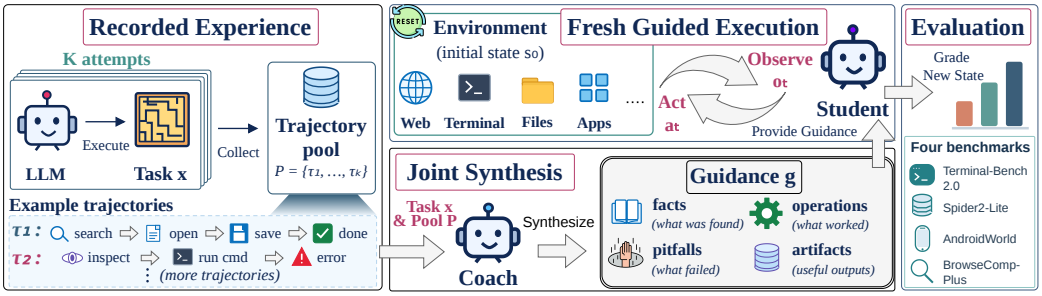

Figure 1: **The joint-synthesis implementation of ReGE.** The coach reads the rendered source pool; the student receives the resulting guidance and interacts with its own environment. Source generation, synthesis, and execution share the same base model.

Repeated attempts can uncover solutions and information that a single run misses (Brown et al., 2024; Snell et al., 2024). For interactive agents, these attempts produce more than candidate answers: they record discovered facts, useful tool operations, intermediate results, and failed strategies. This experience offers material for improving the same model's subsequent actions. We ask: **How can an agent turn experience from its own attempts into more effective subsequent execution?**

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.

Useful discoveries can be distributed across attempts, while the complete records are long and the correctness of individual findings is unknown. Candidate selection, response aggregation, and trajectory summarization provide different ways to reuse prior attempts (Zheng et al., 2023; Li et al., 2025; Kim et al., 2026). The challenge is to retain the details needed for subsequent actions while reducing the history a new agent must process. A file created or an application state reached in a previous attempt does not carry over when the environment is reset: the agent must perform the required operations against the current state.

We introduce **ReGE** (*Re-execution Guided by Experience*), a framework for constructing guidance from recorded attempts. Its main implementation, REGE-SYN, presents bounded execution records and available artifacts together to a coach, which produces self-contained guidance in one call. The coach connects task-specific findings with proposed operations and relevant pitfalls. A fresh student starts from the task's initial environment state, using current observations to follow, adjust, and verify the guidance. Source generation, synthesis, and student execution use the same base model, without parameter updates or a stronger teacher. The framework also includes REGE-SEL, which forwards one judge-selected trajectory as guidance (§4.5).

We evaluate ReGE on Terminal-Bench 2.0, Spider2-Lite, AndroidWorld, and BrowseComp-Plus with GPT and Claude model lines. Reporting the higher observed Syn/Sel score for each model and benchmark yields mean task-success gains of 9.9 and 8.8 percentage points over unguided execution, respectively. On matched BrowseComp-Plus subsets, fixed REGE-SYN reduces guidance-construction tokens by 98.2% on GPT and 95.5% on Claude relative to adapted AGGAGENT, with similar observed success. Although student execution uses more tokens, combined construction and execution usage falls by 15.7% and 35.4%, counting input plus output and excluding source generation. Further BrowseComp-Plus ablations examine which source information helps execution, how to organize multiple attempts, and how the guidance's output form affects its use.

Our contributions are threefold:

- **Joint synthesis of execution experience.** REGE-SYN connects concrete findings, operations, and failure lessons across recorded attempts into actionable guidance for a fresh execution by the same base model.
- **Capability gains with low guidance-construction cost.** Across four benchmarks, ReGE improves mean task completion over unguided execution. On matched BrowseComp-Plus subsets, fixed REGE-SYN substantially reduces construction tokens relative to adapted iterative aggregation and retains lower combined usage after student execution.

2 REGE: EXPERIENCE-GUIDED AGENT EXECUTION

REGE is a framework for constructing guidance from recorded experience and using it in a fresh execution. Guidance can be built by jointly synthesizing multiple attempts (REGE-SYN) or selecting one trajectory (REGE-SEL). We focus on joint synthesis; §4.5 compares the variants. The student starts from the task's initial environment state and uses the guidance alongside current observations. Source generation, guidance construction, and student execution share the same base model, without parameter updates or a stronger teacher. Figure 1 illustrates the synthesis implementation.

2.1 RECORDED EXPERIENCE

A task consists of an instruction x, an initial environment state s_0, and a benchmark evaluator. Repeated attempts produce a source pool $\mathcal{P} = \{\tau_1, \ldots, \tau_K\}$. For REGE-SYN, we construct the text representation $V(\mathcal{P})$ programmatically: each attempt forms a separate block, with available reasoning or action narratives and commands in step order, followed by final artifact paths and contents. These fields are copied and truncated to bound the input length; no per-trajectory LLM summary is generated before the coach sees them.

Benchmark correctness labels are withheld from both coach and student. Raw per-step observation fields are omitted, so execution feedback is accessible only where preserved in the rendered text or artifacts. This representation provides bounded execution records rather than the complete environment history. Appendices A.1 and A.2 specify the field mappings, truncation limits, and prompts.

2.2 JOINT EXPERIENCE SYNTHESIS

Useful experience can be distributed across attempts: one may uncover a necessary task fact, another a workable command, and another a failure to avoid. REGE-SYN brings these records into a shared context and prompts a coach to turn them into self-contained guidance for a fresh attempt. Using the same base model with parameters θ, the coach C_θ constructs

$$g = C_\theta(x, V(\mathcal{P}); p_{\text{syn}}), \tag{1}$$

where p_{syn} specifies the guidance construction task.

All retained source blocks are presented together, preserving attempt boundaries and step order. A single coach call produces the guide, allowing decisions about what to retain and connect to account for information across attempts. Independent per-trajectory summarization makes these decisions separately before combining summaries. The coach is prompted to connect concrete findings with proposed operations and relevant pitfalls, retaining details needed to act on them.

The default guidance is an execution-ordered plan. Exact paths, values, commands, format requirements, and encountered problems are placed alongside the steps where they matter. When the records support a candidate answer or artifact, the coach is asked to include it together with a way to verify it. The guide must be understandable without consulting the source attempts, since the student receives only the resulting guidance. Source correctness labels are unavailable, so its recommendations remain subject to the student's checks against the current environment.

REGE-SYN shares joint aggregation with response-aggregation methods such as Self-MoA (Li et al., 2025). Here, the coach synthesizes bounded execution records and available artifacts into guidance for further interaction: the student must carry out and verify the proposed operations in a fresh environment. Table 1 compares these design choices; §4.2 examines the value of process information, and §4.4 compares joint synthesis with alternative ways of presenting the recorded experience.

Table 1: **Comparison of aggregation methods.**

Criterion	SC (source)	Self-MoA	REFLEXION	Agentic PDR	AGGAGENT	**REGE (Syn)**
Single-pass Synthesis	×	✓	×	×	×	✓
Trajectory Info.	×	×	✓	✓	✓	✓
Student Rerun	×	×	✓	✓	×	✓
Inference Cost	Zero	Low*	High	High	High	**Low**

Prior-work columns describe source voting (Wang et al., 2023), standard Self-MoA (Li et al., 2025), Reflexion (Shinn et al., 2023), agentic PDR (Kim et al., 2026), and AggAgent (Lee et al., 2026).

2.3 GUIDED STUDENT EXECUTION

The student policy π_θ receives the task instruction and guidance, and begins a new interaction from s_0:

$$\tau^g \sim \text{Run}(\pi_\theta, x, s_0; g). \tag{2}$$

Files created and environment changes made in the source attempts do not automatically carry over. The student uses its current observations and interaction history to adopt, adjust, or skip the proposed operations, and performs any required actions in this environment. At termination, the benchmark grades the state or submitted output produced by τ^g.

Section 4.6 illustrates how a student verifies and adapts concrete guidance in a fresh database-recovery execution.

3 EXPERIMENTAL SETUP

Benchmarks. We evaluate on four benchmarks. **Terminal-Bench 2.0** (The Terminal-Bench Team, 2025) includes 89 tasks, with a budget of 30 agent turns per execution. **Spider2-Lite** (Lei et al., 2025) includes 135 instances with local SQLite databases; agents receive read-only access and up

to 18 SQL calls. **AndroidWorld** (Rawles et al., 2025) uses the text-only T3A agent, with guidance supplied through its native task-guideline interface and a step budget of ten times task complexity. After excluding two templates that crash the emulator, we evaluate 114 templates with three parameterisations on GPT and one on Claude, yielding 342 and 114 instances, respectively. **BrowseComp-Plus** (Chen et al., 2025) includes 830 questions over a fixed corpus of 100,000 documents. Agents use retrieval and document-opening tools without live web access, with a 40-step budget. We use DeepSeek-V4-Flash with reasoning disabled as the LLM equivalence judge for BrowseComp-Plus.

Models and budgets. We use `gpt-5.6-luna` and `claude-sonnet-5`, holding the base model fixed across source generation, guidance construction, and student execution. Source pools contain six attempts on GPT Terminal-Bench 2.0, eight on GPT Spider2-Lite, and four elsewhere (pool coverage: Appendix B.1). REGE and AGGAGENT share these pools and each receive one fresh student execution under the same benchmark-specific step limit to match the budget. Generation settings, effective serving limits, and collection exceptions are documented in Appendix A.3.

Baselines. CONTROL performs one unguided execution (pass@1) under the same student step limit. REFLEXION (Shinn et al., 2023) allows up to three executions, using reflections from preceding trials and label-free self-evaluation for stopping. AGGAGENT (Lee et al., 2026) iteratively inspects source records using its released tools and prompts. To align it with our fresh-execution protocol, we adapt its final response into guidance for a fresh student. Our agentic PDR adaptation (Kim et al., 2026) independently summarizes four source trajectories and concatenates the summaries to guide a fresh student (Appendix A.2). We also report two references without student re-execution: SELF-CONSISTENCY (Wang et al., 2023) votes over available source outputs, while BEST-OF-N SELECTION reports the original score of the source selected by the judge model.

Inference cost. Our primary cost metric is the sum of input and output tokens used after source-pool generation, including all method-specific guidance construction and execution calls. Input-only usage and costs that include source generation are explicitly identified.

4 RESULTS

Table 2: **Task success and estimated source-inclusive token costs across four benchmarks.** Acc. denotes task success (%); Cost combines estimated source-generation tokens with measured construction and execution tokens, relative to CONTROL (×). — denotes unavailable or inapplicable entries.

	CONTROL		SC		BEST-OF-N		REFLEXION		AGGAGENT		REGE (Ours)	
Benchmark	Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost
gpt-5.6-luna												
Terminal-Bench 2.0	41.6	1.00	—	—	54.0	4.17	46.1	2.71	46.1	5.83	**56.2**	5.25
Spider2-Lite	59.3	1.00	64.4	4.00	63.7	4.32	64.4	1.50	55.6	11.71	**66.7**	5.36
AndroidWorld	53.8	1.00	—	—	57.9	4.02	55.3	2.30	51.5	4.92	**58.8**	5.06
BrowseComp-Plus	64.1	1.00	74.0	4.00	74.0	4.01	73.3	2.09	76.1	4.74	**76.6**	4.52
Average	54.7	1.00	—	—	62.4	4.13	59.8	2.15	57.3	6.80	**64.6**	5.05
claude-sonnet-5												
Terminal-Bench 2.0	40.4	1.00	—	—	38.2	4.06	48.3	2.62	46.1	6.67	**50.6**	5.26
Spider2-Lite	57.8	1.00	60.0	4.00	61.5	4.16	59.3	1.48	60.7	8.75	**63.7**	5.04
AndroidWorld	61.4	1.00	—	—	64.9	4.01	61.4	2.07	65.8	5.09	**67.5**	5.02
BrowseComp-Plus	74.3	1.00	84.6	4.00	84.7	4.02	84.5	2.14	86.3	4.72	**87.5**	4.39
Average	58.5	1.00	—	—	62.3	4.06	63.4	2.08	64.7	6.31	**67.3**	4.93

Recorded experience strengthens execution by the same model. Across the four benchmarks, REGE improves task completion over CONTROL while keeping the base model fixed and requiring a fresh student execution. For example, GPT success on Terminal-Bench 2.0 rises from 41.6% to 56.2%. Here, the student must use prior experience to perform the required operations in a reset

environment, so the gain reflects improved completion in a new attempt. Joint synthesis provides a low-cost way to construct such guidance: on matched settings, fixed REGE-SYN substantially reduces construction tokens relative to AGGAGENT while retaining lower combined construction and student-execution usage, excluding source generation.

4.1 EFFICIENT EXPERIENCE SYNTHESIS

Table 3: **Joint synthesis reduces guidance-construction tokens by over 95% on both models.** Results on matched BrowseComp-Plus queries. Token counts are mean input plus output per query, including failures and excluding source generation. Construction savings are relative to AGGAGENT. See Appendix A.3 for accounting details.

Method	Success	Construction		Execution	Total
	(%)	Tokens (k)	Saved (%)	Tokens (k)	Tokens (k)
gpt-5.6-luna					
AGGAGENT	75.3	232.9	—	44.8	277.7
REGE-SYN	75.2	**4.2**	**98.2**	229.8	**234.1**
claude-sonnet-5					
AGGAGENT	87.0	211.7	—	134.8	346.6
REGE-SYN	86.2	**9.6**	**95.5**	214.2	**223.8**

Joint synthesis makes effective guidance inexpensive to construct. On matched BrowseComp-Plus queries, fixed REGE-SYN reduces construction tokens by more than 95% relative to AGGAGENT on both base models, while maintaining similar observed task success (Table 3). Guidance construction consequently accounts for only a small fraction of REGE-SYN's combined usage. These results show that consolidating prior attempts into useful guidance need not make construction the dominant inference expense.

The construction savings translate into lower combined usage. Students guided by REGE-SYN consume more tokens than those guided by AGGAGENT, partially offsetting the construction savings. However, the reduction in construction cost more than compensates for this additional student work. On these matched subsets, REGE-SYN uses the fewest combined construction and execution tokens among CONTROL, REFLEXION, AGGAGENT, and REGE-SYN. For REFLEXION, construction includes self-evaluation and reflection, while execution sums token usage across all actor trials. This comparison counts input plus output tokens and excludes source generation.

Experience guidance can also reduce downstream execution effort. Relative to unguided CONTROL, Claude REGE-SYN uses fewer student output tokens and steps on Terminal-Bench 2.0 and Spider2-Lite (Table 4). On Spider2-Lite, SQL calls fall by 50% and output tokens by 30%, whereas both measures remain nearly unchanged on AndroidWorld. These results extend the efficiency analysis beyond BrowseComp-Plus and show that reductions in student effort vary across tasks.

Table 4: **ReGE-Syn reduces student steps and output tokens on Terminal-Bench 2.0 and Spider2-Lite.** Ratios compare Claude REGE-SYN with unguided CONTROL. Only student output tokens and steps are counted, excluding input tokens, source generation, judging, and construction.

Benchmark	Student output tokens	Student steps
claude-sonnet-5 REGE-SYN / CONTROL		
Terminal-Bench 2.0	0.86×	0.88×
Spider2-Lite	0.70×	0.50×
AndroidWorld	1.02×	1.01×

4.2 WHAT INFORMATION MAKES EXPERIENCE USEFUL?

Recorded experience makes coaching more effective. On the same 300 claude-sonnet-5 BrowseComp-Plus queries, coaching without source experience performs similarly to CONTROL, while providing even one recorded attempt substantially improves success in Table 5. Increasing

the source count from one to four adds another 4.0 percentage points, with a paired-test result of $p = .0075$; the smaller difference between two and four remains uncertain. Source coverage and input length increase together, so this comparison does not establish gains at matched inference cost. Appendix B.1 provides the paired comparisons.

Table 5: **Task success increases as the coach receives more source attempts.** CONTROL has no coach; CLOSED-BOOK GUIDANCE uses the coach with zero source attempts. Coached configurations share the same coach, synthesis request, delivery header, and requested output budget; $K' = 1, 2, 4$ expose prefixes of the same ordered source pools.

Configuration	CONTROL No coach	CLOSED-BOOK $K' = 0$	REGE-SYN $K' = 1$	REGE-SYN $K' = 2$	REGE-SYN $K' = 4$
Task success (%)	74.7	75.7	84.3	86.3	**88.3**

Process records help students complete tasks with less downstream work. Table 6 compares final-artifact-only synthesis with full-input REGE-SYN. Nine of the ten net additional successes arise among the 19 queries with only one submitted source answer. The execution traces also reveal a completion bottleneck: among the 17 queries solved only with full input, 16 FAO runs exhaust the 40-step budget without submitting an answer. Even when all four sources submit answers, success counts remain nearly identical while mean student steps fall from 8.9 to 4.8. Thus, the additional information supports both task completion and more efficient execution, including when final answers are already available. Because FAO removes process records and document locators together, their individual contributions are not isolated.

Table 6: **Adding process records yields higher task success with fewer student steps.** FAO uses final artifacts only; REGE-SYN additionally receives process records. Both conditions share the source pools, coach, plan request, and student protocol. The source view and its description differ, and runs span collection windows. Mean steps include failures. Source-answer groups count submitted answers irrespective of correctness and are descriptive.

Metric / source-answer group	**FAO**	**REGE-SYN**
Overall performance		
Task success (%) ↑	85.0	**88.3**
Mean student steps ↓	14.6	**9.1**
Tasks solved, grouped by submitted source answers		
0 source answers ($n = 30$)	3	2
1 source answer ($n = 19$)	9	**18**
2 source answers ($n = 25$)	21	24
3 source answers ($n = 26$)	25	25
4 source answers ($n = 200$)	197	196

4.3 WHEN DOES RECORDED EXPERIENCE HELP?

ReGE-Syn shows its largest gains on pools containing both successful and failed attempts. Table 7 compares fixed REGE-SYN with CONTROL within source-quality groups. Mixed pools show the largest observed success-rate gains in all eight model–benchmark combinations. Across Terminal-Bench 2.0, Spider2-Lite, and AndroidWorld, these gains range from 13.0 to 21.6 percentage points. On BrowseComp-Plus, they reach 48.0 points on GPT and 35.2 on Claude. These results show that source experience need not be uniformly successful to support more effective subsequent execution.

Table 7: **ReGE-Syn achieves its largest observed success-rate gains over Control on mixed source pools.** Tasks are grouped by whether none, some, or all source attempts succeed. Entries report fresh student success rates in percent; n is the number of tasks in each group. Source correctness labels are used only for analysis and are unavailable to the coach and student.

		gpt-5.6-luna			claude-sonnet-5		
Benchmark	**Source-pool condition**	n	CONTROL	REGE-SYN	n	CONTROL	REGE-SYN
Terminal-Bench 2.0	All failed	32	0.0	9.4	47	12.8	14.9
	Mixed	37	51.4	73.0	23	52.2	65.2
	All succeeded	20	90.0	100.0	19	94.7	100.0
Spider2-Lite	All failed	33	0.0	0.0	36	2.8	0.0
	Mixed	44	52.3	72.7	35	48.6	62.9
	All succeeded	58	98.3	100.0	64	93.8	100.0
AndroidWorld	All failed	123	9.8	10.6	35	8.6	8.6
	Mixed	92	57.6	75.0	29	62.1	79.3
	All succeeded	127	93.7	91.3	50	98.0	98.0
BrowseComp-Plus	All failed	198	5.1	7.6	116	6.0	12.1
	Mixed	171	43.9	91.8	196	59.2	94.4
	All succeeded	461	97.0	99.6	517	95.6	99.8

Groups retain the common-valid task intersection of Control, AggAgent, Sel, and Syn, including 829 Claude BCP tasks.

Source quality also reveals the limits of experience reuse. Fresh successes remain possible when all source attempts fail, but improvements over CONTROL are smaller and inconsistent. On GPT Terminal-Bench 2.0, success rises from 0/32 to 3/32, whereas both methods remain at 0/33 on GPT Spider2-Lite. When all sources succeed, CONTROL already solves at least 90% of tasks in every model–benchmark combination, leaving less room for improvement. These retrospective groups may differ in task difficulty; they do not isolate the contribution of failed attempts or establish an advantage over selecting a successful source. Appendix B.2 examines re-execution conditioned on a successful selected source, and Appendix B.3 reports repeated-execution results.

4.4 JOINT SYNTHESIS FOR COMPACT GUIDANCE

Joint synthesis produces shorter guidance with similar observed success. On the 290 matched Claude BrowseComp-Plus queries in Table 8, REGE-SYN produces guidance roughly one quarter as long as direct history or independent summaries. Combined construction and student-execution tokens decrease by 19.8% and 31.8%, respectively, excluding source generation. Paired tests do not establish a success-rate difference.

Table 8: **Joint synthesis reduces guidance length and total token usage while achieving similar observed success.** Total tokens sum construction and student input plus output, including cache reads/writes and construction retries, and exclude source generation.

Method	**Success** (%) ↑	**Guidance length** (characters) ↓	**Total tokens** ↓
Direct history	**88.3**	20,571	268.9k
Independent summaries	86.9	19,590	316.5k
REGE-SYN	**88.3**	**5,063**	**215.8k**

All methods reuse the same four-source pools for a fresh student. Direct history concatenates rendered trajectories without a coach; our agentic PDR adaptation (Kim et al., 2026) concatenates independently generated source summaries; REGE-SYN constructs one guide jointly. Independent

summaries share source content, delivery header, and student protocol with REGE-SYN. Direct history uses a different rendering and header, so its comparison evaluates complete reuse strategies. Requested output limits are 2,500 tokens per independent summary and 10,000 for joint synthesis; these do not imply equal realised usage. Appendix A.2 gives prompts and retries.

4.5 GUIDANCE VARIANTS

Joint synthesis supports effective guidance beyond a fixed execution plan. REGE-A requests an explanation and answer while retaining REGE-SYN's coach, source pool, and delivery header. On the 290 matched Claude BrowseComp-Plus queries in Table 9, this change reduces combined construction and student-execution tokens by 32.1%, while observed success rises from 88.3% to 90.7%. Most of the savings come from student execution, with additional construction savings mainly from shorter coach output.

Table 9: **Explanation-and-answer guidance lowers token usage with higher observed task success.** Total is construction plus student execution, excluding source pool generation.

Variant	Guidance form	Success (%) ↑	Guidance construction ↓	Student execution ↓	Total ↓
claude-sonnet-5 BrowseComp-Plus ($n = 290$)					
REGE-SEL	Selected trajectory	89.3	0.86×	0.79×	0.80×
REGE-A	Explanation + answer	**90.7**	**0.85×**	**0.67×**	**0.68×**
REGE-SYN	Execution plan	88.3	1.00×	1.00×	1.00×

Selecting one trajectory also provides effective guidance: REGE-SEL achieves success close to REGE-SYN with lower student token usage. The answer format does not lead in every tested setting: in the separate 100-query GPT comparison, REGE-A solves 80 queries versus 82 for REGE-SYN.

4.6 FROM RECORDED EXPERIENCE TO FRESH EXECUTION

Concrete guidance supports verification and adaptation during execution. The Claude Terminal-Bench 2.0 `db-wal-recovery` case illustrates how a prior finding and a failure lesson guide operations against the student's current environment.

Execution Case: Recovering a Database from Its Log

Recorded experience. Successful source attempts identify the XOR decoding key `0x42` for the database's write-ahead log. Another attempt opens `main.db` before making a backup and loses `main.db-wal`, leaving the recovery incomplete.

Synthesized guidance. The guide retains the decoding key and instructs the student to back up both files before opening the database, decode and check the log, and then recover the records. The case illustrates guidance that preserves concrete information and a student that checks and adjusts the proposed operations.

Student verification and adjustment. The student backs up its current files, verifies the key and decoded log, and moves the `main.db-shm` cleanup ahead of replacing the log and opening the database. The guide had suggested this cleanup only as a fallback. The student exports 11 ordered records to `recovered.json` and passes all seven official tests.

Incomplete attempts can still supply useful operational knowledge. In Claude AndroidWorld's `BrowserMultiply`, none of the four source attempts completes the task, but two reach a working route for opening the local web page. Here the sources supply an access procedure rather than a completed answer; the student must still obtain the page values and finish the interaction.

Execution Case: Completing a Browser Task from Unsuccessful Attempts

Recorded experience. Repeated taps on `task.html` leave the file manager unchanged. Two attempts discover the route long-press → More options → Open with → Chrome, but all four exhaust their 22-step budgets before completing the multiplication task.

Synthesized guidance. The guide retains this opening sequence, warns against repeated single taps, and describes Chrome's initial setup prompts. It asks the student to obtain the five numbers from the page and submit their product.

Student verification and adjustment. The student uses the opening sequence immediately. When dismissing Chrome's sync prompt returns it to the file manager, it repeats Open with → Chrome to reach the page. It records the displayed numbers $9, 10, 9, 5, 5$ during the five clicks and enters their product, $20{,}250$. The page displays "Success!" and the benchmark records success after 19 steps, within the same 22-step budget as the sources.

5 Related work

Aggregation of multiple attempts. Repeated sampling produces candidates that can be selected through voting (Wang et al., 2023) or evaluation (Cobbe et al., 2021), or combined through generative aggregation (Jiang et al., 2023; Wang et al., 2024b). Self-MoA jointly aggregates responses sampled from a single model (Li et al., 2025). AggAgent extends aggregation to long-horizon agentic tasks by using tools to inspect candidate solutions and retrieve information from their trajectories (Lee et al., 2026). ReGE-Syn shares the goal of combining useful information across attempts. It jointly reads bounded execution records and available artifacts in one coach call, producing guidance that a student uses during a fresh interaction with the environment.

Distillation for subsequent execution. Parallel-Distill-Refine constructs a workspace for subsequent generations using operators that include joint global summarization and candidate selection (Madaan et al., 2025). Kim et al. (2026) adapt PDR to agentic coding by independently summarizing rollouts and concatenating selected summaries to condition new rollouts in freshly initialized environments. Our independent-summary baseline adapts this reuse mechanism, without reproducing the complete refinement-and-selection pipeline. ReGE-Syn constructs guidance jointly from the retained source records, allowing content choices to account for information across attempts before producing the guide.

Reflection and agent memory. Reflexion uses verbal reflections on task feedback to guide subsequent trials (Shinn et al., 2023). ExpeL extracts insights from accumulated experience (Zhao et al., 2024), while Agent Workflow Memory induces reusable workflows (Wang et al., 2024c). AutoGuide derives context-aware guidelines from offline experience (Fu et al., 2024), and AutoManual organizes knowledge acquired through interaction into instruction manuals (Chen et al., 2024). Voyager stores executable skills for later reuse (Wang et al., 2024a); A-MEM organizes experience into linked notes and revises existing memories as new information arrives (Xu et al., 2025). ReGE draws on this broader perspective of improving behavior through textual experience. We study task-specific reuse, asking how records from multiple attempts at the same problem can support a new execution with the same base model.

6 Limitations and Conclusion

Due to computational budget constraints, our evaluation is limited to four benchmarks: Terminal-Bench 2.0, Spider2-Lite, AndroidWorld, and BrowseComp-Plus, with ablation studies conducted on selected subsets of the evaluation data. Further evaluation should cover broader task settings and repeated executions with aligned settings.

We presented ReGE, a framework that turns recorded attempts into guidance for fresh agent execution. Its main implementation jointly synthesizes concrete findings, operations, and pitfalls in a single coach call. Across four benchmarks, experience-guided execution improves mean task completion while keeping the base model fixed. On matched BrowseComp-Plus subsets, joint synthesis sharply reduces guidance-construction tokens relative to adapted iterative aggregation and lowers combined construction and execution usage, excluding source generation. BrowseComp-Plus analyses further support compact experience reuse through multiple guidance forms. These findings establish a practical way to make an agent's prior attempts useful for subsequent action.

AI USE STATEMENT

We used generative AI tools to assist with language polishing and textual revisions of the manuscript, including improvements to clarity, grammar, and presentation. The authors take responsibility for the final content of the paper.

ETHICS STATEMENT

The benchmarks used are public research benchmarks with no human subjects; AndroidWorld runs in an emulator and BrowseComp-Plus uses a fixed offline corpus, so no real user accounts are accessed. The methods reuse an agent's own attempts and do not require additional data collection. We see no direct potential for harm beyond that of stronger autonomous agents in general.

REFERENCES

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/0142921fad7ef9192bd87229cdafa9d4-Abstract-Conference.html.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025.

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

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. *arXiv preprint arXiv:2403.08978*, 2024. URL https://arxiv.org/abs/2403.08978.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-Blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*, 2023.

Joongwon Kim, Wannan Yang, Kelvin Niu, Hongming Zhang, Yun Zhu, Eryk Helenowski, Ruan Silva, Zhengxing Chen, Srinivasan Iyer, Manzil Zaheer, Daniel Fried, Hannaneh Hajishirzi, Sanjeev Arora, Gabriel Synnaeve, Ruslan Salakhutdinov, and Anirudh Goyal. Scaling test-time compute for agentic coding. *arXiv preprint arXiv:2604.16529*, 2026.

Yoonsang Lee, Howard Yen, Xi Ye, and Danqi Chen. Agentic aggregation for parallel scaling of long-horizon agentic tasks. *arXiv preprint arXiv:2604.11753*, 2026.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In *International Conference on Learning Representations (ICLR)*, 2025.

Wenzhe Li, Yong Lin, Mengzhou Xia, and Chi Jin. Rethinking mixture-of-agents: Is mixing different large language models beneficial? *arXiv preprint arXiv:2502.00674*, 2025.

Lovish Madaan, Aniket Didolkar, Suchin Gururangan, John Quan, Ruan Silva, Ruslan Salakhutdinov, Manzil Zaheer, Sanjeev Arora, and Anirudh Goyal. Rethinking thinking tokens: LLMs as improvement operators. *arXiv preprint arXiv:2510.01123*, 2025. URL https://arxiv.org/abs/2510.01123v1.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

The Terminal-Bench Team. Terminal-bench 2.0. https://www.tbench.ai, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

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

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: Llm agents are experiential learners. In *AAAI Conference on Artificial Intelligence*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

A Experimental protocols

A.1 Source Representation

Fields and artifacts. Artifacts are produced files on Terminal-Bench 2.0, submitted SQL and query results on Spider2-Lite, submitted answers on BrowseComp-Plus, and action narratives and the final action summary on AndroidWorld. Empty or non-submitting attempts remain in the pool.

Truncation and overflow. ReGE-Syn and ReGE-A retain at most 120 rendered steps per source, with reasoning and commands truncated to 200 and 160 characters per step, respectively. Each artifact is capped at 6,000 characters. If the combined source serialization exceeds 350,000 characters, all artifact bodies are removed while their paths remain. Any remaining excess is truncated at that limit and followed by an explicit marker.

ReGE-Sel uses a different rendering: at most 120 steps with 250 reasoning and 600 command characters per step, without a separate final-artifact block. For Android UI steps lacking reasoning text, the renderer uses the recorded action narrative.

Direct history uses a step-only rendering that omits tool observations and separate final-artifact blocks.

A.2 Prompts and Guidance Delivery

The boxes reproduce construction prompts and delivery text, with placeholders and line breaks normalized for typesetting.

ReGE-Syn: Joint Synthesis Prompt

System message. You are an aggregation agent: several independent agents attempted a task (correctness unknown) and you synthesize the single best guidance for a fresh agent. You do NOT have access to the ground truth solution.
User message.
TASK: ⟨instruction⟩
K independent previous attempts (full trajectories + final files):
⟨serialized source pool⟩
Write an EXECUTABLE STEP-BY-STEP PLAN for a fresh agent that will solve the task from scratch: (1) numbered steps in execution order, each concrete enough to act on; (2) the critical facts the attempts uncovered – exact paths, values, commands, formats, edge cases – placed at the steps where they matter; (3) pitfalls the attempts exposed and how to avoid them. If the evidence already supports a final answer or artifact, state it in the plan together with how to verify it. The plan must be self-contained: the fresh agent sees only this plan, never the attempts.

ReGE-A: Alternative Construction Request

Produce your solution as a single string with exactly two XML sections: `<explanation>...</explanation><answer>...</answer>`. The explanation must be self-contained – a reader who never saw this prompt understands how the answer was derived. The answer is your best final solution to the task.

ReGE-Sel: Trajectory Selection Prompt

System message. You judge agent trajectories.
User message.
TASK: ⟨instruction⟩
K independent agents attempted this task (success labels hidden):
⟨rendered source trajectories⟩
Which ONE rollout is most likely FULLY correct end-to-end? Reply with EXACTLY one rollout id (e.g. claude-sonnet_1) and nothing else.

Up to eight judge attempts seek a parsable rollout ID, with requested output limits given in Table 10. Parsing prefers an exact ID, then the last valid ID mentioned; if all attempts fail, it selects the first source without consulting correctness labels. Cached picks avoid repeating selection during student collection.

Student Delivery Headers

Coach outputs (Syn, A, and Closed-book).
GUIDANCE (synthesized by an aggregation agent that inspected several previous independent attempts at this task; follow it but verify as you go):
Selected trajectory.
A PREVIOUS AGENT'S COMPLETE ATTEMPT (judged most likely correct among several independent attempts; follow it closely but verify as you go):
(chosen: ⟨rollout ID⟩, label-free judge)
Additional outer header on BrowseComp-Plus.
GUIDANCE (a cross-model-agreed, likely-correct approach — follow it):

Guidance precedes the instruction on Terminal-Bench 2.0 and Spider2-Lite, enters the native T3A task-guideline interface on AndroidWorld, and is appended to the user message on BrowseComp-Plus. The BCP outer header is legacy wording; the experiments use same-model source pools. RANDOM TRAJECTORY REPLAY instead identifies a random pick.

The revised AndroidWorld Sel deliveries below preserve the cached selected record. GPT replaces the selection header; Claude retains it and adds the reset clarification. Neither requires another judge or coach call.

AndroidWorld: Reset and Verification Instructions

GPT header and instruction.
RESET-AND-VERIFICATION GUIDANCE
The following material comes from a previous attempt at this task. Your environment has been reset to its initial state. Use current observations to determine which actions are needed, and verify the current task state before declaring completion.
Claude instruction, following the selected-trajectory header.
The following record comes from a separate, earlier execution. The environment for your current execution has been reset to the task's initial state. Statements about actions taken or completion in the record describe that earlier execution only; they do not establish the state or completion of your current task.

Independent-summary construction. Each independent-summary call receives the task and one block from Syn's bounded pool serialization, inheriting its truncation and artifact-redaction decisions. The reconstructed prompt requests one JSON object with four groups: task requirements, agent actions, artifact changes, and command execution. Initial length instructions allow approximately 1,000 words, six command entries, six items per list, and 250 characters per string, prioritizing concrete commands, paths, values, and failure causes. Full templates are included in the reproduction materials.

Up to three formatting attempts retain the first complete JSON object containing the required task-requirement and agent-action keys. Retries request 500 words, three commands, four items per list, and 120 characters per string; the output-token request remains fixed at the value in Table 10. If none passes, the last nonempty response is kept and flagged; an all-empty response becomes a placeholder. There is no quality-based selection or fallback to raw trajectories.

Usage counts all recorded calls for each delivered summary, including its original generation when a cached summary is reused. Five Terminal-Bench source blocks fall outside the shared serialization budget and receive placeholders, leaving 351 summaries; all 1,200 BCP blocks are summarized. Section 4.4 reports the BCP comparison; the reproduction README indexes full-set and Terminal-Bench results, exact prompt templates, and cache-matching records.

A.3 EVALUATION AND TOKEN ACCOUNTING

Serving and collection conditions. Table 10 distinguishes requested limits from serving evidence. The comparisons measure realized usage under method-specific construction and stopping procedures, not equal total-token caps. Revised AndroidWorld Sel runs were collected separately from the other methods; Claude's 114 tasks combine a 48-task probe and its 66-task complement after interim inspection. Pairing task IDs does not remove collection-window differences. The generated `MarkorCreateFolder` target differs across campaigns, and historical image pixels and some parameter fields cannot be fully matched. This accounts for the 339-instance GPT cost subset; template-aware sensitivity results are retained in the reproduction materials. The Claude Terminal-Bench 2.0 source pool also spans thinking-enabled collection on 69 tasks and thinking-disabled collection on 20; downstream judge, coach, and student runs used thinking-disabled serving configurations. Detailed collection records are indexed in the reproduction README.

Table 10: **Requested generation limits and available serving evidence.**

Component	Output-token request	Scope / evidence
Syn, A, Closed-book	10,000 per call	Shared construction implementation.
Sel judge	2,000 / 8,000 / 20,000	Attempts 1–2 / 3–4 / 5–8, respectively.
AGGAGENT	10,000 per round	Default limit of 100 aggregation rounds.
Independent summaries	2,500 per source	Matched PDR adaptation; Appendix A.2.
Claude Terminal-Bench 2.0 student	12,000 per call	30 agent turns; main records retain the output-token request.
GPT Terminal-Bench 2.0 student	Unknown	30 agent turns; historical output-token request not recorded.
Spider2-Lite/AndroidWorld students	No explicit cap	Main collector request paths omit an output limit.
BrowseComp-Plus student	4,000 per call	Main collector request.
Serving snapshot	**Output-limit handling**	**Other effective settings**
GPT gateway	Requested caps dropped; no proxy-set replacement	Sampling requests dropped; reasoning effort left at the serving default.
Claude gateway	Default 16,000 when no cap is supplied	Sampling requests dropped; thinking disabled in the recorded snapshot.

The upper rows give requested settings, not guaranteed effective limits. September gateway snapshots establish the lower rows for those configurations; they do not attest every earlier request. Missing historical settings remain unknown. Requested caps therefore do not establish equal realized token budgets.

B SOURCE POOLS AND REPEATED EXECUTION

B.1 SOURCE-POOL CHARACTERISTICS

Table 11: **Source-pool coverage.**

Benchmark	GPT coverage (%)	Claude coverage (%)
Terminal-Bench 2.0	64.0	47.2
Spider2-Lite	75.6	73.3
AndroidWorld	64.0	69.3
BrowseComp-Plus	76.1	85.9

Coverage is the fraction of tasks with at least one successful source attempt, using all source pools. Pool sizes are specified in §3; later student analyses use common valid execution records.

Six-source experiment. On a separate 100-query Claude BCP subset, student success is 81%, 82%, 85%, and 89% with one, two, four, and six sources, respectively, versus 70% for CONTROL and 87% source-pool coverage. Six versus four gives $+4/-0$ ($p = .125$). This subset differs from the 300-query comparison in Table 5; the results should not be joined into a single scaling curve.

B.2 RE-EXECUTION FROM A SUCCESSFUL SELECTED SOURCE

Success of the selected source. Table 12 reports fresh student success conditional on the judge selecting a successful source; the same judge pick defines the subset for every method, including those that do not use it.

Table 12: **Fresh student success given a successful selected source.**

Benchmark	n	CONTROL	AGGAGENT	REGE-SEL	REGE-SYN
gpt-5.6-luna					
Terminal-Bench 2.0	47	68.1	78.7	**97.9**	89.4
Spider2-Lite	86	89.5	79.1$^{\diamond}$	95.3	**96.5**
AndroidWorld	201	82.1	76.6	72.6	**87.6**
BrowseComp-Plus	614	83.9	98.4	**100.0**	98.7
claude-sonnet-5					
Terminal-Bench 2.0	34	79.4	91.2	**100.0**	85.3
Spider2-Lite	83	86.7	89.2$^{\diamond}$	**95.2**	**95.2**
AndroidWorld	74	87.8	93.2	**94.6**	91.9
BrowseComp-Plus	703	85.6	99.1	**99.7**	99.0

Rates (%) measure new student execution, not the source's historical outcome. All four methods use the same common-valid subset of n tasks with a successful judge-selected source. Bold marks the highest observed rate per row, including ties. AndroidWorld Sel uses the later delivery revisions described in Appendix A.3.

B.3 REPEATED STUDENT EXECUTIONS

On the 35 Claude Spider tasks with mixed four-source pools, Control and Syn each have three student executions with fixed source experience, model, and guidance settings; the first is the main-table execution.

Table 13: **Three student executions on 35 mixed-pool Claude Spider tasks.**

Metric	CONTROL	REGE-SYN
Mean success (%)	48.6	63.8
Tasks solved on all three executions	9	21
Tasks solved on at least one execution	25	23

Mean success averages all three executions; both coverage counts are out of the same 35 tasks. Per-execution results are retained in the reproduction materials.

An exact task-level test swaps each task's complete three-execution vectors between methods, giving $p = .08896$. This 35-task, source-defined subset does not establish a significant benchmark-wide accuracy gain or a general reduction in execution variance.

C STUDENT EXECUTION ACCOUNTING

Table 4 sums recorded model output tokens over each student execution. Steps count trace entries on Terminal-Bench 2.0 (including the initial snapshot), SQL calls on Spider2-Lite, and recorded agent steps on AndroidWorld. Ratios divide method means on the common valid task set, rather than averaging per-task ratios. The calculation uses output tokens only; input and cache usage were not fully available for this analysis.

D ADDITIONAL EXECUTION CASES AND EVIDENCE

The cases below use recorded Claude executions and the same source–guidance–student structure as §4.6. They are retrospective illustrations; source outcomes are used only for analysis.

D.1 EVIDENCE FOR THE MAIN-TEXT CASES

Table 14 documents the starting conditions and record linkage for the two main-text cases.

Table 14: **Record checks for the main-text execution cases.**

Check	Recorded evidence
Database initial state	Initial database and log fingerprints match across the source attempts and student; source-created backups and recovered outputs are not inherited.
Browser task instance	Source and student task parameters, both random seeds, and the 22-step budget match. The collector resets and initializes each task. Saved summaries record the opening sequence, product entry, and success message.
Record linkage	Student records lack an injected-guidance hash; linkage uses saved guides, construction/execution logs, and collector configurations, indexed in the reproduction materials.

D.2 Adapting a Database Query to Stored Values

Claude Spider2-Lite task `local098` asks how many actors have no four-year interval without a film credit between their first and last credited years.

Execution Case: Checking Year Formats before Counting Actors

Recorded experience. The four sources return different counts after varying how they parse `Movie.year` and bound the gaps between credited years. Some extract the last four characters; others cast or filter the raw year strings.

Synthesized guidance. The guide retains distinct actor–year pairs and a `LAG` calculation, with consecutive credited years at most four years apart. It also recommends filtering for raw four-digit year strings and asks the student to inspect the stored format first.

Student verification and adjustment. The student queries nonstandard year strings and their four-character suffixes, then checks for nulls and invalid suffixes. Its final query uses `CAST(substr(year,-4) AS INTEGER)` instead of the guide's strict raw-string filter, while retaining the proposed gap calculation.

Execution outcome. The submitted query returns 28,699 actors, matching an accepted official result. The record contains ten SQL calls.

The student retains the query structure while adapting preprocessing to the data. Intermediate calls retain SQL prefixes and status flags; the final query and result CSV are complete.

D.3 Reusing Document Leads for Fresh Verification

Claude BrowseComp-Plus query `bcp-285` asks for the student who succeeded a scientist as director of a research center, using clues about the center, the scientist, and a co-edited publication.

Execution Case: Following a Research Lead through Source Documents

Recorded experience. One attempt reaches the Citrus Experiment Station and Herbert J. Webber, then submits Leon Dexter Batchelor. Two other attempts reach relevant documents without submitting an answer; the remaining attempt spends its budget on other leads.

Synthesized guidance. The guide retains the institution and person names, document IDs `9606` and `26507`, and a reminder to read beyond the first excerpt. It prioritizes this lead and lists clues that still require checking.

Student verification and adjustment. The student searches for the institution, reads document `9606` at offsets 0 and 6,000, and opens Webber's biography in document `26507`. These passages provide the succession and biographical evidence; an additional search checks the University of Nebraska's charter date.

Execution outcome. The student submits "Leon Dexter Batchelor," which receives an exact-match success grade, after six recorded tool steps including submission.

The guide preserves both a candidate answer and the document locations that let the student check the research lead.