DATAWORKLEARN: THE WORKSPACE IS THE TRAINING GROUND FOR DATA AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Data agents, LLM agents that query, transform, and maintain data on a user's behalf, work inside workspaces of interdependent sources: databases and files, documentation, and layered pipelines whose definitions depend on one another, so correctness depends on how these pieces fit together. We introduce DATAWORKLEARN, which lets an agent learn a workspace before the first question arrives, from only the workspace and a short description of the expected work, with no labeled tasks, hand-written verifiers, or human in the loop. The agent explores the workspace, writes tasks together with executable solutions over its sources and pipelines, and verifies every solution by execution rather than by a model's judgment. Because every task is an instruction, a starting environment, and a verifier, we use the same tasks to adapt the agent through reinforcement learning on its weights, through learned skills, and through harness optimization, and compare the three on identical tasks. Across four data-analysis and data-engineering benchmarks, DATAWORKLEARN improves a 27B open-weight model by up to 22.5 points on the Data Agent Benchmark (DAB) and 22.4 on DABstep-hard, outperforming GEPA, EvoSkill, and Meta-Harness, each adapted on the same generated tasks, by up to 18.8 points. A data workspace can thus serve as verified, label-free training signal before any question is asked, not only as context at inference time.

1 INTRODUCTION

LLM agents can resolve real-world coding tasks in large repositories Jimenez et al. (2024); Yang et al. (2024), operate websites and desktop applications Zhou et al. (2024); Xie et al. (2024), and perform multi-step data analysis Ma et al. (2024); Sahu et al. (2025). Data agents, which query, transform, and maintain data on a user's behalf, work inside a data workspace: the databases, files, spreadsheets, code, pipelines, and documentation accumulated from data work Ma et al. (2026); Egg et al. (2025); Lei et al. (2026); Tang et al. (2026). A general-purpose agent arriving in a new workspace knows neither what it holds, how its artifacts relate, nor which procedures and tools work there, and the questions it will be asked do not exist yet. We ask whether an agent can learn a workspace before the first question arrives, and call this problem **workspace adaptation**: the agent receives an unseen workspace and a broad description of the expected work, as is standard in task synthesis, but no concrete questions, solutions, gold outputs, graders, or evaluation feedback.

Learning from new data has long been studied through continual knowledge learning and domain adaptation, where models update their knowledge from new corpora Jang et al. (2021); Yang et al. (2025). A workspace also requires learning *how to work with it*: how to inspect heterogeneous artifacts, combine files and databases, and carry out its analytical workflows under a fixed budget. This is harder than in the code repositories where environment-based agent learning has mostly been studied. A data workspace is too large to read end to end, and correctness is rarely local: it depends on which identifiers align across sources, how the layers of a pipeline feed one another, and the conventions the documentation fixes Lei et al. (2025); Li et al. (2026); Tang et al. (2026), so a reasonable-looking step can fail silently. Such couplings are learned by working in the workspace, yet existing approaches assume more than a workspace provides. Methods that adapt an agent's prompts, skills, harness, or weights optimize against target tasks that come with feedback, such as reference answers, tests, or graders Agrawal et al. (2026); Alzubi et al. (2026); Lee et al. (2026); Zhang et al. (2026b), and methods that synthesize practice from an environment judge their tasks with a model or rely on tests the environment already contains Yang et al. (2026); SU et al. (2025);

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.

Sun et al. (2025). In workspace adaptation the agent has neither, only the workspace and a brief description of its workload.

To this end, we introduce DATAWORKLEARN, a framework for learning from data workspaces. It receives the target environment, its schemas, data, project structure, and executable interfaces, together with a broad scenario brief, but not the downstream task content. For each practice task, an author agent writes the *solution first* as an executable program, a typed dataflow graph for analysis and pipeline construction or an edit to the project files for repository-level engineering, and executing it in the sandbox yields the reference outcome. Rather than trusting a model-written answer, DATAWORKLEARN builds the verifier from this executed outcome and adds an instruction and plausible wrong solutions, so the learner sees each task as an instruction, a starting environment, and a verifier.

The practice set supplies exactly what existing adaptation methods assume but a workspace does not provide: tasks with executable feedback. Because every task exposes the same interface, an instruction, a starting environment, and a verifier, one set can drive any of these methods: its verifiers give the rewards for *parametric adaptation* through reinforcement learning Shao et al. (2024) and the evaluation signal for *skill adaptation*, which distills reusable procedures from successful practice Alzubi et al. (2026), and *harness adaptation*, which searches the instructions, tools, or control logic around a fixed model Agrawal et al. (2026); Lee et al. (2026). Decoupling the practice from the learner also lets us ask which part of an agent should adapt to a workspace on identical data, a comparison that is confounded when each method is evaluated with its own task set.

We evaluate DATAWORKLEARN on four benchmarks spanning data analysis and data engineering: the Data Agent Benchmark (DAB), DABstep, ADE-Bench, and DAComp-DE-Verified. Workspace adaptation improves both model scales on all four; on Qwen3.8-27B it raises DAB from 45.6 to 68.1 and DABstep-hard from 19.3 to 41.6, with the largest gains on the hardest splits. Our analysis on DAB shows that searching the prompt, skills, or harness saturates near the initial agent, because gains on the practice tasks barely transfer to the benchmark, whereas reinforcement learning keeps converting compute into accuracy and makes the agent cheaper at the same time, halving its turns without any efficiency reward. Compared with three other task generators, only our practice tasks are compositional and cross-database like the benchmark's questions, and they improve a 9B agent the most (+13.4 points, against at most +6.4). These results suggest that a user's workspace can do more than serve as passive context retrieved at inference time. An agent can learn from the workspace itself, adapting its knowledge, skills, and behavior before it knows exactly what the user will ask.

Our contributions are fourfold. (1) We formulate **workspace adaptation**, in which an agent must improve on an unseen data workspace given only the workspace and a brief workload description. (2) We introduce DATAWORKLEARN, which writes practice tasks that are correct by construction to yield $(\text{instruction}, \text{environment}, \text{verifier})$ triples. (3) Because every task takes this form, we adapt the agent through its weights, its skills, and its harness on identical practice tasks. Reinforcement learning is the only interface that improves every benchmark and split and gives the largest gain on six of the seven metrics. (4) DATAWORKLEARN improves 9B and 27B open-weight models on all four benchmarks, by up to 22.5 points on DAB and 22.4 on DABstep-hard, and outperforms GEPA, EvoSkill, and Meta-Harness trained on the same tasks on every metric but one.

2 RELATED WORK

Synthetic experience for agent learning. Synthetic supervision ranges from instruction–response bootstrapping in Self-Instruct to artifact-seeded agentic generation in AgentInstruct Wang et al. (2023b); Mitra et al. (2024). For interactive agents, SWE-Gym packages real repository tasks in executable environments, SWE-smith synthesizes repository-level bugs, and GUI-agent work builds trajectories from environment interaction Pan et al. (2024); Yang et al. (2026); Xu et al. (2025); Sun et al. (2025). DataEnvGym generates data from a student model's feedback, and Learn-by-Interact relabels agent trajectories on documentation-derived tasks with matching task descriptions Khan et al. (2025); SU et al. (2025). Recent systems scale terminal-task generation or synthesize stateful tool and database environments with verifiable tasks Pi et al. (2026); Dong et al. (2026). Apart from Learn-by-Interact, which adapts to a target environment through model-judged trajectories, these methods primarily target general capabilities; DATAWORKLEARN instead generates execution-verified experience from a target workspace to adapt the agent to it, without downstream tasks.

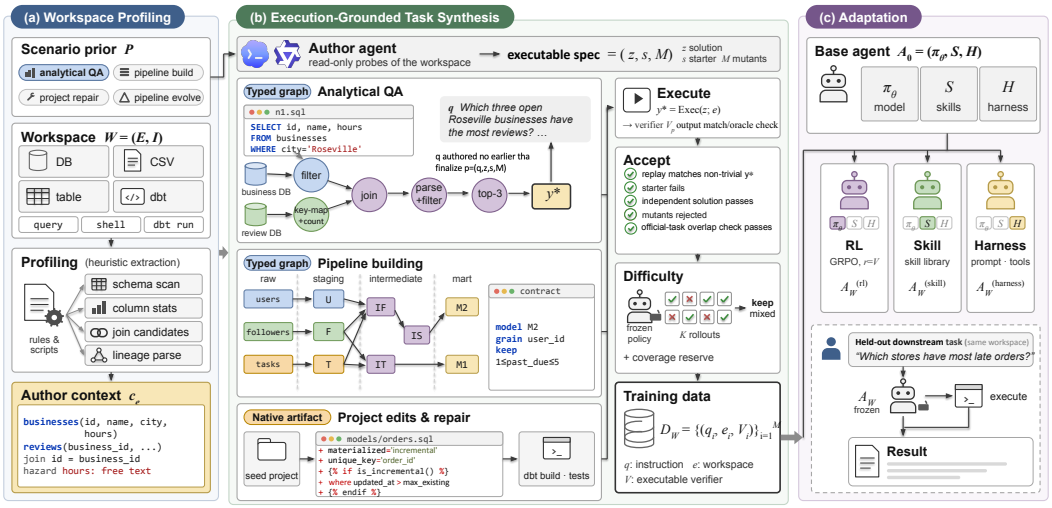


Figure 1: Overview of DATAWORKLEARN. Panels (a) and (b) form the Workspace Learner. (a) Scripts profile the workspace into an author context c_e. (b) An author agent writes an executable solution, either a typed graph or a native project edit. Running it gives the answer $y^\star$, around which the instruction and verifier are built; a task is kept if it passes the acceptance checks and the frozen model does not always solve it. (c) The task set $\mathcal{D}_\mathcal{W}$ is used to adapt the model, skills, or harness.

Self-learning and continual agent adaptation. Self-improvement methods generate, filter, or reward reasoning experience, and domain-adaptation methods adapt models to new corpora Zelikman et al. (2022); Singh et al. (2023); Guan et al. (2025); Zhao et al. (2026); Zhang et al. (2024); Yang et al. (2025). Early Experience learns from future states of an agent's own interactions without reward labels Zhang et al. (2026a). Beyond model weights, Voyager, ExpeL, and Agent Workflow Memory accumulate skills, insights, or workflows Wang et al. (2023a); Zhao et al. (2024); Wang et al. (2024); SkillWeaver distills reusable APIs through exploration, and SkillLearnBench benchmarks continual skill generation Zheng et al. (2025); Zhong et al. (2026). ACE evolves context playbooks from execution feedback, ContDa updates tool documentation, and Continual Harness adapts prompts, memory, sub-agents, and skills Zhang et al. (2026b); Wu et al. (2026); Karten et al. (2026). None compares parameter, skill, and harness adaptation on the same experience, which we build from a target data environment without downstream tasks.

Data-analytic and data-engineering agents. Data agents combine reasoning, code generation, and tool use for tabular analysis, data science, database querying, spreadsheets, business analytics, and pipeline development Hu et al. (2024); Huang et al. (2024); Lei et al. (2025); Ma et al. (2024); Sahu et al. (2025); Lei et al. (2026). Training resources range from synthetic text-to-SQL data in OmniSQL to generalist data-agent training in DataMind-12K Li et al. (2025); Qiao et al. (2026). VeriGraph reasons online over an executable evidence DAG, whereas our typed graphs are solution specifications, fixed before any agent runs, from which supervision is derived Jin et al. (2026). Recent benchmarks cover heterogeneous database systems in DAB, mixed artifact formats in DataSpace, and inter-file dependencies in Workspace-Bench Ma et al. (2026); Li et al. (2026); Tang et al. (2026). These works build general resources, training recipes, or benchmarks; we instead adapt an agent's model, skills, or harness to the data environment it will operate in.

3 METHOD

DATAWORKLEARN builds a synthetic training set from a data workspace and uses it to adapt an agent to that workspace (Figure 1): an author writes executable solutions, execution yields the tasks and their verifiers (§3.2), and the task set then adapts the model's weights, skills, or harness (§3.3).

3.1 SETTING AND TASK FORMAT

Setting. A data workspace $\mathcal{W} = (\mathcal{E}, \mathcal{I})$ has two parts: the data environments $\mathcal{E}$ (e.g., databases, files, SQL and Python dbt code) and the interfaces $\mathcal{I}$ used to work with them (e.g., a SQL tool, a shell, or specific available shell commands including `dbt run`). The agent to be adapted, $\mathcal{A}_0 = (\pi_\theta, \mathcal{S}, \mathcal{H})$, has three parts: a model policy π_θ, a library of reusable skills $\mathcal{S}$, and a harness $\mathcal{H}$, i.e., the prompts, tools, and control code around the model. The only other input is a short scenario prior $\mathcal{P}$ naming the expected work: answering analytical questions, building a pipeline, repairing a project, or evolving a pipeline. The Workspace Learner turns $\mathcal{W}$ and $\mathcal{P}$ into a task set $\mathcal{D}_\mathcal{W}$, and an adaptation method m produces $\mathcal{A}_\mathcal{W}^{(m)} = \mathrm{Adapt}_m(\mathcal{A}_0, \mathcal{D}_\mathcal{W})$, which we freeze and evaluate on the benchmark's held-out official tasks (Appendix B).

Task Format. Each generated task comes from a plan $p = (q, z, s, \mathcal{M})$ with four parts: (1) the *instruction* q that the learner sees; (2) a *solution* z, an executable program whose execution produces the reference result; (3) an optional *starter* s, the state the learner starts from, such as a dbt project with a missing model; and (4) *plausible but wrong solutions* $\mathcal{M}$, such as ones with a wrong data join or a missing filter. Running z gives the reference result $y^\star$, from which we build a *verifier* V_p that checks the learner's final answer or database state. The learner receives q and the workspace (with s, if any) in the benchmark's own task format and is scored by V_p.

3.2 EXECUTION-GROUNDED TASK SYNTHESIS

Rather than have a model state an answer it may get wrong, the author writes an executable solution z, a typed graph or a project edit, and we obtain the reference result $y^\star$ by running z in the workspace. The result is therefore produced under the workspace's own semantics rather than asserted by the model, and the instruction q, the wrong solutions $\mathcal{M}$, and the verifier V_p are all derived from it before the acceptance and difficulty gates below.

Workspace Profiling. For each environment $e \in \mathcal{E}$, scripts extract an author context c_e: table schemas, row counts, column statistics, candidate join keys, the lineage of dbt models, and known data issues such as opening hours stored as free text. The author, Qwen3.8-27B running in the Codex coding-agent harness, receives c_e, the scenario prior, and read-only query tools.

Solution Authoring. The author writes z in one of two executable representations, chosen by how the task is graded. A *typed graph* is used whenever the outcome is graded by comparing output tables: z is a directed acyclic graph $G = (V, E)$ in which each node $v \in V$ is a typed operator that outputs a table and each edge $(u, v) \in E$ passes the output of u to v. Analytical questions, building a pipeline, and modifying an existing pipeline all take this form. In an analytical task, the nodes are operations such as database queries, joins, text parsing, and top-k selection, and one terminal node gives the answer; the author composes the graph from the operators of the task family. In a data pipeline task, the nodes are the SQL models of a layered pipeline (raw, staging, intermediate, and mart), the mart tables are graded, and a compiler turns the graph into SQL; we sample a skeleton of layers and dependencies and the author fills in each node. Graphs that fail validation, such as an operator with the wrong number of inputs, go back to the author. A *native artifact* is used when the outcome is a change to a project rather than a table, such as repairing, refactoring, or adding models to a dbt project: z is written as edits to project files, which can include dbt SQL files, s is the project before the edit, and grading runs hidden tests over the resulting project and database state.

Execution and Verifier Construction. We run z in its environment e to obtain the reference result $y^\star = \mathrm{Exec}(z; e)$: the output tables for graph tasks, or the resulting project and database state for native-artifact tasks. For graph tasks, we then write the instruction q to request exactly what the executed graph computes, a natural-language question whose answer is the terminal table $y^\star$ for analytical tasks, or a data contract or change request describing the target tables for pipeline tasks. The wrong solutions $\mathcal{M}$ come from perturbing the graph or the answer, e.g., changing a join type, dropping a filter, or altering answer values. For native-artifact tasks the edit itself defines the work, so the author writes q and $\mathcal{M}$ alongside z. The verifier V_p compares the learner's result with $y^\star$. Since a reference edit trivially matches its own output, native-artifact tasks are graded by hidden tests from an independent oracle query instead.

Acceptance Checks. A task is kept only if it passes four automatic checks. (1) *Replay*: a fresh run of z reproduces the same non-empty $y^\star$. (2) *Starter check*: the unmodified starter s, if any, fails V_p. (3) *Negative controls*: V_p rejects at least one solution in $\mathcal{M}$ and accepts an independent correct solution if one exists. (4) *Overlap check*: the task does not nearly copy an official task's instruction or graded output. Failed tasks are discarded or, in some environments, repaired by the author.

Difficulty Filtering. A task is useful only if the model to be trained cannot already solve it every time, so a frozen copy of that model attempts each task four or eight times through the full agent loop. With binary rewards, we keep tasks solved in some but not all attempts. For those with graded rewards, which give partial credit, we remove only tasks solved in every attempt or with a high mean score. Some workspaces also keep a small reserve of never-solved tasks so that every task family stays represented. The kept tasks form $\mathcal{D}_{\mathcal{W}} = \{(q_i, e_i, V_i)\}_{i=1}^{M}$, where e_i is the starting workspace of task i. Because the filter keeps only tasks the model cannot already solve consistently, and models differ in what they can already do, each model gets its own task set (Appendix B).

3.3 Adapting from Workspace-Derived Tasks

All adaptation methods learn from the same $\mathcal{D}_{\mathcal{W}}$ and differ only in which part of $\mathcal{A}_0$ they change.

Parametric Adaptation. We train π_θ with Group Relative Policy Optimization (GRPO) Shao et al. (2024), using each task's verifier as the reward. For a task (q, e, V), the policy samples a group of N trajectories $\{\tau_j\}_{j=1}^{N}$ through the full tool-use loop, and τ_j receives reward $r_j = V(\tau_j, e)$. With μ_r and σ_r the mean and standard deviation of the rewards in a group, the normalized advantage and token-level importance ratio are

$$\widehat{A}_j = \frac{r_j - \mu_r}{\sigma_r + \epsilon}, \qquad \rho_{j,t}(\theta) = \frac{\pi_\theta(a_{j,t} \mid h_{j,t})}{\pi_{\theta_{\text{old}}}(a_{j,t} \mid h_{j,t})}, \tag{1}$$

and the policy maximizes the clipped surrogate objective

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}\Big[\operatorname{Agg}_{j,t} \min\big(\rho_{j,t}\widehat{A}_j,\ \text{clip}(\rho_{j,t}, 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}})\widehat{A}_j\big) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\Big], \tag{2}$$

where $\operatorname{Agg}_{j,t}$ averages over sampled trajectories and generated policy tokens. DAB and DABstep use binary rewards, ADE-Bench uses the fraction of hidden tests passed, and DAComp uses CFS normalized to $[0, 1]$. During training, small penalties discourage malformed outputs, tool failures, and exhausting the interaction budget (Appendix C).

Skill and Harness Adaptation. These methods keep π_θ fixed. Skill adaptation turns successful trajectories into reusable procedures in $\mathcal{S}$; we use EvoSkill (Alzubi et al., 2026). Harness adaptation changes $\mathcal{H}$; we use GEPA (Agrawal et al., 2026) for prompt optimization and Meta-Harness (Lee et al., 2026) to search the harness code. Appendix D adds a third harness search, HarnessLab.

4 Experimental Setup

Benchmarks and Metrics. We evaluate DataWorkLearn on four executable data-agent benchmarks: **DAB** for cross-database analysis Ma et al. (2026), **DABstep** for multi-step data analysis over data files and documentation Egg et al. (2025), **ADE-Bench** for tested analytics-engineering edits dbt Labs (2026), and the data-engineering split of **DAComp** Lei et al. (2026). We report **Pass@1**, **accuracy** on the hard and easy splits, **test pass rate**, and **Cascading Failure Score (CFS)**, respectively, on a 0–100 scale. Because scorer and harness defects in the released DAComp suite prevent reproducible evaluation, we construct **DAComp-DE-Verified** by repairing these defects. The resulting suite contains 43 reproducibly scored tasks: 13 implementation and 30 evolution tasks. In every benchmark, self-learning receives the data environment and broad scenario priors but no downstream task content. Appendix A provides benchmark details, evaluation protocols, and the formal definition of CFS.

Models and Agent Harness. We experiment with two open-weight agent models at different scales: **Qwen3.5-9B** and **Qwen3.8-27B**. On each benchmark, both models start from the same

benchmark-specific harness. Both models are trained with RL; unless otherwise noted, Qwen3.8-27B is the default model for comparing adaptation methods. Each harness provides file-system inspection, shell and Python execution, and benchmark-specific database and data-processing tools.

Baselines. We compare against the base agent (**No Adaptation**) and three adaptation methods that keep model weights fixed and learn from the same generated tasks: reflective prompt optimization with **GEPA** (Agrawal et al., 2026), skill evolution with **EvoSkill** (Alzubi et al., 2026), and harness-code search with **Meta-Harness** (Lee et al., 2026). Each keeps its original adaptation interface but uses our data-agent tools. As reference points, we also run six proprietary models in each benchmark's harness (Table 1). Comparisons with other task generators are specific to DAB and described in §5.3.

Implementation Details. For RL on both models, we use rank-32 LoRA ($\alpha = 64$) on all linear layers, a learning rate of 5×10^{-5}, 16 task prompts per batch, and 16 rollouts per prompt. Rollouts use temperature 1.0, top-p 1.0, and a 131,072-token context; training runs for up to 100 steps. Appendix C gives the optimization, reward, budget, and checkpoint settings. GEPA, EvoSkill, and Meta-Harness run on Qwen3.8-27B served through OpenRouter, each with its original search procedure on the generated tasks. At evaluation, all components are frozen and each benchmark's official environment and evaluator are used.

5 Experimental Results

We organize our experiments around five research questions: **RQ1:** Does workspace self-learning improve data agents across different analytical environments? **RQ2:** Which adaptation interface makes the best use of workspace experience, and at what cost? **RQ3:** What experience does the Workspace Learner generate, and how does it differ from other task generators? **RQ4:** Are harness learning and reinforcement learning complementary? **RQ5:** How do accuracy and efficiency evolve during adaptation?

5.1 RQ1: Effectiveness of Workspace Self-Learning

Table 1 summarizes the results of the base agents, the three adaptation baselines, DataWorkLearn and six proprietary models on the four benchmarks. We have the following observations. (i) **Workspace self-learning improves both models on every benchmark.** DataWorkLearn raises Qwen3.8-27B from 45.60 to 68.06 on DAB and Qwen3.5-9B from 30.55 to 47.68, and improves every split of DABstep, ADE-Bench and DAComp-DE-Verified with available results, across workspaces as different as cross-database querying, multi-step file analysis and analytics engineering. (ii) **The harder the split, the larger the gain.** DABstep-hard more than doubles for both models (19.26→41.64 and 7.65→16.43) and DAComp implementation tasks gain 9.1 and 8.2 points, whereas the easy DABstep split, which the base agents already largely solve, gains 4–6 points. (iii) **Adapted open-weight models close much of the gap to proprietary ones.** Adapted Qwen3.8-27B (68.06 on DAB, 41.64 on DABstep-hard) is ahead of GPT-5.6-Terra, Claude Sonnet 5 and Gemini 3.5 Flash on both and close to GPT-6 Astra on DAB (70.37). (iv) **Adaptation without weight updates gives small and uneven gains.** On the same generated tasks, GEPA, EvoSkill and Meta-Harness help on some columns and hurt on others: GEPA gains 11 points on DABstep-hard but loses 7 on the easy split, all three lower ADE-Bench (by 2.7–5.0), and EvoSkill loses 14 points on DAComp implementation. DataWorkLearn improves every column on which it is evaluated, most on DAB and DABstep-hard.

5.2 RQ2: Comparison of Adaptation Interfaces and Their Cost

To understand which adaptation interface makes the best use of the generated tasks, we compare DataWorkLearn with prompt search (GEPA), harness-code search (Meta-Harness) and skill evolution (EvoSkill) on DAB. All methods start from the same initial harness and learn from the same generated tasks. On DAB, DataWorkLearn has two stages: a harness-learning stage searches for a better harness, and an RL stage then trains the model inside it. We measure cost as the inference tokens the 27B model reads and generates on generated tasks, including the searches' reflection and proposer calls and excluding validation, and evaluate on DAB every harness a search proposes and every fifth RL checkpoint up to step 65, the end of its plateau. Two costs are estimated: the

Method	DAB	DABstep		ADE-Bench	DAComp-DE-Verified		
	Pass@1 ↑	Hard ↑	Easy ↑	Pass rate ↑	All ↑	Impl. ↑	Evol. ↑
Proprietary models							
GPT-6 Astra	70.37	63.17	94.37	64.06	33.94	16.54	41.48
GPT-5.6-Terra	62.96	9.35	38.03	68.75	38.74	32.73	41.34
Claude Fable 5.1	77.78	60.06	94.37	68.75	46.70	51.67	44.54
Claude Opus 5	79.63	46.18	91.55	67.19	43.79	43.68	43.84
Claude Sonnet 5	55.56	25.50	83.10	51.56	39.34	29.74	43.50
Gemini 3.5 Flash	44.44	1.42	73.24	53.12	27.97	29.95	27.11
Open-weight models							
Qwen3.5-9B	30.55	7.65	77.46	40.38	19.24	8.03	24.11
w/ DATAWORKLEARN	**47.68**$_{+17.1}$	**16.43**$_{+8.8}$	**81.69**$_{+4.2}$	**43.46**$_{+3.1}$	**23.39**$_{+4.2}$	**16.19**$_{+8.2}$	**26.51**$_{+2.4}$
Qwen3.8-27B	45.60	19.26	85.92	71.48	34.39	25.85	38.10
w/ GEPA	49.31	30.59	78.87	68.83	36.74	20.92	**43.60**
w/ EVOSKILL	45.60	17.85	88.73	68.42	31.72	11.76	40.38
w/ Meta-Harness	45.14	18.13	84.51	66.53	38.74	29.14	42.91
w/ DATAWORKLEARN	**68.06**$_{+22.5}$	**41.64**$_{+22.4}$	**91.55**$_{+5.6}$	**72.58**$_{+1.1}$	**39.01**$_{+4.6}$	**34.98**$_{+9.1}$	40.79$_{+2.7}$

Table 1: Main results. Within each benchmark, all models use the same harness; scores are benchmark-native on a 0–100 scale. Each indented "w/" row applies one adaptation method to the open-weight model above it. Shaded rows are ours, trained with GRPO, with gains over that model in green; bold marks the best result within each open-weight model. GEPA, EvoSkill, Meta-Harness, and the unadapted Qwen3.8-27B on DAB run on the same model served through OpenRouter. DAB, ADE-Bench, and open-weight DAComp report Avg@4; the others are single runs.

Meta-Harness proposer, whose coding-agent CLI does not report usage, from its transcripts, and our harness-learning stage, which was not logged, at the cost of GEPA's initial search (808 episodes, 0.23B tokens).

Figure 2a shows accuracy against cost. We can see that the search-based methods saturate near the initial agent: over 0.9B (GEPA) and 1.4B (Meta-Harness) tokens, the 44 proposed harnesses stay between 40 and 50 around the seed's 45.6, and the harness each search returns ends at 49.3 and 45.1; doubling or tripling their budget does not move this band. EvoSkill does no better: none of its seven skill sets beats the seed on its search split, so its own stopping rule returns the unadapted agent. Our harness-learning stage behaves like these searches (45.6→48.4), but the RL stage that follows keeps converting compute into accuracy and reaches **68.1** after 3.7B tokens.

Why do the searches stall? Figure 2b places every state against its gain on the tasks it optimised. The RL stage carries 57% of its in-sample gain to DAB, whereas a proposal's gain on its search split says almost nothing about DAB: the least-squares slope is -0.01 for GEPA and 0.15 for Meta-Harness. This suggests that a search, which selects among candidates on a few dozen noisy tasks, mostly fits that split, while RL, which updates the policy on hundreds of tasks, learns behaviour that transfers.

5.3 RQ3: ANALYSIS OF THE GENERATED EXPERIENCE

To understand what experience the Workspace Learner generates, we compare our tasks with three task generators that could feed the same RL loop: DataMind-12K, a released training set generated by DeepSeek-V3/V3.1 Qiao et al. (2026), and OmniSQL Li et al. (2025) and Agent World Model (AWM) Wang et al. (2026), which we run from their official code on the DAB databases with Qwen3.8-27B (Appendix B). We label 200 random tasks per source with the operations a correct solution requires, using the same 27B prompt, and train Qwen3.5-9B with RL on 308 tasks from each source.

Figure 3a shows that the generators differ in what they ask. Our tasks combine 3.1 operation categories and 67% join databases, close to DAB's 2.5 and 31%, whereas the other generators combine 1.0–1.7 categories and rarely need a second database (0–10%), because they phrase each task from a single

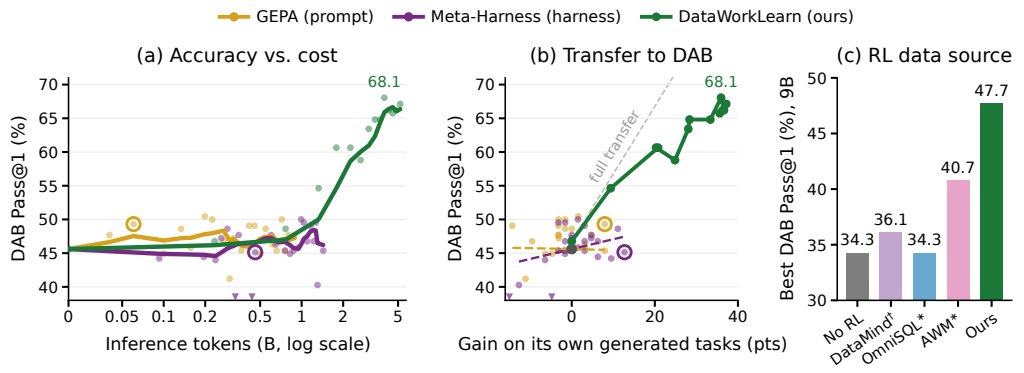


Figure 2: Cost and transfer of adaptation on DAB. (a) Pass@1 vs. inference tokens. (b) Pass@1 vs. gain on each method's own generated tasks. (c) Qwen3.5-9B after RL on 308 tasks per generator (†released data; *stopped after 15 and 20 steps).

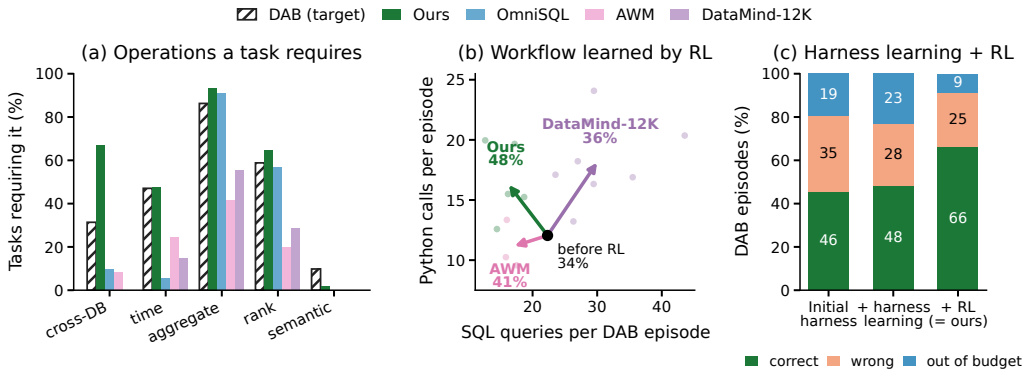


Figure 3: What the task generators produce and what RL learns from them. (a) Operations a correct solution requires. (b) Workflow of Qwen3.5-9B on DAB before and during RL on each source. (c) DAB outcomes along the two stages of DATAWORKLEARN.

database or table; the one gap is semantic judgement of text, needed by 10% of DAB questions but only 2% of our tasks. These differences decide what RL can learn (Figure 2c): our tasks raise DAB Pass@1 from 34.3 to **47.7**, AWM tasks to 40.7, DataMind-12K only to 36.1, although a stronger model wrote it, and OmniSQL tasks not at all (34.3). (Our run used 16 prompts $\times$ 16 rollouts per step, the others 32 $\times$ 8; the AWM and OmniSQL runs stopped after 15 and 20 steps.)

Further analysis reveals two insights. First, difficulty matters as much as composition: the frozen 27B agent always solves 74% of DataMind and 46% of OmniSQL tasks, which give GRPO no signal, whereas 55% of our unfiltered tasks are never solved and 36% have mixed outcomes. Second, the training data shapes the workflow the agent learns (Figure 3b): on our tasks it learns to retrieve once and compute in Python (22.3→15.9 SQL queries and 12.1→16.6 Python calls per DAB episode, averaged over checkpoints), while on DataMind's single-database tasks it issues more queries (30.4) and runs longer episodes without becoming more accurate.

5.4 RQ4: COMPLEMENTARITY OF HARNESS LEARNING AND RL

To see what each stage of DATAWORKLEARN contributes, we classify DAB episodes as correct, wrong or out of budget after each stage (Figure 3c) and compare which questions each stage improves.

We can see that the two stages fix different failures. Harness learning mainly turns wrong answers into exhausted budgets (wrong 35%→28%, out of budget 19%→23%, correct 46%→48%), and the RL stage then removes the budget failures (9%) and further reduces wrong answers (25%), for 66% correct. The stages also improve different questions: of the 22 DAB questions whose pass rate rises by more than 0.25, 16 improve only under RL, 4 only under harness learning, and 2 under both. This indicates that the stages are complementary: the harness changes how the agent works, and RL teaches it to finish that work correctly.

Case study. `PATENTS/2` asks for the year of peak filing EMA of every CPC group among German patents granted in late 2019. Before RL the agent never solves it: it filters free-text grant dates with SQL string patterns and misses whole CPC groups. After RL it solves 83% of samples with fewer calls (20 vs. 32): it retrieves the German 2019 subset once, normalises the dates and explodes the CPC list in Python, then aggregates. No generated task asks this question, but 27 of the 70 PATENTS tasks require parsing natural-language dates, 23 an EMA, and 13 exploding the CPC field.

5.5 RQ5: Evolution of Accuracy and Efficiency

Our RL reward does not favour short episodes: the only length-related term penalises exhausting the context. To see how accuracy and cost evolve together, we track DAB Pass@1 and context per episode over RL checkpoints and compare them with the harnesses proposed by the searches (Figure 4).

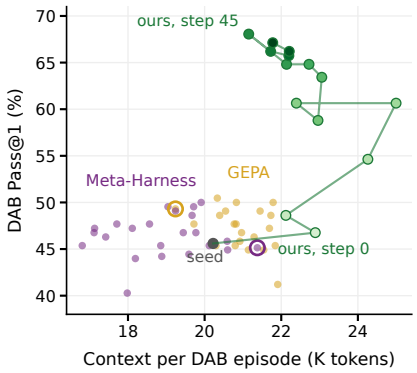


Figure 4: DAB Pass@1 vs. context per episode for RL checkpoints and search proposals.

We can see that DataWorkLearn becomes more accurate and cheaper at the same time: Pass@1 rises from 46.8 to 68.1 while context per episode falls from 22.9K to 21.2K tokens and model turns halve (24.3→12.6). The searches, in contrast, change cost but not accuracy: their proposals span 16.8K–21.9K tokens per episode, several cheaper than the seed's 20.2K, yet none leaves the 40–50% band.

The saving does not come from doing less: episodes that end correctly grow slightly (14.9K→17.0K) as harder questions become solvable. Instead, the agent learns to issue independent tool calls in the same turn and runs out of budget less often, and the training rollouts show the same trend (success 43%→83%, context 24.1K→21.2K). This suggests that efficiency emerges from RL on the generated tasks even without an explicit incentive.

6 Conclusion

We studied **workspace adaptation**, in which a general-purpose agent receives a data workspace and a brief description of its workload, but no downstream tasks, and must improve before the first question arrives. DataWorkLearn turns the workspace into executable practice: an author agent writes each task's solution first, executing it in the sandbox yields the reference outcome and the verifier, acceptance checks keep only tasks that reproduce and reject wrong solutions, and tasks the unadapted agent already solves are removed. Across four data-analysis and data-engineering benchmarks, workspace-adapted agents consistently outperform their unadapted counterparts, and on the same practice tasks, gains from updating the model's weights transfer to the benchmark while gains from optimizing the prompt, skills, or harness do not. A user's workspace is thus not only context available at inference time, but a source of experience from which an agent can learn to become a more effective coworker before it is asked anything.

AI Use Statement

Language models are part of our proposed method: they explore data environments and generate synthetic training tasks along with executable solutions and verification signals, as described in Section 3.2. Outside the method itself, we used AI tools to help write and debug code, search the literature, and refine experimental designs, and improve the wording of the manuscript. Generated tasks were kept only when they passed the automatic acceptance checks, and the authors reviewed all AI-assisted code, experiments, and text. We take full responsibility for the content of this paper.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems, 2026. URL https://arxiv.org/abs/2603.02766.

dbt Labs. ADE-Bench: Analytics and data engineering benchmark. GitHub repository, 2026. URL https://github.com/dbt-labs/ade-bench.

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

Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. Dabstep: Data agent benchmark for multi-step reasoning, 2025. URL https://arxiv.org/abs/2506.23719.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*, 2024.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, et al. Da-code: Agent data science code generation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 13487–13521, 2024.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo. Towards continual knowledge learning of language models. *arXiv preprint arXiv:2110.03215*, 2021.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Jiajie Jin, Zhao Yang, Wenle Liao, Yuyang Hu, Guanting Dong, Xiaoxi Li, Yutao Zhu, and Zhicheng Dou. Verigraph: Towards verifiable data-analytic agents, 2026. URL https://arxiv.org/abs/2606.16603.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026.

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

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=tmbOUyFx3R.

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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=XmProj9cPs.

Fangyu Lei, Jinxiang Meng, Yiming Huang, Yitong Zhang, Jianwen Luo, Xin Zou, Ruiyi Yang, Wenbo Shi, Yan Gao, Shizhu He, et al. Dacomp: Benchmarking data agents across the full data intelligence lifecycle. In *International Conference on Learning Representations*, volume 2026, pp. 104463–104501, 2026.

Boyan Li, Zhuowen Liang, Yupeng Xie, Xiaotian Lin, Tianqi Luo, Xinyu Liu, Yizhang Zhu, Zhangyang Peng, Yuan Li, Zhengxuan Zhang, et al. Dataspace: Benchmarking data agents for verifiable analytics over heterogeneous workspaces. *arXiv preprint arXiv:2608.03451*, 2026.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. Omnisql: Synthesizing high-quality text-to-sql data at scale, 2025. URL https://arxiv.org/abs/2503.02240.

Ruiying Ma, Shreya Shankar, Ruiqi Chen, Yiming Lin, Sepanta Zeighami, Rajoshi Ghosh, Abhinav Gupta, Anushrut Gupta, Tanmai Gopal, and Aditya G Parameswaran. Can ai agents answer your data questions? a benchmark for data agents. *arXiv preprint arXiv:2603.20576*, 2026.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, et al. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*, 2024.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling llm terminal capabilities, 2026. URL https://arxiv.org/abs/2602.21193.

Shuofei Qiao, Yanqiu Zhao, Zhisong Qiu, Xiaobin Wang, Jintian Zhang, Zhao Bin, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Scaling generalist data-analytic agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5PxFqpIYWC.

Gaurav Sahu, Abhay Puri, Juan A Rodriguez, Amirhossein Abaskohi, Mohammad Chegini, Alexandre Drouin, Perouz Taslakian, Valentina Zantedeschi, Alexandre Lacoste, David Vazquez, et al. Insightbench: Evaluating business analytics agents through multi-step insight generation. In *International Conference on Learning Representations*, volume 2025, pp. 4683–4715, 2025.

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.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Hongjin SU, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan O Arik. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=3UKOzGWCVY.

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

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, et al. Workspace-bench 1.0: Benchmarking ai agents on workspace tasks with large-scale file dependencies. *arXiv preprint arXiv:2605.03596*, 2026.

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

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

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OvLnVkV14P.

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

Bin Wu, Edgar Meij, and Emine Yilmaz. Beyond static toolsets: Self-evolving llm tool agents via continual documentation adaptation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21519–21539, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, volume 2025, pp. 79822–79843, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

John Yang, Kilian Lieret, Carlos Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. *Advances in Neural Information Processing Systems*, 38, 2026.

Zitong Yang, Neil Band, Shuangping Li, Emmanuel Candes, and Tatsunori Hashimoto. Synthetic continued pretraining. In *International Conference on Learning Representations*, volume 2025, pp. 44379–44421, 2025.

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

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Xuefei Cao, Lawrence Keunho Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason E Weston, Yu Su, and Yifan Wu. Agent learning via early experience. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=N3dXUHY5dD.

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

Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E Gonzalez. Raft: Adapting language model to domain specific rag. *arXiv preprint arXiv:2403.10131*, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *Advances in Neural Information Processing Systems*, 38:105816–105879, 2026.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo FR Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks. *arXiv preprint arXiv:2604.20087*, 2026.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Benchmark and Evaluation Details

DAB. The Data Agent Benchmark evaluates enterprise-style analytical questions over heterogeneous databases, including cross-database integration, irregular joins, unstructured fields, and domain-specific knowledge Ma et al. (2026). We use its 54-task challenging evaluation set and report verifier-based Pass@1 averaged over four independent runs.

DABstep. DABstep evaluates multi-step data-analysis questions over a payments workspace whose answers depend on rules stated in an accompanying manual Egg et al. (2025). We use the full 450-task split (72 easy, 378 hard) with the benchmark's reference agent and official scorer. Because gold answers are released only through the leaderboard, we score against an answer key reconstructed from publicly graded submissions, which reproduces the official verdicts on 99.96% of held-out submissions; 26 tasks whose reference cannot be recovered are excluded, leaving 71 easy and 353 hard tasks. We report easy- and hard-level accuracy, each an unweighted mean over the tasks of that level, from a single greedy run.

ADE-Bench. ADE-Bench evaluates analytics-engineering agents on executable edits to real dbt projects and databases dbt Labs (2026). Each task runs in an isolated environment and is checked by hidden SQL tests over the resulting project and database state. We evaluate on the 64-task suite and report test pass rate averaged over four runs.

DAComp-DE-Verified. DAComp spans the data-intelligence lifecycle. Its data-engineering tasks, which compose the subset of the benchmark DAComp-DE, require repository-level implementation or evolution of multi-layer SQL pipelines Lei et al. (2026). Several released gold projects did not achieve a perfect score under the original scorer. Additionally, we repaired reproducibility defects in the scoring and harness, accelerated execution by approximately $2.5\times$, and retained 43 reproducibly scored tasks: 13 implementation and 30 evolution tasks to create DAComp-DE-Verified

	DAB	DABstep	ADE-Bench	DAComp-DE-Verified
Environment seed	12 multi-database environments	1 payments workspace	5 dbt project environments	23 raw-data domains
Executable representation	Typed analytical DAG	Typed analytical DAG	Project edits plus independent oracle	Typed lineage DAG compiled to SQL
Scenario families	48 analytical families	45 analytical families	Repair, new model, refactor, data quality, incremental	Pipeline implementation plus seven evolution families
Author model	Qwen3.8-27B	Qwen3.8-27B	Qwen3.8-27B	Qwen3.8-27B
Difficulty screen	8 rollouts; mixed band plus 15% never-solved reserve	8 rollouts; keep 1–5 successes plus 38% never-solved reserve	4 rollouts; keep 1–3 successes[†]	8 rollouts; drop 8/8 or mean CFS > 80
Training tasks (9B)	397	607	300	384
Training tasks (27B)	521	626	296	268

Table 2: Environment and scenario instantiations of the shared generation pipeline. The same Qwen3.8-27B author is used throughout. [†]The ADE set contains 64 accepted tasks that were not difficulty-screened.

from DAComp-DE. Let $\mathcal{T}_\ell$ denote the expected output tables in layer ℓ, with normalized layer weights w_ℓ and normalized within-layer table weights $w_{t|\ell}$. Cascading Failure Score is

$$\mathrm{CFS} = 100 \sum_{\ell} w_\ell \sum_{t \in \mathcal{T}_\ell} w_{t|\ell} \mathbb{I}\big[\mathrm{match}(\hat{t}, t^\star)\big] \,, \tag{3}$$

where a table matches only if it exists and its row count and multiset of row hashes over the compared columns equal the gold table. This layer-weighted construction gives partial credit when upstream failures invalidate only part of a pipeline. We report **CFS** overall and for each task type, averaged over four runs.

Common protocol. Scores are reported on a 0–100 scale and higher is better. Before evaluation, task generation and adaptation stop and all agent components are frozen. The authoring and adaptation processes receive the data environments and broad task-family priors, but not the official questions, reference solutions, gold outputs, graders, evaluation feedback, or trajectories. Evaluation then uses each benchmark's official execution interface and evaluator.

B Learning-Experience Generation Details

This appendix records the benchmark-specific decisions omitted from the main text for space. The four generators share the same contract: an author receives a data environment and a broad task-family brief, but no downstream question or task-specific supervision; emits an instruction and executable solution representation; obtains supervision by executing that representation; passes positive and negative controls; and is finally screened against a frozen copy of the model to be trained. The generators retain each benchmark's native task format and execution interface rather than translating all environments into a common language.

B.1 DAB

We seed the generator with 12 multi-database environments and 48 analytical task families. Qwen3.8-27B authors a typed analytical DAG whose terminal relation defines the reference answer. The graph is executed twice and must produce identical intermediate hashes and final answers. A generated verifier must accept equivalent answer forms and reject automatically constructed answer mutants, including perturbed values, swapped fields, dropped or duplicated rows, and percentage-unit errors. Of 925 released tasks, 917 pass packaging checks. Eight frozen-policy rollouts per task yield 337 mixed-outcome tasks; we add a stratified reserve of 60 never-solved tasks for a final set of 397.

Evaluation is performed on the 54 official DAB questions, none of which is provided to the authoring or adaptation process.

DAB's negative controls operate on answers rather than independently authored programs. In addition, 31 final tasks contain labels produced by a semantic classification node. These labels are frozen during execution; the planned blind-consensus audit for this batch remains to be verified.

Baseline task generators on DAB. DataMind-12K is used as released. OmniSQL and AWM are run from their official code on the DAB databases with every model replaced by Qwen3.8-27B; AWM's verifier prompt embeds a dump of the database, which we truncate per table for databases that exceed the 27B context. OmniSQL questions do not prescribe an answer format, so, like DataMind's, they are scored against the reference answer by the same 27B judge; a strict programmatic check rejects many correct answers. For the 9B RL comparison, 308 tasks are sampled from each source.

B.2 DABstep

DABstep generation uses the single payments workspace and a family catalogue fixed before any task is written: 42 families in eight groups in the first round, extended in later rounds with list-shaped, compound rule-matching, mechanism, and counterfactual families, for 45 families in the final set. A family fixes an operator template, a sampling rule that draws every parameter from the data, an answer-format rule, and mutant recipes; Qwen3.8-27B completes the template into a typed analytical DAG whose terminal node defines the reference answer. The program is validated and executed twice from freshly loaded data with per-node content hashes compared, and its formatted result is the gold. The verifier is the benchmark's own scorer closed over that result; a task is released only if the scorer accepts at least two equivalent surface forms and rejects every mutant, including rescaled and mis-rounded values after a format-diversification pass that rewrote the guidelines into 68 distinct answer-format instructions. A candidate is also rejected when a shallower program yields the same answer or duplicates an earlier program signature. In the first round, 1,344 proposed programs yield 921 verified candidates, of which a checklist audit (the question asks for the quantity the final node computes rather than a related or rescaled one, comparisons and set differences run in the program's direction, it asks for a single thing rather than a name and a total, and every entity, period, category, and filter that scopes the computation is stated) of the question text removes 134 and two families are excluded, leaving 753. Eight sampled rollouts of the frozen policy through the evaluation interface band each task: tasks solved in one to five rollouts form the mixed band, tasks solved in six or more are dropped, and never-solved tasks form a reserve. The Qwen3.8-27B set holds 607 tasks (376 mixed, 231 never-solved) drawn from a pool of 1,031 verified tasks; for Qwen3.5-9B the same pool is re-banded against the 9B every five training steps (604–613 tasks per round). Evaluation uses the 450 official tasks, none of which is provided to the authoring or adaptation process.

B.3 ADE-Bench

ADE generation uses five dbt projects and five task families: repair, new model, refactor, data quality, and incremental transformation. Qwen3.8-27B authors each plan and GPT-5.6 Sol performs a separate review. Each plan includes a starter, a reference, an independently written equivalent, at least three mutants, and hidden singular tests derived from an independent oracle query. The reference is replayed twice through fresh environments and two grading harnesses; the equivalent must pass and every mutant must fail. Rejected plans receive concrete execution or review errors and may be repaired twice in new sessions. The final 300 tasks are selected from 1,360 candidates; 188 accepted tasks require at least one repair. Difficulty screening retains tasks solved in one to three of four frozen-policy attempts, although 64 later accepted tasks entered the final snapshot without this screen. Evaluation uses the 64 official tasks; their instructions, solutions, tests, and trajectories are not supplied to generation or adaptation.

B.4 DAComp-DE-Verified

DAComp-DE-Verified uses 23 domains from the verified evaluation environments and covers pipeline implementation plus seven evolution families. Qwen3.8-27B fills a typed lineage DAG; the compiler, rather than the author, emits SQL for the multi-layer pipeline. Two cold builds derive and replay the gold database. Automatic mutants alter the grain, filter, join type, window frame, dependency, or

schema; a retained mutant must build, score below 100 CFS, and reduce credit for its target table. The final set combines 141 retained tasks from the first generator version and 243 from the second, for 384 tasks. Filtering uses eight frozen-policy rollouts and removes tasks solved in every rollout or with mean CFS above 80. Because CFS supplies partial credit, difficult tasks remain trainable: 257 of the 384 retained tasks are never solved exactly before adaptation.

Replay for DAComp compares scores and column multisets rather than byte-level hashes. Columns that remain unstable across cold builds are excluded from grading. This benchmark-specific relaxation and the number of affected tasks will be reported with the final release artifacts.

B.5 Task-Instance Information Boundary

Our claim is task-instance separation, not distribution-free task discovery. Task-family briefs deliberately encode the intended workload: for example, whether the learner should answer cross-database analytical questions, repair a dbt model, or evolve a data pipeline. The author may also use schemas, samples, statistics, project files, and read-only environment tools. It does not receive the downstream question text, reference solution, gold output, official grader, evaluation feedback, or task trajectory. A separate leakage auditor may compare a generated candidate with official material, but that material is not placed in the authoring context or adaptation data.

The enforcement and audits are benchmark specific. ADE and DAComp-DE-Verified isolate official task files from the authoring container; DABstep points its training sandbox at a private asset root that holds only the seven workspace files, the generator's own task index, and its own answer key, so a worker cannot serve an official task even by accident, and a static guard keeps benchmark vocabulary out of the generator's code, with domain knowledge entering only as a hash-pinned input pack; DAB enforces the boundary through its authoring protocol and a post-hoc overlap audit. DAComp-DE-Verified compares generated text and graded structures with official tasks; DABstep compares every training question and the authored workspace notes with all official questions, finding no shared eight-word span, identical question, or task identifier, and no gold string in any training row; ADE removes exact normalized prompt matches. These checks support the narrower claim that downstream task content is not used as training data; they do not imply that generation is agnostic to the target task distribution.

Some benchmark-level priors are informed by aggregate observations. DAB's and DABstep's family catalogues include failure categories observed across benchmark runs, and DAComp-DE-Verified uses aggregate complexity targets. Such priors guide which kinds of tasks to synthesize but do not provide an official question, answer, or per-task learning signal to the author. We disclose this distinction because it is central to interpreting environment-seeded self-learning correctly.

C Training and Optimization Details

Shared configuration. The Qwen3.5-9B and Qwen3.8-27B runs use LoRA rank 32 and scaling factor 64 on every linear layer. The optimizer uses a learning rate of 5×10^{-5}, a constant schedule without warmup, weight decay 0.01, and gradient-norm clipping at 1.0. Each step samples 16 task prompts and performs one optimization epoch. Every prompt produces 16 independent rollouts with temperature 1.0 and top-p 1.0. The maximum prompt, response, and model-context lengths are 16,384, 114,688, and 131,072 tokens, respectively, with thinking mode disabled. Runs are configured for 100 steps, checkpoint and validate every five steps, and use a single node with eight NVIDIA B200 GPUs.

Training-only reward shaping. Reward shaping follows each environment's execution protocol and is removed when reporting downstream scores. DAB assigns -0.25 to an unparseable output, -0.05 per tool error or timeout, and a context-exhaustion penalty that ramps to -1. DABstep keeps the official scorer's binary verdict and, on successful episodes only, subtracts the larger of two penalties capped at 0.2: a turn penalty that ramps linearly from zero after the ninth of twelve training turns (the evaluation cap is ten) and a context penalty that ramps from zero once fewer than 4,096 response tokens remain. ADE-Bench applies no auxiliary shaping. DAComp-DE-Verified multiplies CFS$/100$ by a context factor whose penalty ramps to 0.6 near exhaustion.

Table 3: Per-benchmark GRPO configurations.

Setting	DAB	DABstep	ADE-Bench	DAComp-DE-Verified
Training tasks	397	607	300	384
Rollouts per prompt	16	16	16	16
PPO mini-batch (prompts)	4	4	8	4
KL loss to reference	Off	Off	0.01	0.01
Clip $(\varepsilon_{\text{low}}/\varepsilon_{\text{high}})$	0.20/0.28	0.20/0.28	0.20/0.20	0.20/0.20
Loss aggregation	Sequence mean	Sequence mean	Token mean	Token mean
Base reward	Binary answer	Binary answer	Test pass fraction	CFS$/100$
Validation decoding	Greedy, 4 samples	Greedy, 1 sample	T=0.6, p=0.95, 4 samples	Greedy, 4 samples
Reported checkpoint	Step 85	Step 70	Step 55	Step 50

Interaction and checkpoint protocol. DAB permits at most 200 agent iterations. ADE-Bench uses 150 turns during training and 150 at evaluation. DABstep uses 12 turns during training and 10 at evaluation, with a 300-second execution budget per step. DAComp-DE-Verified uses 1,500 seconds per training rollout and 3,600 seconds per evaluation rollout, with a 240-second tool-call limit and a maximum of 200 turns. The DAComp-DE-Verified run stopped after 63 steps and reports its step-50 checkpoint; DAB reports step 85; DABstep stopped after 79 steps and reports step 70. The same validation-based checkpoint-selection rule is used for the remaining runs.

D HarnessLab: Reflective Harness Search

To look inside a harness search, we built HarnessLab, a reflective search in which Qwen3.8-27B plays every role. An analyst reads the incumbent's failed trajectories on a search split of generated tasks and ranks failure modes; three engineers each edit the prompt, the tool implementations, or hooks in the control loop (after each turn, on tool errors, and before accepting an answer); a candidate replaces the incumbent only if it wins a paired comparison on the same tasks in the same time window ($z \geq 1.5$). Candidates are linted so that they cannot name tables, columns, or copy questions.

Over 12 iterations the agent proposed 35 candidates; most edits targeted the control loop (20) and the prompt (11), and three were accepted (Figure 5a, Table 4). The learned harness does not add tools; it adds *feedback* around the existing ones: a monitor that warns about repeated calls and low remaining context, a check that rejects a malformed answer once, a rule about the SQL dialect of the stock databases (DuckDB rejects per-branch `LIMIT` inside `UNION ALL`), and a recovery hint for queries that exceed the row limit, which steers the agent to aggregation rather than truncation. Each component was accepted on evidence from generated tasks (+16.0, +6.9 and +7.6 points on the search split, $z = 2.9$, 2.2 and 1.6), and together they raise held-out generated tasks from 37.5% to 54.2%. On DAB the successive incumbents score 44.9, 46.5, 53.2 and 45.8, all within one another's confidence intervals (Figure 5b): the components are sensible and partly workspace-specific, but on 54 real questions none of them changes accuracy measurably. Tightening the search does not change this picture: a variant that screens candidates on half of the search split and must confirm every acceptance on a disjoint validation split reaches its final harness with 717 episodes (a tenth of the cost), but that harness does not beat the seed on held-out generated tasks (33.9% vs. 37.5%).

E Reproducibility Checklist

For the final submission, we will release the authoring briefs, task schemas, environment inventories, accepted-task manifests, verifier tests, difficulty rollout summaries, and fixed random seeds. We will also report candidate counts at every gate, author and reviewer model versions, inference settings, repair budgets, training hyperparameters, hardware, and token/API accounting. Official task content is excluded from authoring and adaptation contexts and data; any access by leakage auditors is logged separately and used only to reject overlapping candidates.

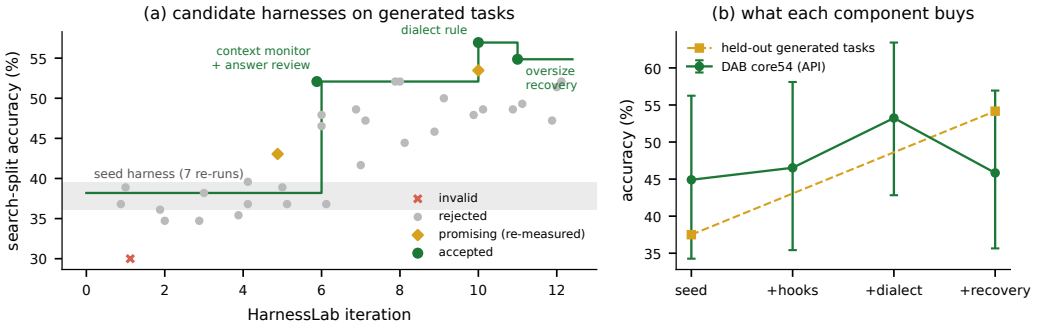


Figure 5: Harness learning with HARNESSLAB. (a) Search-split accuracy of every candidate harness (band: seven re-runs of the seed harness; line: accepted incumbent). (b) Accuracy of the successive incumbents on held-out generated tasks and on DAB (API serving, 54 × 8, 95% CIs).

Component (type)	What it does	Search Δ
Context monitor (loop hook)	warns when fewer than 15K tokens remain or calls repeat; asks for aggregation, reuse of stored results, or an answer	+16.0
Answer review (loop hook)	rejects an answer once if it is wrapped in JSON, adds % signs or field labels	
Dialect rule (prompt)	DuckDB: no per-branch `LIMIT`/`ORDER BY` in `UNION ALL`; use a `ROW_NUMBER` CTE	+6.9
Oversize recovery (tool-error hook)	on >1000-row results: aggregate in SQL or loop per entity in Python, never just add `LIMIT`	+7.6

Table 4: Components of the harness learned by HARNESSLAB, in order of acceptance, with the paired gain on the search split that admitted them (percentage points).