GDPEVO: EVALUATING AGENT SELF-EVOLUTION ON REAL BUSINESS TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent self-evolution updates an agent's persistent state from prior experience and reuses it to solve related tasks more effectively. Evaluating self-evolution is difficult: existing benchmarks provide limited coverage of economically valuable task domains; do not always design training and test tasks such that test-time gains can be attributed to training experience; and remain vulnerable to data contamination. We present *GDPevo*, an evolution-native benchmark grounded in GDP-related enterprise workflows, where an agent must learn each organization's hidden, undocumented rules on the job, together with the largely automated data pipeline that generates it. Its core mechanism, **rule hybridization**, decomposes each enterprise workflow into atomic business rules, distributes subsets of these rules across training tasks, and recombines them in held-out test tasks, so that test-time gains are attributable. *GDPevo* spans CRM, ERP, finance, healthcare, legal, and data-centric workflows. Its V1 release contains 120 tasks in 12 groups, with 5 training and 5 held-out test tasks per group; its automation lets the pipeline expand the suite to 240 tasks in 24 groups (V2) within two days, providing a practical response to contamination. Using *GDPevo*, we evaluate four agents (harness + model) under four supervision types. Self-evolution consistently improves held-out accuracy by up to 16.44 pp. Yet the best evolved agents remain far below the 91.6% accuracy achieved through fully informed human–AI collaboration, indicating that the self-evolution ability of current agents remains far from fully realized. We publicly release the pipeline, benchmark, and full evaluation results.

1 INTRODUCTION

In the AI era, once a task can be *automated* and *evaluated*, it is rarely far from being solved at scale. To bring these two capabilities to agentic task solving, agent self-evolution has emerged as a promising paradigm. Under this paradigm, an agent continuously improves task performance by updating its internal state, such as parametric states and non-parametric states (e.g., memory, skills, prompts, or harness code), based on prior interactions and reusing the resulting state in subsequent related tasks (Gao et al., 2026a). Automation includes techniques, such as continual learning (Wang et al., 2024b), experiential learning (Shinn et al., 2023; Zhao et al., 2024), and recursive self-improvement (Schmidhuber, 2007; Zhang et al., 2025), for performing these internal updates. Evaluation, on the other hand, determines whether these updates actually improve subsequent behavior. Although the literature now offers many techniques to automate self-evolution, evaluation methodology remains less mature.

We focus on evolution benchmarks that measure learning through an explicit train–test split[1], falling into two categories. First, evolution-native benchmarks are designed to test whether an agent learns from training experience and transfers what it learns to held-out tasks. They deliberately construct the train–test split so that the capabilities an agent acquires on the training tasks are genuinely required and exercised on the test tasks (Zhang et al., 2026; Jiang et al., 2026; Gao et al., 2026b; Huang et al., 2026). Second, evolution-adaptive benchmarks repurpose task suites originally designed for static, single-episode capability measurement, such as interactive reasoning environments and

[1]A separate family of benchmarks instead measures evolution over time, tracking how performance improves with accumulated experience within a single long-horizon deployment (e.g., EdgeBench (Zhu et al., 2026)) rather than across a train–test split.

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.

software-engineering tasks (Shinn et al., 2023; Zhao et al., 2024; Jimenez et al., 2024; Merrill et al., 2026). They introduce no deliberate design and directly split the suite into training and test sets, reporting whether the experience gained while solving the training set improves performance on the test set.

Existing benchmarks leave three limitations. First, evolution-native benchmarks provide limited coverage of difficult, economically valuable tasks found in domains such as finance, law, and healthcare (Patwardhan et al., 2025; Nandi et al., 2025; Li et al., 2026). Tasks such as invoice auditing are governed by business-specific rules that an agent must learn and follow, and their correctness can be verified against deterministic criteria, making them well suited for studying evolution. Such work also carries substantial economic value: adapting an agent to a new enterprise today relies on forward-deployed engineers (FDEs) who, like an intern, must embed themselves in the organization to learn its rules before becoming productive, so an agent that could instead learn a company's internal, undocumented conventions on its own would deliver an enormous economic payoff. Second, evolution-adaptive benchmarks provide broad and difficult tasks, but because their inherited splits of train–test sets were never aligned to any notion of transferable ability, an accuracy gain after evolution is not attributable. Even evolution-native benchmarks often begin with existing benchmarks and search for plausible relationships from which to form training and test splits (Zhang et al., 2026; Gao et al., 2026b; Huang et al., 2026). Because such relationships are extracted post hoc from existing benchmarks, they are inherently limited in number and diversity. Third, both categories of benchmarks release static and public task sets, leaving them exposed to data contamination (White et al., 2025; Wu et al., 2025; Chen et al., 2025). In an era of rapidly advancing AI, any benchmark without an explicit mechanism to counter contamination quickly loses its validity.

GDPevo addresses these limitations as an evolution-native benchmark. First, it is the first benchmark to evaluate self-evolution on GDP-related tasks, spanning CRM, ERP, finance, healthcare, legal, and data-centric domains. Second, we introduce **rule hybridization** to design the train–test relationship from the outset. The construction pipeline decomposes each domain workflow into atomic business rules, plants subsets of these rules across five training tasks, and recombines them across five held-out test tasks. An agent must infer reusable rules from the training sets and apply them compositionally at test time, so that test-time gains are attributable. Third, *GDPevo* uses an automatic construction pipeline (Figure 1) to expand the benchmark as public tasks become exposed. This automation lowers the marginal effort of generating new task groups and provides a practical response to contamination. *GDPevo* is therefore both a construction pipeline and the benchmark generated by that pipeline.

The first version (V1) of *GDPevo* contains 120 tasks in 12 task groups, with 5 training and 5 held-out test tasks per group (Figure 4). Thanks to this largely automated pipeline, we scale the benchmark to 240 tasks in 24 groups (V2) in just two days. The benchmark has three additional properties. First, every task uses a deterministic, rule-based grader instead of relying on an LLM as a judge. An LLM transforms independent rubric points into code-based test cases, which make scores reproducible and trace each failure to a violated rule. Second, *GDPevo* treats cost as a first-class metric. Tokens, agent turns, and monetary cost accompany accuracy because a useful evolution strategy should improve both task success and resource efficiency. Third, its reports provide per-group breakdowns, radar views, and transfer heatmaps, so users can trace aggregate gains to specific domains and examine how evolution on a source domain affects a target domain, since not all such transfer is beneficial.

Unless otherwise specified, all experiments in this paper use the combined V1 and V2 data. Evolution has two dimensions: the *supervision type* determines what signal the agent may draw on during training, and the *evolution method* determines how that signal becomes persistent state. We evaluate four agents (harness + model) under four supervision types: a no-evolution `base`; `fewshot`, which uses the training questions and their gold answers, akin to supervised fine-tuning; `reflect`, which uses the training questions and the scores of its own attempts rather than the gold answers, akin to reinforcement learning; and `self`, which uses only the training questions, akin to unsupervised learning. Throughout, the evolution method is skill-based. Our study is organized around three research questions (Section 4). **RQ1**: how much does each agent benefit from each supervision type, in terms of both accuracy and cost? We find that `fewshot` is the most reliable supervision type, that every evolved combination improves over its `base` (by up to 16.44 pp) and can even reduce test-time cost, yet all agents remain far below the 91.6% accuracy achieved by a fully informed human–AI collaboration reference. **RQ2**: how well do evolved skills transfer across task groups of different domains? We find that `fewshot`, like supervised fine-tuning, overfits its source group and can hurt on others, whereas `reflect`, like reinforcement learning, transfers more robustly. **RQ3**: which

matters more for evolution, the evolution method inside the harness, such as the skill creator, or the model itself? We find that even a minimal evolution method already works well, so the magnitude of evolution comes mainly from the model's intelligence.

The main contributions of this paper are as follows:

- We propose **rule hybridization**, which makes generalization from training to test concrete and test-time gains attributable.
- We conduct a comprehensive evaluation of four agents (harness + model) under four supervision types.
- We release the largely automated *GDPevo* generation pipeline and its generated benchmark (both the V1 and V2 versions).

2 Background and Related Work

This section positions *GDPevo* by reviewing which internal states agent self-evolution automatically updates and how these updates are evaluated.

2.1 Self-Evolving Agents

An agent's behavior can be viewed as sampling from $P(y \mid C, \theta)$, where y denotes the output, θ denotes the model parameters, and C denotes the context that conditions its behavior. Fine-tuning and reinforcement learning update θ to shift the output distribution toward desired behavior; when such updates continue across tasks, they represent parametric continual learning (Wang et al., 2024b). By contrast, prompt, context, and harness engineering modifies C to constrain the output distribution and increase the likelihood of task-specific responses. Here, C includes the prompt and persistent components such as memories, skills, and even the harness code. For example, Reflexion and ExpeL store feedback or reusable lessons in memory (Shinn et al., 2023; Zhao et al., 2024), while Voyager and SkillFlow build and maintain executable skill libraries (Wang et al., 2024a; Zhang et al., 2026). Other techniques update prompts or harness code. For example, GEPA evolves prompts through reflective feedback (Agrawal et al., 2025), and the Darwin Gödel Machine edits agent code and retains variants that improve benchmark performance (Zhang et al., 2025).

In this paper, our experiments focus on a non-parametric evolution method in the form of skills for ease of evaluation. However, *GDPevo* is agnostic to the form of persistent state being updated.

2.2 Evaluating Self-Evolution

Evolution benchmarks evaluate whether an agent can learn from prior experience and transfer what it learns to unseen, related tasks, and they fall into two categories. The first category does not define an explicit train–test split: EdgeBench (Zhu et al., 2026) fits scaling laws to an agent's learning curve over ultra-long-horizon deployment within a single environment, whereas RSIBench (Meng et al., 2026) evaluates the data-centric research loop that recursively improves a fixed target model. The second category explicitly defines a train–test split, which is the setting this paper focuses on. Within this setting, we review two types of benchmarks defined by how they construct the train–test relationship—**evolution-native** and **evolution-adaptive**.

2.2.1 Evolution-Native Benchmarks

The proper evaluation of self-evolution requires a carefully designed train–test relationship. If the training and test tasks are identical or nearly identical, improvements may reflect memorization rather than generalization. If they are unrelated, experience gained during training may provide no relevant support for test performance. These benchmarks must therefore use distinct but related tasks and deliberately construct the train–test split so that capabilities acquired during training are both required and exercised on held-out test tasks. For example, EvoAgentBench (Gao et al., 2026b) constructs ability graphs and a 528/267 train–test split in which every test task has verified procedural support from the training tasks. SkillFlow (Zhang et al., 2026) organizes 166 tasks into 20 workflow families to evaluate skill discovery, repair, and reuse across sequential tasks. SEA-Eval (Jiang et al., 2026)

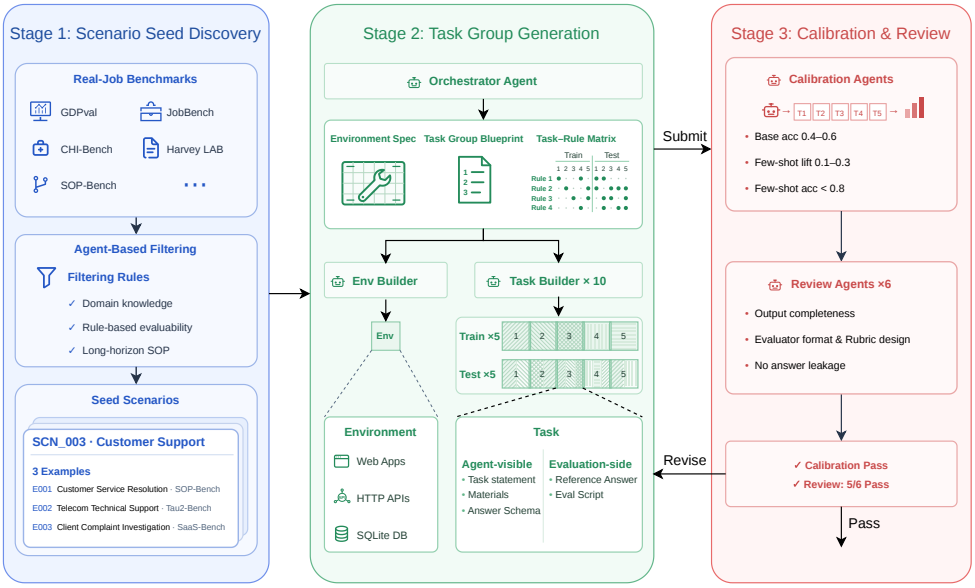


Figure 1: The *GDPevo* data construction pipeline. Every agent in this pipeline uses the same harness and model, Codex with GPT-5.5, but each runs as a separate instance with its own context, so the agents remain relatively independent.

measures both task success and token consumption along sequential tasks, capturing the effectiveness and efficiency of evolution.

However, existing evolution-native benchmarks provide limited coverage of more difficult and economically valuable enterprise workflows. Moreover, these benchmarks derive train–test relationships post hoc from existing benchmarks, limiting the diversity of transferable knowledge or procedures. In contrast, *GDPevo* targets difficult, economically valuable enterprise workflows and constructs train–test relationships from the ground up, using diverse business rules, enabling a broader range of knowledge and procedural transfer.

2.2.2 Evolution-Adaptive Benchmarks

Unlike evolution-native benchmarks, evolution-adaptive benchmarks do not explicitly construct train–test pairs around transferable abilities. Because their direct train–test splits are not aligned with any notion of transfer, performance gains after evolution cannot be reliably attributed to learning from prior experience. Nevertheless, by reusing a wide range of established task suites, these benchmarks offer greater task diversity and broader coverage of difficult problems. For example, Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2024) evaluate learning on repeated or partitioned tasks from ALFWorld, WebShop, HotpotQA, HumanEval, and related environments. Coding-agent systems commonly use SWE-bench (Jimenez et al., 2024), which contains 2,294 repository issues paired with executable tests, or realistic command-line suites such as Terminal-Bench (Merrill et al., 2026). The Darwin Gödel Machine (Zhang et al., 2025), for instance, selects self-modified coding agents based on their benchmark performance.

GDPevo is an evolution-native benchmark whose train–test relationships are explicitly constructed to evaluate transfer from experience. Furthermore, although *GDPevo* is built from scratch, its construction is largely automated and can produce a fresh set of data within two days, effectively resisting data contamination (White et al., 2025; Wu et al., 2025; Chen et al., 2025).

3 The *GDPevo* Benchmark

This section describes how the *GDPevo* pipeline constructs the benchmark; Section 4 describes how we evaluate self-evolution with the benchmark.

3.1 Largely Automated Construction Pipeline

GDPevo is constructed by agents in three stages, as shown in Figure 1; humans are involved only at the two ends of the pipeline: collecting the real-job benchmarks that kick off the process, and a final review of the generated tasks and rubrics. Because generation is always harder than verification, this review is simple and quick. Therefore, *GDPevo* can counter data contamination by rapidly regenerating a fresh version whenever a released one risks exposure. First, our initial release, V1, contains 120 tasks in 12 task groups. Each task group contains one shared business environment (e.g., a medical-records system or an employee database), 5 training tasks, and 5 held-out test tasks that use the same environment. V1 spans 4 groups in CRM, 4 in ERP, and 4 in finance (Figure 4). Second, in under two days the same pipeline produced V2, adding another 120 tasks in 12 task groups with the same 5+5 structure, spanning 4 groups in healthcare, 4 in legal, and 4 in data-centric work.

The next three subsections discuss these stages in turn.

3.2 Selecting economically valuable seed scenarios

Economically valuable enterprise work runs on undocumented, company-specific rules that a capable agent must learn on the job, much as a new hire does. To construct difficult and diverse tasks over such work, we draw from public benchmarks of real work, including GDPval (Patwardhan et al., 2025), SOP-Bench (Nandi et al., 2025), and JobBench (Li et al., 2026), and we expand these candidate benchmarks into concrete seed scenarios and their associated examples. For instance, in the customer-support scenario, we generate a customer-service-resolution example. This example is a real-life task in which we need to verify a customer's account, diagnose the reported service fault, carry out a repair, and record its final disposition. We later expand these examples into tasks, and each scenario becomes a task group of 10 tasks. For more detailed examples of seed scenarios, see Appendix B.

Agents expand scenarios and examples following three criteria. First, the examples must embed domain-relevant hidden rules. For instance, the customer-service-resolution example carries many enterprise-specific rules: which account states are eligible for support, which diagnostic readings count as a fault, and which faults must be escalated and to which team. Rule hybridization later recombines these rules across tasks (Section 3.3) so that an agent must genuinely learn them from the training tasks and generalize to the held-out test tasks. Second, the rules must be measurable, complete, and deterministic, so that task outcomes can be checked by a deterministic rule-based grader (Section 4.1). Third, the scenario must be complex enough to sustain a long-horizon standard operating procedure (SOP), and the scenarios must be diverse.

3.3 Separating training and test tasks through rule hybridization

To make agents genuinely learn transferable rules from the training tasks and apply them to the held-out test tasks, we introduce **rule hybridization**. First, for each scenario, the pipeline decomposes the business logic of its examples into **atomic rules**, which are minimal and independently checkable. These rules are deliberately absent from the model's world knowledge; they are internal conventions specific to a particular enterprise, such as one company's sponsor-status priority and another company's blacklist exclusion. Second, the pipeline scatters the rules across the 5 training tasks, so each training task exposes only a subset of these rules. Third, the pipeline recombines these rules across the 5 test tasks. Only an agent that has learned the rules during training and can compose them appropriately solves the test tasks.

Stage 2 of Figure 1 implements rule hybridization in two steps. First, an orchestrator agent writes a plan to construct a task group based on the scenario and examples: an environment specification, the 10 task statements, the atomic rules extracted from the examples, and a task–rule matrix that fixes which rules each training and test task exposes. Second, the orchestrator spawns one environment-building agent to implement the shared environment and 10 task-building agents that each construct a single task from its assigned rule subset.

3.4 CALIBRATION AND INDEPENDENT REVIEW

In the last stage of Figure 1, we ask a calibration agent to check each candidate task group against three requirements, all measured with a single reference agent (Codex with GPT-5.5 at `xhigh` reasoning effort). First, solving the 5 test tasks directly, without any training experience, should yield a score of roughly 40–60%; this band rejects task groups that are already solvable from the model's world knowledge alone as well as those that are too difficult. Second, when the agent first learns from the 5 training tasks and evolves before solving the test tasks—here we use `fewshot`, the evolution method that currently performs best (Section 4)—the test score must improve by about 0.1–0.3; this rejects task groups whose hidden rules either cannot be learned from training or are too easy to learn. Third, even after evolution the final score must stay below 0.8, so that measurable headroom remains. Although these thresholds are calibrated with a single reference agent, the resulting difficulty transfers across models: in the main evaluation, the `base` accuracies of all four evaluated models fall within the calibrated 40–60% band (43.58–50.63%), and none saturates after `fewshot` evolution, so the headroom we report reflects the tasks rather than the calibration model.

Finally, six independent reviewer agents inspect each candidate task group separately, and a group is accepted only if at least 5 of the 6 vote to accept it. They check three aspects. First, output completeness: builder agents may declare completion while leaving files, evaluators, or environments incomplete owing to model laziness (Liu et al., 2025), so reviewers verify that every required artifact is present. Second, evaluation format and rubric design: the rubrics should be well-formed and not overly similar to one another, presenting a degree of diversity that the reviewers agree on. Third, no answer leakage: when organizing a task, its answers must reside in a standalone, separate folder and must not appear anywhere else. Beyond passing this 5-of-6 agent vote, a task group is accepted only if a human expert also signs off on it.

4 EVALUATION

4.1 EVALUATION SETUP

Dataset. All experiments use the full *GDPevo* benchmark of 240 tasks in 24 task groups, combining V1 (CRM, ERP, and Finance) and V2 (healthcare, legal, and data-centric work).

Metrics. To make the evaluation deterministic, comprehensive, and interpretable, *GDPevo* provides three complementary forms of measurement. First, every task is scored by a deterministic, rule-based grader rather than an LLM judge. Each task comes with a set of weighted rubric points, but instead of asking an LLM to decide whether each point is satisfied, we ask an LLM to translate each rubric point into a code-based test case, which makes grading deterministic across runs. A task's score is the normalized sum of its passed point weights. Every test task is run three times, and all reported accuracies are means over those three runs. Second, *GDPevo* reports costs alongside accuracy. Each run records token consumption, agent turns, and monetary cost derived from model prices. Cost is a first-class metric because a good self-evolving agent should be not only accurate but also efficient. Third, *GDPevo* provides multiple views of the same results. Aggregate tables and radar maps compare different agents and supervision types; per-task score breakdowns reveal which rubric points are satisfied and, in turn, which hidden rules the agent discovered and which it missed; and transfer heatmaps expose which training experiences help—or hurt—which held-out groups.

Agents. We define an agent as Agent $=$ Harness $+$ Model. The harness determines tool use, context management, skill loading, and execution control, while the model supplies the underlying reasoning policy. We evaluate two harnesses (Codex and Claude Code[2]) and four models (GPT-5.5, Opus-4.8, GLM-5.2, and DeepSeek-V4-Pro-Preview).

Supervision types and evolution method. Evolution has two independent dimensions. The *supervision type* determines what signal the agent may draw on while working through the training tasks; the *evolution method* determines how that signal is turned into a persistent state that helps with later tasks.

[2]Codex: `https://openai.com/codex/`; Claude Code: `https://docs.anthropic.com/en/docs/claude-code/overview`.

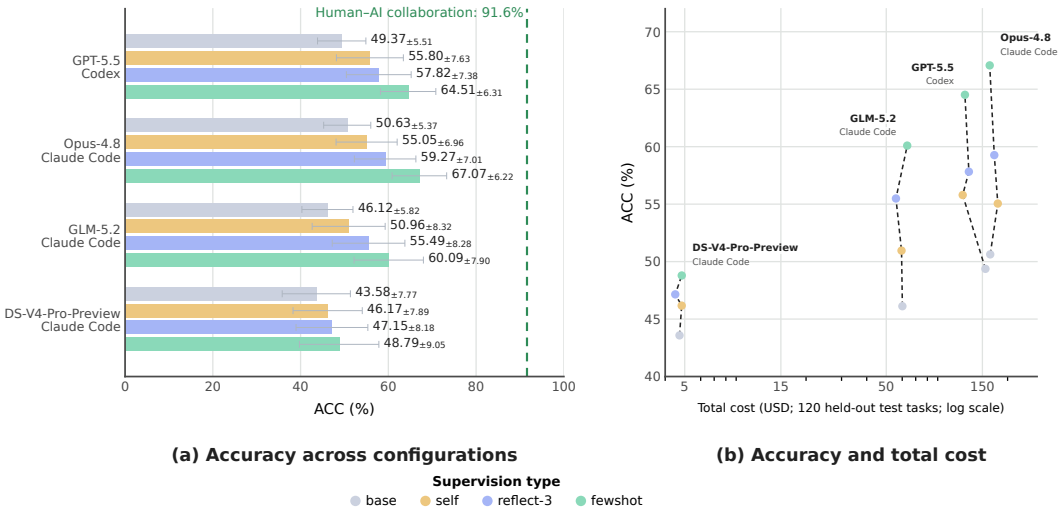


Figure 2: Accuracy–cost trade-offs on *GDPevo*. (a) Mean accuracy across task groups for each model–harness configuration and supervision type; each test task is averaged over three runs. Error bars show mean accuracy ± STD. The green dashed line marks the fully informed human–AI collaboration reference (91.6%). (b) Accuracy and total evaluation cost over all 120 held-out test tasks for each model–harness–supervision configuration. Colors denote supervision types, and dashed lines connect configurations sharing the same model and harness. Skill-generation overhead is excluded and can be found in Table 3.

We compare four supervision types—no training, supervised fine-tuning (SFT) (Brown et al., 2020), reinforcement learning (RL) (Shinn et al., 2023; Song et al., 2025), and unsupervised learning. We borrow these names from the standard setting in which the model weights are the persistent state being updated. First, in `base` (no training), the agent receives no training experience and directly attempts the test tasks. Second, in `fewshot` (SFT), the agent is given the questions, environment, and gold answers of the 5 training tasks; it reflects on how those gold answers are derived and distills the resulting experience into a persistent state, then attempts the test tasks with that state. Third, in `reflect` (RL), the agent receives the questions and environment of the 5 training tasks but not their gold answers, and attempts to solve them on its own; we then run our rule-based grader and return the score as feedback, from which the agent learns and revises its persistent state. This process repeats for three rounds (`reflect`-3). Unless noted otherwise, `reflect` refers to this three-round setting. Fourth, in `self` (unsupervised learning), the agent receives only the questions and environment of the 5 training tasks and attempts them with no answer-related supervision. It nonetheless can explore the environment—its database contents, which fields hint at the underlying rules, and how to query and operate it efficiently—so that at test time it can skip this exploration.

Self-evolution can update many kinds of persistent state, from model parameters to context, and, within context, memory, skills, or even the agent harness. *GDPevo* is agnostic to this choice and can evaluate any of these forms (Section 2). For ease of evaluation, all experiments in this paper adopt a skill-based evolution method, following recent evolution-native benchmarks such as EvoAgentBench (Gao et al., 2026b) and SkillFlow (Zhang et al., 2026). Our evolution state is thus a portable `SKILL.md` artifact. Within a skill-based method, the *skill creator* is the component that performs this distillation.

Environment. Every training and testing attempt executes in an independent Docker container with a dedicated working directory that contains only the files it needs; an attempt cannot read other attempts, source answers, evaluator files, or prior traces. Each raw trace, generated skill, answer, score, and metadata record is written beneath the attempt's own working directory.

4.2 RQ1: How Much Does Each Agent Benefit from Each Supervision Type?

Figure 2 and Tables 2–3 yield four findings. First, `fewshot` attains the highest accuracy for all four agents, and every agent surpasses its `base` when provided with any form of supervision, by

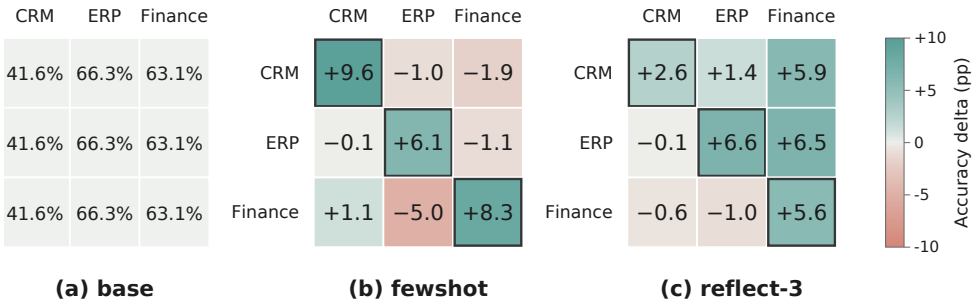


Figure 3: Cross-domain evolution transfer across CRM, ERP, and Finance. (a) reports target-group `base` accuracy. (b) and (c) show accuracy deltas using `fewshot` and `reflect-3`, respectively, relative to the corresponding `base` accuracy. Rows denote source groups used for training and columns denote target groups used for testing.

2.59 to 16.44 pp. Second, relative to model training, skill-based evolution offers high leverage: it achieves large accuracy gains at low cost. DeepSeek-V4-Pro-Preview `fewshot` attains accuracy comparable to GPT-5.5 `base` (48.79% versus 49.37%), and GLM-5.2 `fewshot` surpasses GPT-5.5 `base` and Opus 4.8 `base` by 10.72 and 9.46 pp, respectively. Third, evolution does not only improve accuracy; it can also make test-time execution cheaper. GPT-5.5 `fewshot` raises accuracy by 15.14 pp while reducing test-time cost by 20.88%; amortizing its one-time skill-generation cost (Table 3) over the five held-out tasks yields $1.296 per task, nearly identical to the $1.294 `base` cost. Opus 4.8, in contrast, achieves the largest accuracy gain with essentially unchanged test-time cost (−0.57%), yet its amortized end-to-end cost rises from $1.37 to $1.87 per task due to its costly skill generation. Fourth, a weaker starting point does not entail greater evolution headroom: DeepSeek-V4-Pro-Preview has the lowest `base` accuracy (43.58%) yet the smallest `fewshot` gain (+5.21 pp), whereas Opus-4.8 has the highest `base` accuracy (50.63%) and the largest gain (+16.44 pp).

To interpret these gains against an upper bound, we evaluate a fully informed human–AI collaboration reference. Because we lack domain experts across all covered fields, in each attempt a non-expert human operator works with the Codex harness and GPT-5.5 at `xhigh` reasoning effort, as a practical substitute for a fully informed expert. The pair receives the five training tasks' complete packages, including questions, gold answers, rubrics, notes, and evaluators that collectively expose the hidden rules, together with the current test input and access to the task environment. The current test answer and evaluator are withheld. Although the rules are available, the pair must still identify relevant records, determine how the rules apply, and produce a valid answer. This condition achieves 91.6% accuracy (the green dashed line in Figure 2; see Appendix D). This reference reflects how solvable the tasks become once the underlying rules are known, so it serves as a common, task-level ceiling for every evolved agent rather than a per-model bound. Domain experts may achieve higher accuracy, so the observed 91.6% should not be interpreted as the maximum attainable performance. Nevertheless, even this potentially conservative reference leaves substantial headroom above the best evolved agents, underscoring the limitations of current autonomous self-evolution.

The appendix provides further detail behind these aggregate numbers: the complete leaderboard and one-time skill-generation costs (Appendix C); case studies that trace individual rules through good and bad evolution cases, the latter covering both underfitting and overfitting (Appendix F); and rubric-level score breakdowns together with task-group radar charts (Appendix E).

4.3 RQ2: How Well Do Evolved Skills Transfer Across Domains?

The previous subsection shows that, within a single task group, an agent that learns from the 5 training tasks acquires within-domain knowledge that helps it solve the 5 held-out test tasks. A natural next question is whether training on a *different* domain still helps: if the 5 training tasks come from another domain, does the resulting experience (skills) benefit the target domain? To answer this question, we run a cross-domain transfer experiment. To keep it tractable, we restrict the experiment to the three V1 domains—CRM, ERP, and Finance—and randomly pick one task group from each, namely tg02 for CRM, tg06 for ERP, and tg10 for Finance; for the same reason we drop `self` and compare only `fewshot` and `reflect`. We then cross these three groups: an agent trains on one group and

Table 1: Controlled evolution-method comparison. We fix the agent and the supervision type (`fewshot`) and vary only the evolution method, which here means the skill creator. Every test task is run three times; each reported ACC is the mean over those three runs, macro-averaged over the 24 task groups, and Δ is the lift over the same agent's `base`.

	GPT-5.5 / Codex		DS-V4-Pro-Preview / Codex	
Evolution method	**ACC (%)**	**Δ (pp)**	**ACC (%)**	**Δ (pp)**
`base`	49.66	–	42.48	–
CC creator	62.15	+12.49	46.74	+4.26
Codex creator	62.19	+12.53	47.05	+4.57
DeepAgents creator	62.69	+13.03	47.75	+5.27
OpenCode creator	60.79	+11.13	47.84	+5.36
Naive creator	**65.12**	**+15.46**	**48.01**	**+5.54**

is tested on all three, yielding the 3×3 source–target matrix in Figure 3, where rows are the source (training) group and columns the target (test) group.

Figure 3 supports three findings. First, the diagonal cells—training and testing on the same group—are consistently positive, and under `fewshot` they are the largest gain in every row, matching the within-domain results reported in Section 4.2. Second, off the diagonal, `fewshot` behaves much like SFT: it tends to overfit the source domain, so transferring its generated skill across domains mostly fails to help and is often harmful, with five of the six off-diagonal cells negative and a worst case of -5.0 pp (training on Finance, testing on ERP). Third, `reflect` behaves much like RL, whose overfitting is milder: half of its off-diagonal cells are positive (up to $+6.5$ pp when transferring from ERP to Finance) and its worst case loses only 1.0 pp, so cross-domain evolution rarely hurts and sometimes still helps. Under `reflect`, however, the diagonal no longer dominates: evolving on CRM helps Finance ($+5.9$ pp) more than it helps CRM itself ($+2.6$ pp), and evolving on ERP helps Finance ($+6.5$ pp) nearly as much as ERP ($+6.6$ pp). On-policy feedback thus appears to yield more general skills, at the cost of some within-domain specialization.

4.4 RQ3: Which Matters More, the Evolution Method or the Model?

In the experiments so far, each agent evolves on its own: it starts from its own `base` accuracy and improves to its own `fewshot` accuracy. Here we instead fix a single agent—so there is a single `base` accuracy—and fix the supervision type to `fewshot`, leaving the evolution method as the only variable, to see how much the method affects the effectiveness of evolution. Since the method is skill-based throughout, varying it means varying the skill creator. We compare the five creators: (1) *Naive*, a minimal baseline we write ourselves—a single sentence that asks the agent to summarize experience from the available information and record it as a skill—together with the built-in skill creators of (2) Claude Code, (3) Codex, (4) OpenCode, and (5) deepagents. We experiment on two fixed agents, Codex with GPT-5.5 and Codex with DeepSeek-V4-Pro-Preview.

Table 1 supports two findings. First, a minimal skill-creator prompt may already suffice. Surprisingly, the *Naive* creator performs on par with, and often better than, the more elaborate off-the-shelf creators. Second, the harness can steer a model to evolve with high leverage—no parameter training required—but the *degree* of evolution is governed by the model's own intelligence rather than by the guidance encoded in the evolution method. With the model fixed, swapping the creator barely changes the size of the gain, and over-engineering it can even hurt.

5 Conclusion

We presented *GDPevo*, an evolution-native benchmark and largely automated pipeline for evaluating whether agents learn from prior experience and transfer that knowledge to new tasks. Its core mechanism, **rule hybridization**, makes generalization from training to test concrete and test-time gains attributable, and the pipeline can quickly regenerate the benchmark as a practical response to data contamination. We publicly release the pipeline, benchmark, and full evaluation results to support reproducible and economically grounded research on agent self-evolution.

AI USE STATEMENT

We use AI to build our benchmark data pipeline and to automatically produce data, as described in the main text. We also use AI to revise the grammar and wording of the already-written paper, and to make the figures more visually polished.

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, Alexandros G. Dimakis, Ion Stoica, Daniel Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. *CoRR*, abs/2507.19457, 2025.

Victor Barrès, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025. doi: 10.48550/ARXIV.2506.07982. URL https://dblp.org/rec/journals/corr/abs-2506-07982.

BIRD Team. LiveSQLBench: A dynamic and contamination-free benchmark for evaluating LLMs on real-world text-to-SQL tasks. https://github.com/bird-bench/livesqlbench, 2025.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Haolin Chen, Deon Metelski, Leon Qi, Tao Xia, Joonyul Lee, Steve Brown, Kevin Riley, Frank Wang, T. Y. Alvin Liu, Hank Capps, Zeyu Tang, Xiangchen Song, Lingjing Kong, Fan Feng, Tianyi Zeng, Zhiwei Liu, Zixian Ma, Hang Jiang, Fangli Geng, Yuan Yuan, Chenyu You, Qingsong Wen, Hua Wei, Yanjie Fu, Yue Zhao, Carl Yang, Biwei Huang, Kun Zhang, Caiming Xiong, Sanmi Koyejo, Eric P. Xing, Philip S. Yu, and Weiran Yao. CHI-Bench: Can AI agents automate end-to-end, long-horizon, policy-rich healthcare workflows? *CoRR*, abs/2605.16679, 2026. doi: 10.48550/ARXIV.2605.16679. URL https://dblp.org/rec/journals/corr/abs-2605-16679.

Simin Chen, Yiming Chen, Zexin Li, Yifan Jiang, Zhongwei Wan, Yixin He, Dezhi Ran, Tianle Gu, Haizhou Li, Tao Xie, and Baishakhi Ray. Benchmarking large language models under data contamination: A survey from static to dynamic evaluation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10080–10098, 2025.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026a.

Xingze Gao, Chuanrui Hu, Hongda Chen, Pengfei Yao, Zhao Wang, Yi Bai, Zhengwei Wu, Yunyun Han, Xiaofeng Cong, Jie Gui, Yafeng Deng, and Teng Li. EvoAgentBench: Benchmarking agent self-evolution via ability transfer. *CoRR*, abs/2607.05202, 2026b.

Harvey AI. Harvey LAB: The legal agent benchmark. GitHub repository, version 1.0, 2026. URL https://github.com/harveyai/harvey-labs/tree/v1.0.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. InfiAgent-DABench: Evaluating agents on data analysis tasks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 19544–19572, 2024. URL https://dblp.org/rec/conf/icml/HuZWCM0WSXZCY0K24.

Jiahao Huang, Fei Cheng, Junfeng Jiang, Zefan Yu, and Akiko Aizawa. BenchTrace: A benchmark for testing reflection ability and controlled evolution in LLM agents. *CoRR*, abs/2605.29225, 2026.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Özcan, Yannis Papakonstantinou, and Reynold Cheng. BIRD-INTERACT: Re-imagining text-to-SQL evaluation for large language models via lens of dynamic interactions. *CoRR*, abs/2510.05318, 2025. doi: 10.48550/ARXIV.2510.05318. URL https://dblp.org/rec/journals/corr/abs-2510-05318.

Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Shisong Chen, Jinghao Zhang, Tianjun Pan, Weijia Zhou, Jiaqing Liang, and Yanghua Xiao. SEA-Eval: A benchmark for evaluating self-evolving agents beyond episodic assessment. *CoRR*, abs/2604.08988, 2026.

Yixing Jiang, Kameron C. Black, Gloria Geng, Danny Park, Andrew Y. Ng, and Jonathan H. Chen. MedAgentBench: Dataset for benchmarking LLMs as agents in medical applications. *CoRR*, abs/2501.14654, 2025. doi: 10.48550/ARXIV.2501.14654. URL https://dblp.org/rec/journals/corr/abs-2501-14654.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Gyubok Lee, Elea Bach, Eric Yang, Tom J. Pollard, Alistair Johnson, Edward Choi, Yugang Jia, and Jong Ha Lee. FHIR-AgentBench: Benchmarking LLM agents for realistic interoperable EHR question answering. *CoRR*, abs/2509.19319, 2025. doi: 10.48550/ARXIV.2509.19319. URL https://dblp.org/rec/journals/corr/abs-2509-19319.

Yuetai Li, Yichen Feng, Zhangchen Xu, Zixian Ma, Kaiyuan Zheng, Fengqing Jiang, Xinghua Sun, Rulin Shao, Zichen Chen, Yue Huang, Xinyang Han, Brian Lee, Kayla Xu, Shenglai Zeng, Hang Hua, Xiangliang Zhang, Basel Alomair, Ranjay Krishna, Luke Zettlemoyer, Pang Wei Koh, Bhaskar Ramasubramanian, Luyao Niu, Xiang Yue, and Radha Poovendran. JobBench: Aligning agent work with human will. *CoRR*, 2026.

Yuchen Liu, Junhao Hu, Yingdi Shan, Ge Li, Yanzhen Zou, Yihong Dong, and Tao Xie. LLMigrate: Transforming "lazy" large language models into efficient source code migrators. *CoRR*, abs/2503.23791, 2025.

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. In *Advances in Neural Information Processing Systems*, 2024. URL https://dblp.org/rec/conf/nips/MaZZYZZLW024.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. RSIBench-data: Benchmarking data-centric research for recursive self-improvement. *CoRR*, abs/2607.25886, 2026.

Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, Estefany Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *CoRR*, abs/2601.11868, 2026.

Subhrangshu Nandi, Arghya Datta, Nikhil Vichare, Indranil Bhattacharya, Huzefa Raja, Jing Xu, Shayan Ray, Giuseppe Carenini, Abhi Srivastava, Aaron Chan, Man Ho Woo, Amar Kandola, Brandon Theresa, and Francesco Carbone. SOP-Bench: Complex industrial SOPs for evaluating LLM agents. *CoRR*, abs/2506.08119, 2025. doi: 10.48550/ARXIV.2506.08119. URL https://dblp.org/rec/journals/corr/abs-2506-08119.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. GDPval: Evaluating AI model performance on real-world economically valuable tasks. *CoRR*, 2025.

Jürgen Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial General Intelligence*, pp. 199–226. 2007.

Kean Shi, Zihang Li, Tianyi Ma, Zengji Tu, Jialong Wu, Xinbo Xu, Qingyao Yang, Ruoyu Wu, Weichu Xie, Ming Wu, Jason Zeng, Michael Heinrich, Elvis Zhang, Liang Chen, Kuan Li, and Baobao Chang. SaaS-Bench: Can computer-use agents leverage real-world SaaS to solve professional workflows? *CoRR*, abs/2605.15777, 2026. doi: 10.48550/ARXIV.2605.15777. URL https://dblp.org/rec/journals/corr/abs-2605-15777.

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*, volume 36, 2023.

Kefan Song, Amir Moeini, Peng Wang, Lei Gong, Rohan Chandra, Yanjun Qi, and Shangtong Zhang. Reward is enough: LLMs are in-context reinforcement learners. *CoRR*, abs/2506.06303, 2025.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. WorkBench: A benchmark dataset for agents in a realistic workplace setting. *CoRR*, abs/2405.00823, 2024. doi: 10.48550/ARXIV.2405.00823. URL https://dblp.org/rec/journals/corr/abs-2405-00823.

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.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46:5362–5383, 2024b.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, et al. LiveBench: A challenging, contamination-limited LLM benchmark. In *Proceedings of the 13th International Conference on Learning Representations*, 2025.

Xiaobao Wu, Liangming Pan, Yuxi Xie, Ruiwen Zhou, Shuai Zhao, Yubo Ma, Mingzhe Du, Rui Mao, Anh Tuan Luu, and William Yang Wang. AntiLeakBench: Preventing data contamination by automatically constructing benchmarks with updated real-world knowledge. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 18403–18419, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert T. Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. *CoRR*, abs/2505.22954, 2025.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qisheng Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. SkillFlow: Benchmarking lifelong skill discovery and evolution for autonomous agents. *CoRR*, abs/2604.17308, 2026.

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

Deyao Zhu, Xin Zhou, Shengling Qin, et al. EdgeBench: Unveiling scaling laws of learning from real-world environments. *CoRR*, abs/2607.05155, 2026.

A Reproducibility and Artifact

The full *GDPevo* pipeline, the 240-task benchmark, the rule-based graders, and the released evaluation runs are publicly available. Each task group ships its shared environment, 5 training tasks, 5 held-out test tasks, and per-task evaluation scripts whose rubrics enumerate the weighted scoring points described in Section 4.1. The evaluation workspace is natural-language-driven: an orchestration agent opens a folder of Markdown skills and guides and launches an experiment from a natural-language request. Every skill-generation run and test-solving attempt uses a dedicated Docker container, staged directory, harness home, and trace path.

B Task Group Details

Figure 4 organizes all 24 released task groups. V1 contains four CRM groups, four ERP groups, and four Finance groups; V2 adds four Healthcare groups, four Legal groups, and four Data Analysis groups. Each group contains 5 training and 5 held-out test tasks, for a total of 240 tasks.

As a representative V1 example, the CRM lead-capture group (tg01) hides atomic rules for sponsor-status priority, blacklist exclusion, contact deduplication and normalization, and follow-up scheduling. Its training tasks each exercise a subset of these rules, and its test tasks recombine them. The per-task rubrics make every atomic rule independently checkable, enabling the traceable and reproducible scoring used throughout the paper.



Figure 4: Hierarchical taxonomy of the released *GDPevo* benchmark: 240 tasks across 24 stateful task groups in six domains. The inner ring shows the macro domains, and the outer ring shows their task-group scenarios. Each task group shares one business environment, 5 training tasks, and 5 held-out test tasks, so that the benchmark can measure whether evolving from earlier tasks improves later work in the *same* environment. Every task is paired with a deterministic rule-based grader (Section 4.1).

Inside a task group. Figure 5 expands the CRM lead-capture example into training rules, a held-out task, and its deterministic grader. The three highlighted training tasks illustrate complementary rules; the `fewshot` learning agent uses all five training tasks to produce a reusable skill. The selected reference outputs illustrate how these rules combine in `test_004`. The grader checks the complete JSON answer, awarding each of seven rubric items its full weight or zero and normalizing the sum by the total weight of 14.

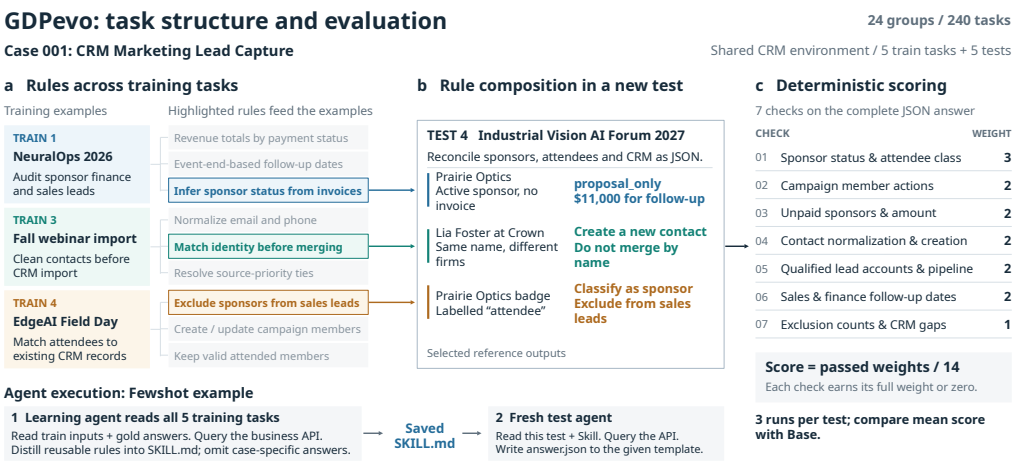


Figure 5: Task structure and evaluation in the CRM marketing lead-capture group (tg01). (a) Selected training tasks expose complementary business rules. (b) A held-out task combines these rules in a new instance; the displayed outputs are selected reference answers. (c) Seven weighted checks score the complete JSON answer. The bottom row illustrates `fewshot` execution: a learning agent distills the five training tasks into a skill, and a fresh test agent uses it to solve a held-out task in the shared business environment.

C Evaluation Results Details

Table 2 gives the complete numerical counterpart of the project website's leaderboard. It retains all four supervision types and reports the model–harness composition, reasoning setting, accuracy and uncertainty, lift over `base`, and the principal test-time efficiency measures.

Table 2: Full task-group 001–024 leaderboard corresponding to the current project website. Each test task is run three times. Accuracy is reported as mean accuracy ± STD. Lift is the absolute percentage-point change from the matching `base` row. Test-time cost and tokens exclude one-time skill generation.

			Accuracy		Test-time efficiency		
Model (setting) + Harness	**Supervision type**	**Evolution method**	**ACC ± STD**	**Lift (pp)**	**USD/task**	**Rounds**	**Tokens (k)**
• **GPT-5.5** (xhigh) Codex	`base`	–	49.37 ± 5.51	–	1.29	14.96	767.9
	`fewshot`	Skill Creator	64.51 ± 6.31	+15.14	1.02	12.04	543.9
	`self`	Skill Creator	55.80 ± 7.63	+6.42	1.00	11.45	487.3
	`reflect-3`	Skill Creator	57.82 ± 7.38	+8.45	1.07	11.85	526.8
• **Opus-4.8** (xhigh) Claude Code	`base`	–	50.63 ± 5.37	–	1.37	17.06	785.8
	`fewshot`	Skill Creator	67.07 ± 6.22	+16.44	1.36	14.46	819.1
	`self`	Skill Creator	55.05 ± 6.96	+4.42	1.49	15.57	964.0
	`reflect-3`	Skill Creator	59.27 ± 7.01	+8.64	1.43	15.58	938.2
• **GLM-5.2** (max) Claude Code	`base`	–	46.12 ± 5.82	–	0.50	22.94	1006.4
	`fewshot`	Skill Creator	60.09 ± 7.90	+13.97	0.53	22.63	1111.8
	`self`	Skill Creator	50.96 ± 8.32	+4.84	0.50	21.93	1037.8
	`reflect-3`	Skill Creator	55.49 ± 8.28	+9.37	0.47	20.65	952.1
• **DS-V4-Pro-Preview** (max) Claude Code	`base`	–	43.58 ± 7.77	–	0.039	15.65	776.0
	`fewshot`	Skill Creator	48.79 ± 9.05	+5.21	0.040	14.73	764.6
	`self`	Skill Creator	46.17 ± 7.89	+2.59	0.040	14.29	777.1
	`reflect-3`	Skill Creator	47.15 ± 8.18	+3.57	0.037	12.68	635.3

Table 3 separately reports the one-time cost of producing the skills used by the non-base supervision types. These costs are excluded from the test-time efficiency columns in Table 2.

Table 3: One-time skill-generation overhead per task group. Each value is averaged first over three independent generation attempts and then over task groups 001–024. Costs are in USD and tokens are in millions. A generated skill is reused for the five held-out tasks in its group, so the amortized per-test-task overhead is one fifth of each reported value. `base` has no skill-generation stage.

	fewshot		self		reflect-3	
Model + Harness	**USD**	**M tok.**	**USD**	**M tok.**	**USD**	**M tok.**
• **GPT-5.5** / Codex	1.36	1.02	1.17	0.82	3.08	2.61
• **Opus-4.8** / Claude Code	2.56	2.25	1.66	1.31	5.41	5.03
• **GLM-5.2** / Claude Code	0.45	1.11	0.38	0.95	2.16	6.04
• **DS-V4-Pro-Preview** / Claude Code	0.03	0.52	0.02	0.38	0.10	4.70

Machine-readable results. The released aggregates and pointers to the corresponding structured reports are maintained in the experiment board (identifying URL omitted for double-blind review); the report directories contain the task-group-level metrics and generated skill artifacts used to construct both tables.

D How Is the Human–AI Collaboration Reference Obtained?

We construct a *fully informed human–AI collaboration reference* by pairing the Codex harness and GPT-5.5 at `xhigh` reasoning effort with one non-expert human operator in each attempt. Multiple operators participate across the evaluation. The condition estimates performance when the evidence needed to recover the task-group rules is directly available, while the held-out task must still be completed in the benchmark environment. Distributing attempts across operators reduces dependence on the familiarity or workflow of any single person.

Each attempt is evaluated in a fresh context. The assigned operator and Codex receive the complete packages of the five corresponding training tasks, including their inputs, standard answers, notes, deterministic evaluators, rubrics, supporting files, and declared dependencies. They additionally receive the current test input and access to the running task environment. The current test answer, test notes and evaluator, other test tasks, environment source, task-group construction metadata, and records from previous attempts are not included. The protocol therefore provides the available training-side evidence without exposing the held-out solution, and it uses no separate training execution or skill-generation stage.

The five training packages share the current task-group environment and expose complementary parts of its hidden business logic. Standard answers demonstrate the expected output form; notes, rubrics, and evaluators specify the constraints that make those outputs valid; and supporting files and dependencies define the records and tools available in the environment. The assigned operator can inspect these materials while working with Codex on the held-out task. The condition is therefore more informative than the main evaluation conditions, but it does not reveal the target output or the grader used for the current test.

The evaluation covers all 24 task groups, with five held-out tasks per group and three independent attempts per task, for 360 attempts in total. Each attempt is scored by the same deterministic grader used in the main evaluation. Although the training-side evidence is available, each human–Codex pair must still find the relevant records, determine which demonstrated rules apply to the current instance, perform any required calculations, and produce a valid structured output. Under this protocol, the human–Codex condition achieves an accuracy of 91.6%.

E Breakdown and Diagnostic Views

Figure 6 exposes the rule-level structure behind the aggregate evaluation results. The paired views hold the model and harness fixed and show where `fewshot` changes the consistency with which individual rubrics are satisfied relative to `base`.

Figure 7 provides a complementary task-group view. For Opus-4.8 with Claude Code, `fewshot` raises macro accuracy from 50.63% to 67.07% and outperforms `base` on 23 of 24 task groups; `reflect` and `self` improve 20 and 16 groups, respectively, with smaller macro gains. Holding `fewshot` fixed, Opus-4.8 attains the highest macro average, followed by GPT-5.5, GLM-5.2, and DeepSeek-V4-Pro-Preview. The intersecting profiles show that these aggregate advantages are broadly distributed but not uniform across task groups.

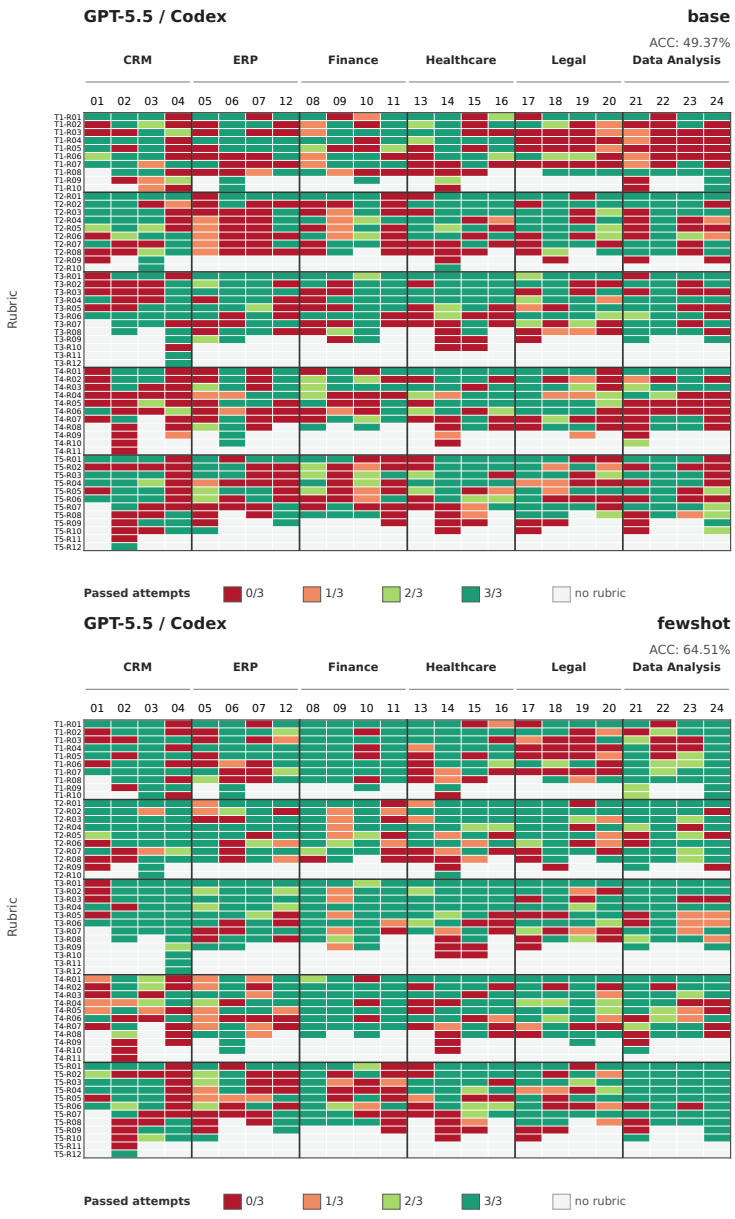


Figure 6: Rubric-level diagnostic views for GPT-5.5 with the Codex harness, comparing GPT-5.5 `base` (top) and GPT-5.5 `fewshot` (bottom) on task groups 001–024. Columns are task groups, arranged into six four-group domains; rows are test-local rubric indices and reset within each of the five held-out test tasks. Each colored cell reports how many of three independent attempts receive full credit on that binary rubric ($0/3$–$3/3$). Pale gray cells indicate that the corresponding test task has no rubric at that row, rather than a missing experimental result; all 24 task groups and all three attempts are present in both panels. Rubric weights are used to compute the reported accuracy, but color encodes the unweighted number of passing attempts so that individual rule reliability remains directly interpretable.

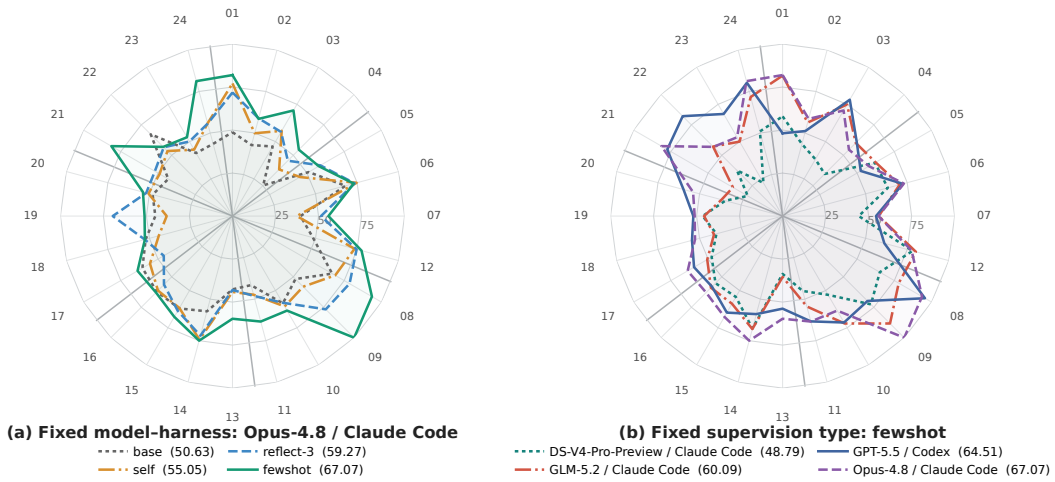


Figure 7: Task-group accuracy profiles on task groups 001–024. Panel (a) holds the model–harness configuration fixed at Opus-4.8 with Claude Code and compares the four supervision types. Panel (b) holds the supervision type fixed at `fewshot` and compares the four model–harness configurations. Each spoke reports task-group accuracy on a shared 0–100 scale, averaged over three runs per test task; parenthetical legend values are macro averages over the 24 task groups. Spokes proceed clockwise through CRM (001–004), ERP (005–007, 012), Finance (008–011), Healthcare (013–016), Legal (017–020), and Data Analysis (021–024).

F Evaluation Case Studies

Aggregate accuracy does not reveal what a generated skill learns or why it fails to transfer. We therefore examine four task groups using the generated skills, training traces, held-out solver traces, and deterministic rubric outputs. The cases cover two broad improvements and two forms of negative or limited transfer. Because they use different model–harness configurations, they are diagnostic examples rather than controlled comparisons between models. Table 4 summarizes the score patterns.

Table 4: Four diagnostic case studies. Values are held-out accuracy in percent, averaged over three runs per test task. The rows illustrate different transfer patterns and are not controlled model comparisons.

Task group	Model + Harness	Observed transfer pattern	`base`	`fewshot`	`self`	`reflect-3`
tg024	GPT-5.5 / Codex	Conditional operational rules	51.16	80.21	52.68	66.25
tg014	GLM-5.2 / Claude Code	Cross-workflow rule recombination	51.00	68.00	68.67	71.33
tg018	DeepSeek-V4-Pro-Preview / Claude Code	Dropped scope conditions	48.36	39.83	41.34	45.71
tg016	GLM-5.2 / Claude Code	Limited hidden rule structure	64.25	58.92	57.51	62.07

F.1 TG024: Broad Transfer of Operational Decision Rules

Task group 024 evaluates engineering-operations analysis with GPT-5.5 and the Codex harness. The shared environment contains work items, teams, owners, statuses, labels, SLAs, releases, milestones, blockers, and dependencies. The five held-out tasks ask the agent to construct portfolio summaries, audit SLA backlogs, assess release readiness, and prioritize overdue work. These tasks require more than aggregation: the solver must identify authoritative fields, exclude duplicate and cancelled records, classify work by its operational purpose, and distinguish release gates from ordinary unfinished items.

The score pattern indicates broad positive transfer. GPT-5.5 `fewshot` improves held-out accuracy from the GPT-5.5 `base` value of 51.16% to 80.21%, while GPT-5.5 `self` and GPT-5.5 `reflect` reach 52.68% and 66.25%, respectively. Its generated skill covers 33 of 35 training rules consistently and the remaining two partially. The improvement also spans different workflows rather than a single repeated template.

Two tests expose what transfers. In `test_001`, the solver must remove cancelled and duplicate candidates before computing the portfolio mix. The official population contains nine projects,

classified as 0 NewFeature, 3 TechDebt, 4 Reliability, and 2 Security projects. The skill records that current status, duplicate relations, and actual business purpose control inclusion and classification; legacy categories are only references. An incorrect population would change both the denominator and the investment recommendation.

In `test_003`, nine work items are unfinished, but only four contain high-impact blockers or incomplete critical dependencies. These four items are hard release gates; the other five require monitoring but do not independently prevent shipment. The learned rule therefore maps blocker evidence to a release action rather than treating every unfinished item as equivalent. The resulting decision is `NO_SHIP` because unresolved hard gates remain.

This case illustrates a transferable skill as a sequence of conditional decisions: validate the record, construct the population, classify by business meaning, calculate only after the population is fixed, and act on risks that can change the final decision. The skill also reduces search: GPT-5.5 `fewshot` averages approximately 15.53 tool calls per test, compared with about 30 for GPT-5.5 `base`. The gain therefore comes with a more selective evidence-gathering procedure, not additional exploration.

F.2 TG014: Recombining Rules across Insurance Workflows

Task group 014 evaluates healthcare insurance operations with GLM-5.2 and Claude Code. The tasks cover therapy authorization, drug-denial appeals, workers-compensation claim recalculation, physician peer-to-peer review, and a mixed work queue. The agent must both choose an operational action and provide an audit trail that distinguishes supporting, missing, and excluded records. All three supervision types improve over `base` at 51.00%: `fewshot`, `self`, and `reflect` reach 68.00%, 68.67%, and 71.33%, respectively.

One transferable rule concerns the granularity of missing evidence. In `train_002`, a broad packet item describes formulary-failure evidence, but the gold output names the exact records that remain absent. In `test_002`, prior therapies are already documented for the Dupixent appeal, so the insurance appeal no longer lacks medication evidence. Household income proof remains missing only for the parallel manufacturer-assistance workflow. A skill that repeats a broad packet label, or treats the assistance gap as an appeal blocker, produces a plausible route but an incorrect audit trail.

A second rule concerns evidentiary grounding. In `train_004`, a narrative note that PET imaging "may be better" is insufficient; the output must cite structured records for each required PET-over-SPECT factor. In `test_004`, new peer-to-peer materials support overturning the denial because they fill those factor-level requirements, not because they contain a generally favorable narrative. Similarly, `train_003` teaches that a claim adjustment uses the current effective benchmark and computes each line as the current allowed amount minus the amount already paid. The same operation is reused in `test_003` and inside the mixed queue of `test_005`.

The held-out tasks thus recombine rules learned from different training workflows: name the exact missing record, keep parallel programs separate, ground authorization decisions in structured factors, and recalculate payments against the current benchmark. These rules are not tied to one patient, drug, or claim line. They describe how the workflow should respond when the entities change, which explains why all three generated skills outperform `base`.

F.3 TG018: Negative Transfer from Dropped Scope Conditions

Task group 018 evaluates post-hearing court administration with DeepSeek-V4-Pro-Preview and Claude Code. The solver reconciles case-system records, hearing notes, clerk memoranda, and financial materials; determines whether a matter is disposed, dismissed, continued, or pending; and then calculates charges, payment schedules, docket actions, and forms. Jurisdiction, processing date, petition type, and case status act as scope conditions that can change several downstream outputs at once.

The generated skills do not transfer reliably. `base` reaches 48.36%, while `fewshot`, `self`, and `reflect` reach 39.83%, 41.34%, and 45.71%. For `fewshot`, the gap is particularly diagnostic: training replay reaches 73.88%, but held-out accuracy falls to 39.83%. The skill has learned regularities that fit the training cases, but it often omits the conditions under which those regularities apply.

Several learned statements turn local observations into global policies. A source that is authoritative for identity is treated as authoritative for every fact; fees absent from several training answers become a fixed "never charge" list; dates separated by roughly 30 days become a default deadline; and reporting more conflicts is treated as safer. Each heuristic can match the training jurisdictions while failing after the jurisdiction, policy date, or case status changes.

The propagation is visible in `test_004`. Case `LC-25-0331` is continued and has no final order, so it cannot be closed or charged in advance. Treating it as final changes the disposition, financial entry, system action, and document together. The expected output contains exactly seven material conflicts, whereas the `fewshot` skill encourages the solver to report nine. On this test, DeepSeek-V4-Pro-Preview `fewshot` scores 17.78%, while DeepSeek-V4-Pro-Preview `base` scores 40.00%. The stored skill is therefore not merely incomplete; it actively directs the solver toward decisions that are invalid in the current case.

This case separates rule content from rule scope. A useful court-administration skill must state which source controls each question, which jurisdiction and effective date select a fee schedule, and which case status permits closure. Remembering the action without these conditions produces confident negative transfer.

F.4 TG016: Limited Transfer from Narrow Clinical Rules

Task group 016 evaluates clinical triage and care coordination with GLM-5.2 and Claude Code. The solver retrieves current patient facts, reads a runtime protocol, and fills a structured disposition containing risk, medication, tests, follow-up time, and evidence identifiers. Unlike the preceding cases, many decisive clinical thresholds are already visible in the runtime protocol. The training tasks therefore expose fewer hidden rules that can add value at test time. `base` reaches 64.25%, while `fewshot`, `self`, and `reflect` reach 58.92%, 57.51%, and 62.07%.

Hypokalemia provides the clearest example. In `train_003`, potassium is 3.2 without arrhythmic symptoms, so the routine oral-repletion branch schedules follow-up at 08:00. In `test_003`, the latest final potassium value is 2.8 and arrhythmia symptoms are present. The runtime protocol now selects urgent escalation, with clinician notification, EKG, telemetry or emergency department evaluation, and follow-up at 18:00. Reusing the learned 08:00 value preserves the output but loses the branch condition that made it correct.

The respiratory workflow exposes a second limitation. A training answer teaches the exact recommended-test set for one pneumonia scenario, which can help keep the structured output precise. It does not determine the emergency disposition, risk label, current-record selection, or evidence identifiers in a new case; those decisions follow from the visible protocol and current patient records. The learned rule is locally valid but covers only a small part of the held-out scoring surface.

Negative transfer in this case therefore does not require a broadly incorrect medical rule. It can arise when the task offers little hidden reusable structure and the generated skill preserves a narrow value without its protocol branch. When the current protocol already supplies the decisive rule, an additional skill must remain subordinate to that evidence rather than override it.

F.5 Cross-Case Synthesis

The four cases distinguish three requirements for reliable skill transfer. First, the training tasks must expose reusable structure. TG024 and TG014 contain operational rules that recur after the entities and evidence change; TG016 places more of its decisive structure directly in the runtime protocol. Second, a generated skill must preserve scope. TG018 drops jurisdiction and case-status conditions, while TG016 drops a clinical branch condition. Third, the skill must connect evidence to the outputs it controls. The successful cases specify which fields determine population membership, classification, evidence placement, payment correction, or release action.

These patterns cannot be attributed solely to the task or model because the cases use different model–harness configurations. They nevertheless suggest a common audit for generated skills: each rule should identify its triggering evidence, the output fields it changes, and the conditions under which it stops applying. A skill that records all three behaves as a reusable decision rule; a skill that records only a surface value or recurring action is more likely to overfit or interfere with the current evidence.

G LESSONS FROM AUTOMATED SELF-EVOLUTION BENCHMARK GENERATION

Constructing the 24 task groups in *GDPevo* exposed recurring failure modes in automated benchmark generation. We summarize eight practical lessons under four themes: task construction, leakage control, grading and calibration, and evaluation integrity. Together, these lessons describe when an automatically generated task suite can support the claim that training experience improves held-out behavior.

G.1 DESIGNING TRAIN–TEST RELATIONSHIPS

Train–test relationships must be designed from the outset. A self-evolution benchmark must construct a learning problem, not merely a collection of difficult tasks. In *GDPevo*, each task group contains five training tasks and five held-out test tasks in one shared business environment. The two sets share enterprise-specific rules, but differ in their entities, evidence, noise, and rule combinations. Rule hybridization creates this relationship by distributing atomic rules across training tasks and recombining them in test tasks.

Both extremes lead to misleading results. If training tasks directly demonstrate the test procedure, improvement may reflect template reuse. If training and test are unrelated, a lack of improvement does not show that the agent cannot evolve; it shows that the benchmark supplied no transferable experience. We therefore specify which test scoring points have training anchors, what knowledge transfers, and what changes at test time.

Generation and review should be independent. When one agent generates the environment, tasks, answers, graders, and review rationale in the same context, these artifacts may share the same assumptions and mistakes. A prompt may unintentionally echo the answer, or several rubric points may reward the same decision. *GDPevo* separates task-group design, environment construction, task construction, calibration, and final review. Independent reviewers inspect the completed group rather than accepting the builder's own assessment. This separation reduces builder–grader coupling and makes missing or inconsistent artifacts easier to detect.

G.2 CONTROLLING THE INFORMATION AVAILABLE DURING EVOLUTION

Isolation is more reliable than prompting. Asking an agent not to inspect hidden material is not a stable experimental control. For every attempt, we create a minimal workspace containing only the files allowed by the current stage and run the agent in an isolated container. Standard answers outside supervised training, notes, evaluators, unrelated tasks, and prior attempts are not mounted. The business environment runs separately and is accessible only through the allowed endpoints. Thus, agents with different file-exploration behavior still receive the same information.

Information access must match the declared supervision type. The four supervision types share the same task group, environment, grader, test protocol, and evolution method, but expose different training signals. `base` receives no training experience; `fewshot` receives the training inputs and gold answers; `self` receives only the training inputs; and `reflect` receives the training inputs together with a fixed budget of feedback on its own training attempts. These information boundaries must hold throughout the evaluation. If `self` can access a training answer, or if `reflect` can query the grader during testing, the configuration receives stronger supervision than declared and no longer measures the intended supervision type. A benchmark should therefore specify and audit the files, endpoints, feedback, and query budget available at every stage.

G.3 VALIDATING GRADING AND CALIBRATION

Rubric points should be deterministic, binary, and semantically distinct. A deterministic grader is reproducible, but reproducibility alone does not guarantee a meaningful score. Each scoring point in *GDPevo* represents a distinct business outcome and receives a raw weight from $\{1, 2, 3\}$. It earns either its full normalized weight or zero; we do not award partial credit within a point. The task-factory prompt additionally requires 6–10 scoring points spanning at least four semantically

distinct business outcomes and prohibits rewarding the same criterion, answer fact, or root decision more than once under different wording.

Together, these restrictions prevent a task-building agent from adjusting the score distribution through arbitrary subchecks, duplicate criteria, denominators, or tolerances. A submission may still receive a score between zero and one by passing a subset of independent points, but no individual point contributes a builder-chosen fraction. We validate the rubric by modifying one business outcome at a time and checking that the intended point changes while unrelated points remain unchanged.

Calibration must consider both difficulty and evolution gain. Reasonable `base` accuracy is necessary but not sufficient. A task group may be difficult because information is missing or because its test tasks depend on rules that never appear in training. Conversely, near-perfect post-evolution accuracy may indicate that training and test are too similar. We therefore calibrate both initial difficulty and moderate, non-saturating improvement after training. When a group fails calibration, we revise its task design, evidence, or train–test relationship rather than changing rubric fractions to force the score into a target range. Calibration runs use fresh agent processes that do not inherit the builder's context. These runs also use a single reference model, so the calibrated difficulty band is model-specific in principle; empirically, however, the `base` accuracies of all evaluated models land within the target band and none saturates after evolution, indicating that the calibrated difficulty transfers across models.

G.4 Keeping Evaluation Comparable and Auditable

The evaluation agent must be capable and remain fixed. We adopt an *AI-evaluates-AI* protocol in which a separate evaluation agent organizes the assessment of each evaluated agent. It stages files, launches containers, pairs generated skills with test attempts, invokes deterministic graders, handles failures, and aggregates results. The evaluation agent does not assign semantic scores itself, but its orchestration decisions can still affect the measured outcome. In particular, a less capable model in this role may mis-stage files, mismatch skills and attempts, fail to recover interrupted containers, or aggregate outputs incorrectly, causing evaluation failures unrelated to the capability being measured. We therefore use a sufficiently capable evaluation agent and keep its model, version, reasoning setting, prompt, tool permissions, retry policy, and aggregation rules fixed across comparable model–harness combinations. Changing the evaluated agent should not simultaneously change the agent that organizes the experiment.

Traces and costs should be preserved. Every evolution and test attempt receives a unique run identifier, an independent workspace, and a dedicated raw trace. A test process solves only one task, and each non-base test attempt uses the skill produced by the corresponding evolution attempt. Failed or contaminated attempts are retained for audit and replaced under the original protocol rather than silently scored as zero or dropped.

We also report skill-generation cost separately from test-time cost. The former measures the investment required to acquire a reusable skill; the latter measures the efficiency of applying it. Keeping the two stages separate makes it possible to determine whether a supervision type pays a larger one-time cost in exchange for repeated downstream savings.

In summary, automated self-evolution benchmark generation produces more than a task set. It also requires a designed train–test relationship, isolated information budgets, deterministic graders, calibration evidence, independent review, and attempt-level traces. These controls distinguish transferable improvement from answer leakage, template reuse, scoring artifacts, and execution failures.

H Use of Public Source Benchmarks

We use public benchmarks only as sources for seed scenario discovery. The source pool spans professional deliverables and operating procedures, workplace software and customer-support workflows, healthcare and legal operations, and data-centric tasks. Table 5 lists the 15 public benchmarks represented in the released task groups and the type of seed each source contributes.

Table 5: Public benchmarks used during seed scenario discovery. Source examples motivate the work setting, artifacts, and constraints; *GDPevo* constructs new environments, train–test tasks, reference answers, and graders.

Public source benchmark	**Seed focus**
GDPval (Patwardhan et al., 2025)	Economically valuable professional deliverables
SOP-Bench (Nandi et al., 2025)	Standard operating procedures and rule-governed workflows
SaaS-Bench (Shi et al., 2026)	Multi-application business, healthcare, and software workflows
JobBench (Li et al., 2026)	Occupation-specific professional tasks
Terminal-Bench (Merrill et al., 2026)	Command-line data preparation and analysis
τ^2-Bench (Barrès et al., 2025)	Tool-mediated customer-support workflows
CHI-Bench (Chen et al., 2026)	Healthcare administration and insurance workflows
MedAgentBench (Jiang et al., 2025)	Electronic-health-record operations
FHIR-AgentBench (Lee et al., 2025)	FHIR-based clinical record retrieval
Harvey LAB (Harvey AI, 2026)	Legal investigation and transactional workflows
SpreadsheetBench (Ma et al., 2024)	Spreadsheet data preparation and quality control
InfiAgent-DABench (Hu et al., 2024)	Tabular data analysis and cleaning
BIRD-INTERACT (Huo et al., 2025)	Interactive analytical SQL tasks
LiveSQLBench (BIRD Team, 2025)	Stateful SQL and CRUD workflows
WorkBench (Styles et al., 2024)	Database-backed workplace analytics

For each scenario, the discovery stage selects a small number of source examples and abstracts the work setting, artifacts, and operational constraints that make the scenario professionally meaningful. A scenario may combine examples from multiple benchmarks. These examples are not reused as evaluation instances: the downstream pipeline constructs a new shared environment, five training tasks, five held-out test tasks, reference answers, and deterministic graders. The source benchmarks therefore determine scenario coverage and realism, while the evaluated instances and train–test relationships are newly constructed for *GDPevo*.