DATAPRAXIS: SCALING ANALYTICS TASKS VIA VERIFIED WORKSPACE EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

General-purpose agents are increasingly expected to perform professional analytics work, which requires sustained reasoning, iterative investigation, and reliable evidence. Training these capabilities requires diverse and challenging tasks. However, generating tasks directly from raw data often overlooks deeper questions that emerge through analysis and provides limited evidence for verification. To address these limitations, we introduce DataPraxis, an analytics task synthesis framework based on verified workspace evolution. The framework progressively enriches workspaces with validated findings and artifacts, then inherits and combines these workspaces to uncover deeper questions. It constructs tasks from accumulated findings and their dependencies, verifying analytical results both during evolution and after task assembly. Using this framework, we construct DataPraxis-5K, a dataset of 5,000 verified trajectories spanning diverse domains and output formats, to train DataPraxis-27B and DataPraxis-35B-A3B. DataPraxis-27B achieves 55.11 on DABStep and 51.9 on JobBench, matching or surpassing frontier models such as GLM-5.2. We will release DataPraxis-5K to the community for further study.

1 INTRODUCTION

As large language models acquire capabilities for code execution, tool use, and multi-turn interaction, general-purpose agents are beginning to undertake professional work in real environments (Yao et al., 2023; Wang et al., 2025). Analysis is a representative example: users expect models not only to answer questions, but also to understand the materials in a working directory, conduct analyses autonomously, and deliver well-supported conclusions and files (Hong et al., 2024; Zhang et al., 2025a). CLI agents such as Claude Code (Anthropic), Codex (OpenAI), and Pi Agent (Zechner & contributors) provide a general-purpose execution interface for such work. However, training models to reliably complete these long-horizon analytical tasks still requires training tasks that cover complex analytical processes and provide reliable evaluation signals. Constructing such tasks at scale is a key challenge in improving general-purpose agents' professional analysis capabilities.

Existing agent task synthesis methods broadly fall into two categories. The first generates and combines tasks based on data descriptions and predefined structures (Qiao et al., 2026; Xie et al., 2025; Shi et al., 2025). These methods guide task generation and increase complexity, but may struggle to discover tasks that emerge only through actual analysis. The second constructs tasks from interaction experience and execution evidence obtained through environment exploration (Mai et al., 2025; Chen et al., 2026). These methods ground tasks in interaction evidence, but offer limited support for discovering deeper tasks that require further computation on prior results or integration of findings across analytical branches. Discovering deeper questions that are not apparent from raw materials through continued analysis therefore remains a key challenge in professional data analysis task synthesis. Meanwhile, intermediate findings are not inherently reliable, and errors can propagate when these findings are reused. Even when local findings are confirmed, the complete tasks constructed from them may still lack reliable evaluation criteria. Ensuring both the trustworthy accumulation of analytical results and reliable evaluation of the resulting tasks is another key challenge.

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.

To this end, we propose DataPraxis, a professional analytical task synthesis framework based on verified workspace evolution. Its central idea is that analytical results are not only answers to current questions, but also a basis for discovering subsequent ones. DataPraxis retains scripts, intermediate data, and analytical findings produced during execution in executable workspaces, enabling subsequent analysis to inherit prior results, explore new directions, and combine different branches for joint analysis. Figure 1 illustrates the role of workspace evolution in task discovery: without continued workspace evolution, task generation cannot fully exploit progressively accumulated analytical results; by inheriting and combining verified workspaces, DataPraxis enriches the basis for proposing questions, progressively revealing deeper questions that are difficult to discover directly. The framework then constructs tasks backward from the resulting findings and their dependencies, removing intermediate support from the evolution process so that solvers must independently reconstruct the analysis from raw materials, thereby converting accumulated analytical dependencies into challenging long-horizon training tasks. To ensure reliability from analytical discovery to task construction, DataPraxis introduces dual-evidence verification, combining verified findings from workspace evolution with independent solution trajectories to construct and validate evaluation rubrics. These rubrics assess final deliverables and filter solution trajectories for training.

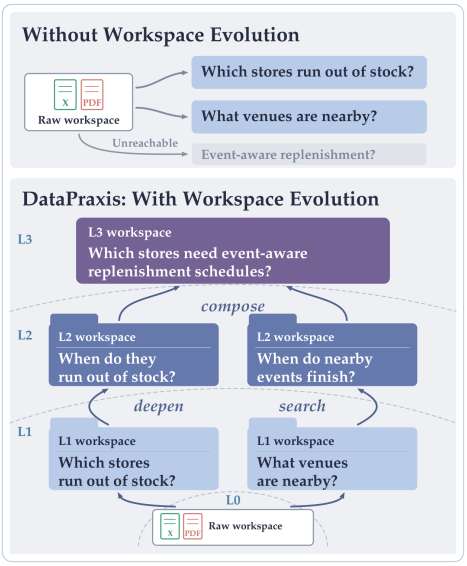


Figure 1: **Up:** Tasks are generated from the initial materials without workspace evolution. **Down:** DataPraxis inherits and combines verified analytical results through workspace evolution, providing new grounds for discovering deeper questions.

Our main contributions are as follows:

- We propose DataPraxis, a data analysis task synthesis framework based on workspace evolution. Verified workspace evolution and dual-evidence verification enable the scalable construction of challenging, verifiable terminal-agent tasks.
- We construct DataPraxis-5K, containing 5K long-horizon professional analytical training examples across 15 domains, including business, finance, sports, and entertainment. We train DataPraxis-27B, a 27B dense model, and DataPraxis-35B-A3B, a 35B mixture-of-experts model with 3B active parameters.
- We demonstrate that workspace-evolution-based synthesis improves agents' ability to perform data analysis work. DataPraxis-27B achieves leading performance among the evaluated models on data analysis and long-horizon professional tasks.

2 Motivation

Let $X_0 = (\mathcal{D}, \mathcal{E})$ denote a seed workspace containing data assets $\mathcal{D}$ and an execution environment $\mathcal{E}$. DataPraxis constructs a task instruction q and a verifier V from this workspace:

$$X_0 \longrightarrow (q, V). \tag{1}$$

The instruction q specifies the analytical objective and delivery requirements for an agent operating in X_0. The verifier V evaluates the analytical correctness, supporting evidence, and overall quality of the agent's final output. The challenge is to formulate deep analytical questions from the initial workspace, where many patterns and relationships have yet to be uncovered. Findings uncovered during analysis can motivate questions that are not apparent from the initial workspace alone.

To examine how analysis informs subsequent questions, an agent investigates three successive questions in each of 50 workspaces, using findings from one analysis to formulate the next question.

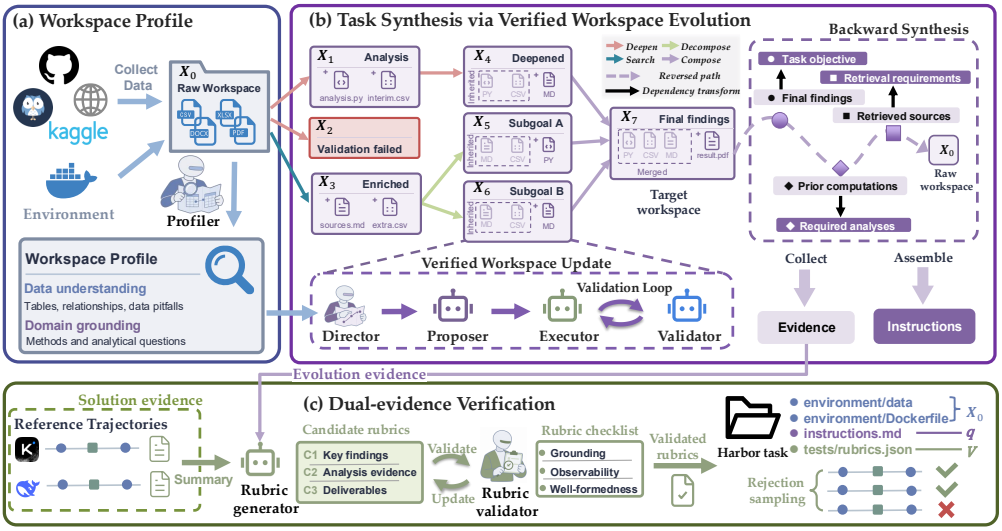


Figure 2: The DataPraxis framework. (a) The Profiler characterizes the initial data and domain. (b) Verified workspace evolution progressively discovers and validates analytical findings, which are converted into standalone tasks through backward synthesis. (c) Dual-evidence verification combines evidence from evolution with independent solving trajectories.

We use Qwen3.6-27B to measure the perplexity of each proposed question given either the initial workspace alone or the initial workspace together with findings from earlier analyses. Figure 3 shows that mean perplexity given only the initial workspace rises from 16.87 to 31.07, whereas including prior findings reduces it by 46.1% and 53.0% in rounds 2 and 3, respectively. These observations suggest that continued analysis gives rise to questions that are less apparent from the initial workspace, while accumulated findings provide useful context for formulating them.

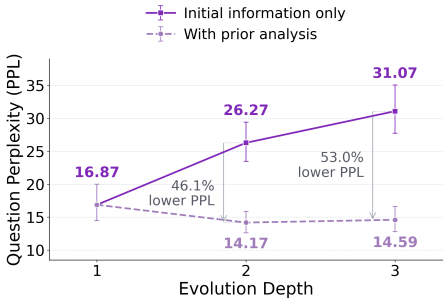


Figure 3: Question perplexity with initial workspace information alone or with accumulated findings.

Motivated by this observation, DataPraxis evolves a workspace by updating its state with verified results from each analysis. A workspace state captures the data, scripts, and analytical results available to an agent within its execution environment at a given stage. Each update carries forward prior analytical assets and incorporates new findings, providing the basis for subsequent analysis. For each evolution path, DataPraxis traces the accumulated findings and their dependencies backward to synthesize one task:

$$X_0 \xrightarrow{\text{Evolve}} X_1 \xrightarrow{\text{Evolve}} \cdots \xrightarrow{\text{Evolve}} X_m \xrightarrow{\text{Synthesis}} (q, V), \tag{2}$$

where m is the number of evolution steps and $X_1, \ldots, X_m$ are the successive workspace states. The q requires a single agent to reconstruct the analysis from X_0 within one session, without access to scripts, intermediate artifacts, or conclusions produced during evolution.

3 Method

3.1 Workspace Initialization and Profiling

As shown in Figure 2(a), we collect raw datasets containing files in diverse formats from Kaggle, DataWorld, and GitHub repositories to construct initial workspace. Filtering by relational structure and size yields 500 data collections spanning 15 domains, including business, government, human

resources, and others. Each collection is paired with a shared execution environment $\mathcal{E}$, including dependencies such as pandas and NumPy, to form a seed workspace X_0.

A common approach to task synthesis uses predefined task categories as seeds to guide generation (Qiao et al., 2026; Zhang et al., 2025a). However, a fixed taxonomy limits the range of questions considered, while model preferences may further concentrate generation on familiar categories. DataPraxis therefore derives seeds from each workspace's data and domain properties. A Profiler Agent analyzes $\mathcal{D}$ to construct a *Workspace Profile*, capturing relationships among tables, data pitfalls, and field semantics, alongside relevant analytical methods and common domain questions. This profile connects the available data with domain problems, guiding the generation of diverse analytical objectives suited to each workspace.

3.2 Task Synthesis via Verified Workspace Evolution

Building on the workspace profile, DataPraxis constructs task instructions through progressive workspace evolution followed by backward synthesis. As shown in Figure 2(b), evolution strategy selection determines which states to inherit and how to advance the analysis. Verified workspace evolution then updates these states through execution and verification, providing a basis for subsequent objectives. Finally, backward task synthesis converts the accumulated analytical dependencies into task instructions, withholding intermediate support to require independent completion.

3.2.1 Evolution Strategy

Starting from a workspace with no predefined task, DataPraxis uses a director agent to decide where analysis should proceed. The director considers the workspace profile and findings verified during earlier evolution steps to select eligible source workspaces. It proposes an analytical question grounded in the available data and distinct from earlier analyses. When extending an existing workspace, the director identifies the specific prior finding on which the new question depends. It then selects an evolution mode according to how the question draws on the chosen workspaces. Table 1 summarizes the four modes.

Table 1: Workspace evolution modes. The mapping $s \rightarrow t$ denotes s source workspaces and t analytical directions, with $k \geq 2$.

Mode	Sources $\rightarrow$ Tasks	Description
`deepen`	$1 \rightarrow 1$	Extend existing findings through further analysis.
`search`	$1 \rightarrow 1$	Retrieve and verify relevant external materials.
`decompose`	$1 \rightarrow k$	Split a compound objective into independent subtasks.
`compose`	$k \rightarrow 1$	Merge verified findings from complementary workspaces.

Both `deepen` and `search` start from a single source workspace. The former evolves the workspace by investigating existing findings in greater depth, while the latter expands its contents by retrieving and filtering relevant external materials. The other two modes support workspace branching and merging. Starting from a single workspace, `decompose` divides a compound objective into complementary subtasks, each driving the evolution of an isolated workspace. Under `compose`, complementary verified workspaces are merged after conflicts among their contents have been resolved. The merged workspace then evolves through an integrated analysis that requires findings from multiple sources. Together, these modes produce diverse workspace evolution paths, enabling the synthesis of tasks with different analytical dependencies.

3.2.2 Verified Workspace Evolution

Each evolution step builds on inherited workspace states through workspace preparation, analytical modification, and verification, coordinated by four agents: director, proposer, executor, and validator. The director uses the workspace profile and previously verified findings to identify an analytical question that is grounded in the data and distinct from earlier analyses. It selects the source workspaces and an evolution mode based on what the question requires. When extending an existing workspace, the director identifies the specific prior finding on which the new analysis depends. The

proposer then prepares the inherited workspace state by organizing its data and analytical assets. When combining multiple sources, it removes duplicate materials and resolves file conflicts. The proposer formulates a local analytical objective and explicit output requirements from the prepared state. The executor updates the workspace to address this objective through reasoning and tool use. It incorporates relevant materials, creates or revises scripts, and produces artifacts and findings.

Before this state can support further evolution, the validator reviews the changes in the workspace. It checks whether data dependencies, metric definitions, and analytical methods are appropriate. It also assesses whether evidence supports the conclusions and whether scripts reproduce the reported results. If errors are found, the executor revises the affected workspace contents and submits the changes for another review. This cycle continues until no errors are identified or the review limit is reached. Updates that pass verification are retained as new workspace states, while those with unresolved errors are discarded. The accepted states can then serve as sources for another evolution step, where their findings inform new objectives and their analytical assets support execution. Repeating this procedure produces multiple evolution paths, preserving verified findings, supporting materials, and their dependencies for backward task synthesis and subsequent collection of evolution evidence.

3.2.3 Backward Task Synthesis

Backward synthesis converts verified analyses from workspace evolution into standalone tasks and collects evolution evidence for subsequent dual-evidence verification. DataPraxis first selects a target evolved workspace and uses its findings to define the task's objective. It traces the evolution path backward to the initial workspace X_0, converting dependencies on prior findings into explicit task requirements. For example, `search` nodes yield retrieval instructions, while computations at `deepen` nodes become prerequisite analysis steps. The task instruction q combines these requirements and specifies a deliverable format suited to the domain, such as JSON, Markdown, or a spreadsheet. During this backward traversal, DataPraxis collects evolution evidence: the verified findings and supporting materials accumulated along the evolution path. This evidence is passed to the rubric generator and combined with solution evidence from independent reference trajectories to ground task-specific evaluation criteria. Given q and X_0, the solver independently plans the full workflow, reconstructs the supporting evidence, and produces the requested deliverable within one session. This consolidates analyses previously distributed across multiple agents into one task, placing greater demands on independent planning and sustained reasoning. Appendix B.1 illustrates how workspace evolution and composition lead to a standalone task through a hotel pricing example.

3.3 Dual-evidence Verification and Rejection Sampling

Backward synthesis turns verified analyses into task instructions, but evidence from workspace evolution alone does not directly establish criteria for evaluating independently produced deliverables: a solver may follow a different analytical path. DataPraxis therefore uses dual-evidence verification to construct and validate task-specific evaluation rubrics from two evidence sources, as shown in Figure 2(c). Evolution evidence consists of verified findings and supporting materials collected along the workspace evolution path during backward synthesis. Solution evidence is extracted from reference trajectories produced by two models that independently solve the synthesized task from the initial workspace X_0 in a sandbox. The rubric generator combines both sources to define criteria for conclusions, analytical methods, and delivery requirements. The rubric validator then checks whether each rubric is grounded in the data, assessable from the final deliverable, and explicit about what constitutes a satisfactory result. Rubrics that fail these checks are revised and revalidated, or removed if they remain invalid. Appendix B.2 presents a case study of rubric synthesis and validation.

DataPraxis packages the synthesized instruction, initial workspace, execution environment, and validated rubrics into a Harbor task. We uses the rubrics to select teacher trajectories through rejection sampling. It assesses the final deliverables of candidate teacher trajectories against the rubrics constructed and validated using evolution evidence and solution evidence, retaining only runs that pass task-level verification. The final training dataset contains the accepted trajectories, any continuations after context compaction, and subagent trajectories from those runs.

Table 2: Performance on data analysis benchmarks. Scores marked with † are taken from the DABStep leaderboard.

Model / data	DABStep			JobBench	RideBench	Avg
	Easy	Hard	Overall			
Claude Opus 5	87.50†	48.68†	54.89	69.50	82.02	68.80
Kimi-K3	84.72	69.58	72.00	57.20	79.78	69.66
GLM-5.2	87.50	49.47	55.55	51.20	78.65	61.80
DeepSeek-V4-Pro	84.72	44.71	51.11	49.20	75.28	58.53
GPT-5.6-Terra	83.33	45.50	51.55	33.80	68.54	51.30
Gemini-3.7-Flash	90.28	53.70	59.55	57.90	77.53	64.99
Gemini-3.6-Flash	91.67	41.01	49.12	43.70	79.78	57.53
Qwen3.6-27B						
Base	84.72	26.46	35.78	38.50	70.79	48.36
DataMind-5K	86.11	38.36	46.19	37.90	69.66	51.25
DataPraxis-5K	83.33	49.74	55.11	51.90	74.16	60.39
Qwen3.6-35B-A3B						
Base	81.94	27.78	36.45	31.30	59.55	42.43
DataMind-5K	84.72	29.63	38.44	26.60	59.55	41.53
DataPraxis-5K	81.94	47.09	52.67	41.00	71.91	55.19

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and harness. We evaluate whether DataPraxis improves data analysis agents built on Qwen3.6-27B (Qwen Team, 2026a) and Qwen3.6-35B-A3B (Qwen Team, 2026b). We use Claude Code (Anthropic) as the harness for both trajectory collection and evaluation. The models trained on DataPraxis-5K are denoted DataPraxis-27B and DataPraxis-35B-A3B, respectively.

Benchmarks. We evaluate the effectiveness of DataPraxis training data on three benchmarks: (1) DABStep (Egg et al., 2025) measures data analysis reasoning across multiple steps; (2) JobBench (Li et al., 2026a) assesses agents' ability to analyze heterogeneous data and produce professional deliverables across occupations; (3) RideBench is a benchmark developed internally by a company for business analysis in ride-hailing scenarios, with construction details in Appendix A. We conduct all benchmark evaluations in Harbor sandboxes and use Gemini-3.7-Flash as the judge model.

Baselines. We compare DataPraxis-27B and DataPraxis-35B-A3B against 7 leading frontier models from the Claude (Anthropic, 2026), Kimi (Moonshot AI, 2026), GLM (Z.ai, 2026), DeepSeek (DeepSeek, 2026), GPT (OpenAI, 2026), and Gemini (Doshi, 2026a;b) families. For each backbone, we also compare against the original model and a counterpart trained on 5K DataMind examples (Qiao et al., 2026). To match our agent framework and training set size, we regenerate the DataMind solution trajectories using GLM-5.2 in Claude Code.

Implementation and training. We use GLM-5.2 as the backbone model. For dual-evidence verification, DeepSeek-V4-Pro and Kimi-K3 provide reference trajectories. We construct a training set of 5K trajectories, including continuations after context compaction and subagent trajectories. The agent prompts in DataPraxis are provided in Appendix E. We use LlamaFactory (Zheng et al., 2024) for supervised fine-tuning. We train the base models for 5 epochs, using a learning rate of 1×10^{-6}, a batch size of 32, and a maximum sequence length of 128K tokens.

4.2 MAIN RESULTS

Table 2 reports consistent improvements across both backbones and all three benchmarks after training on DataPraxis. The average benchmark score rises from 48.36 to 60.39 for Qwen3.6-27B and

Table 3: Ablation results for Qwen3.6-27B. DataPraxis-5K uses 5K training examples with verification. DABStep reports overall scores.

Setting	DABStep	JobBench
DataPraxis-5K	**55.11**	**51.90**
DataPraxis-2K	48.89	45.50
DataPraxis-4K	54.45	48.80
Shallow-2K	53.11	46.50
Deep-2K	51.33	48.80
w/o dual-evidence verification	50.00	47.50

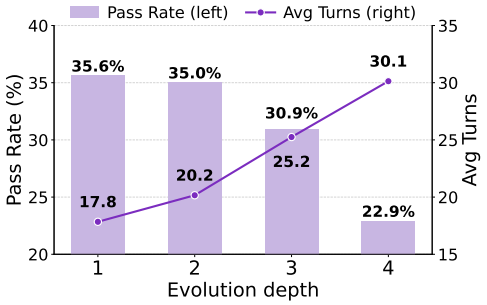


Figure 4: Average pass rate and tool-call turns across evolution depths.

from 42.43 to 55.19 for Qwen3.6-35B-A3B. DABStep Hard shows particularly large gains in pass rate, at 23.28 and 19.31 percentage points, respectively, suggesting improved reasoning across multiple analytical steps. For the challenging workplace tasks in JobBench, the corresponding score increases are 13.40 and 9.70 points. RideBench scores reach 74.16 and 71.91, up by 3.37 and 12.36 points, extending the observed benefits to practical business analysis.

A comparison with DataMind further examines the effectiveness of the synthesis approach at a fixed training set size of 5K examples. Using DataPraxis yields average benchmark scores higher by 9.14 points for Qwen3.6-27B and 13.66 points for Qwen3.6-35B-A3B. These advantages demonstrate the value of workspace evolution for synthesizing training data for complex analytical tasks.

With just 5K training examples, both DataPraxis models also achieve scores close to or above those of several frontier models. DataPraxis-27B is comparable to GLM-5.2 in both average benchmark score and JobBench score, while exceeding DeepSeek-V4-Pro, GPT-5.6-Terra, and Gemini-3.6-Flash in average score. For DataPraxis-35B-A3B, the DABStep score is close to those of DeepSeek-V4-Pro. Taken together, these comparisons demonstrate the potential of synthesized training data to make smaller models competitive with frontier models on complex analytical tasks.

4.3 Ablation Studies

Training data volume. Table 3 shows that increasing the training set from 2K to 5K examples improves the DABStep score from 48.89% to 55.11% and JobBench from 45.50 to 51.90. From 4K to 5K examples, DABStep gains 0.66 percentage points, while JobBench improves by a further 3.10 points. DABStep emphasizes prerequisite computations within complex analyses, whereas JobBench requires agents to complete complex tasks and produce deliverables. The continued improvement on JobBench suggests that additional training data remain valuable for learning to complete such tasks, even as gains on the computational subtasks become smaller.

Evolution depth. We define task evolution depth as the number of evolution steps along the longest dependency chain from the initial workspace to the workspace selected for synthesis. To examine how depth affects task difficulty, we use Qwen3.6-35B-A3B to solve synthesized tasks independently. Figure 4 shows that the task pass rate decreases from 35.60% at depth 1 to 22.90% at depth 4, while the mean number of turns involving tool calls increases from 17.8 to 30.1. These trends indicate that deeper tasks require more tool interaction and are harder for the model to solve. We further examine how tasks synthesized at different evolution depths contribute to model performance. Table 3 compares two Qwen3.6-27B models, each trained on 2K tasks. The shallow setting uses tasks synthesized at workspace evolution depths 1–2, while the deep setting uses tasks from depths 3–5. Compared with the shallow setting, the deep setting improves the JobBench score by 2.30 points, suggesting that the greater demands of deeper workspace evolution can benefit learning to complete complex tasks. The shallow setting, in turn, achieves a DABStep score 1.78 percentage points higher than the deep setting, indicating the value of earlier evolution stages for learning prerequisite analytical computations. These complementary benefits motivate selecting tasks from both shorter and longer workspace evolution paths to develop both capabilities.

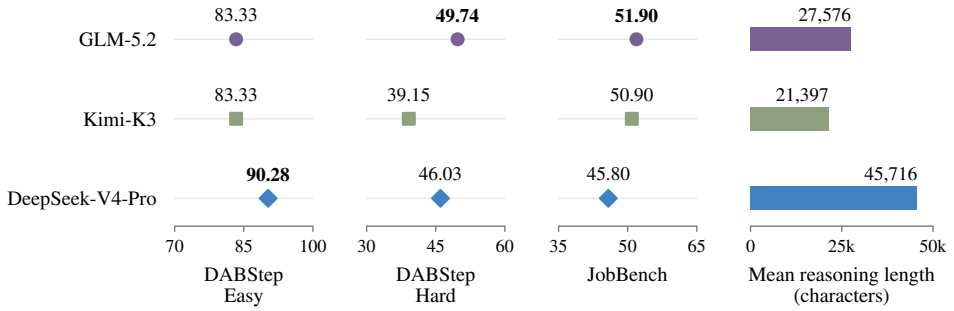
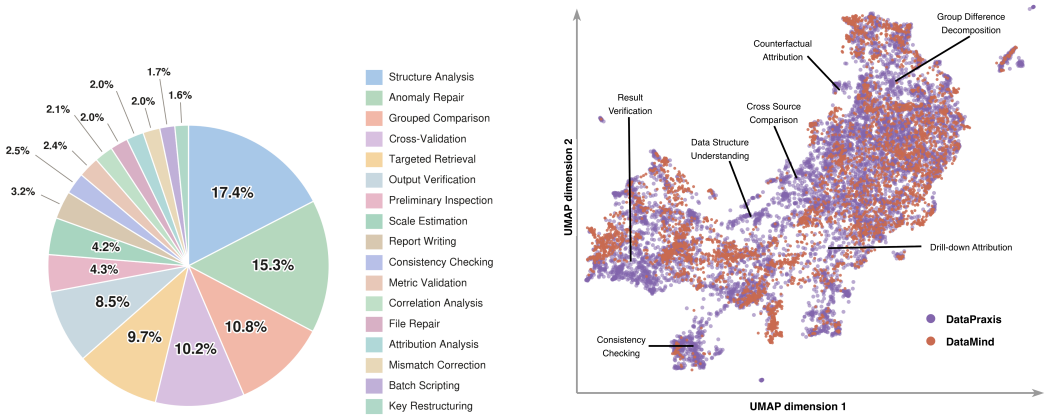


Figure 5: Teacher ablation for Qwen3.6-27B using 5K training trajectories per teacher. Benchmark values denote student scores, with the best in bold.

Dual-evidence verification. With the training set size fixed at 5K examples, removing dual-evidence verification on trajectories reduces DABStep from 55.11% to 50.00% and JobBench from 51.90 to 47.50. Sampled trajectories from teacher model may contain undesirable behavioral patterns or analyses that lack sufficient evidential support. Dual-evidence verification excludes trajectories whose deliverables fail the rubrics, helping remove unreliable solutions.

Trajectory teacher. Figure 5 compares student performance and the mean reasoning length of training trajectories from three teachers. Kimi-K3 has the highest average teacher score in Table 2, yet students trained on GLM-5.2 trajectories score 10.59 percentage points higher on DABStep Hard. Compared with DeepSeek-V4-Pro, GLM-5.2 yields student scores 3.71 percentage points higher on DABStep Hard and 6.10 points higher on JobBench. DeepSeek-V4-Pro yields the highest student score on DABStep Easy, at 90.28%. Teacher performance alone does not predict how much the student benefits from its trajectories. Kimi-K3 has the lowest mean reasoning length and may leave intermediate steps implicit despite its stronger reasoning. DeepSeek-V4-Pro averages 45,716 characters, but its longer reasoning does not yield better performance. GLM-5.2 offers a more suitable balance of reasoning quality and explicit intermediate steps for these tasks.

4.4 Diversity of Analytical Behaviors



(a) (b)

Figure 6: Analytical diversity. (a) Action shares across the top-level behavioral clusters derived from DataPraxis-5k. (b) UMAP projection of action representations from DataPraxis and DataMind.

By varying source workspaces and evolution strategies, DataPraxis constructs tasks with different analytical requirements. We characterize the behavior diversity in training trajectories using a hierarchical clustering procedure inspired by Clio (Tamkin et al., 2024). Each action represents a coherent analytical intent. We embed individual actions with Qwen3-4B-Embedding (Zhang et al., 2025b),

cluster their representations, and use GLM-5.2 to name the clusters. In each subsequent round, the names assigned in the preceding round are embedded, clustered, and named in the same manner. Three rounds of this procedure produce a behavioral hierarchy without predefined categories.

Figure 6(a) shows that DataPraxis trajectories span 17 top-level behavioral clusters, indicating diversity across forms of analytical work. Clusters for structure analysis and anomaly repair appear alongside those for grouped comparison and output verification, with further behaviors represented by the remaining clusters. Figure 6(b) projects the action representations used in the initial clustering round into a common UMAP (McInnes et al., 2018) space. Its annotations describe the analytical behaviors captured by selected clusters. Compared with DataMind, DataPraxis actions occupy a broader range of behavioral regions in the projection, providing qualitative evidence that its trajectories encompass more varied analytical work.

5 Related Work

Task-Type and Skill Guided Task Synthesis. DataMind (Qiao et al., 2026) generates analytical questions from data metadata, fine-grained task categories, and exemplars, then increases complexity through recursive composition. DeepAnalyze (Zhang et al., 2025a) conditions interaction-task generation on sampled task types before executing and checking solution trajectories. AgenticDataBench (Sun et al., 2026), SkillSynth (Fan et al., 2026), and SkillNet (Liang et al., 2026) organize task generation through skill paths or workflows, with SkillNet additionally using exploration reports to construct tasks and evaluation components. These methods enable targeted coverage of specified tasks or skills. However, predefined categories and skill structures may constrain question discovery. Our DataPraxis continually guides task generation with verified analytical findings, uncovering deeper questions beyond these predefined structures.

Exploration-Based Task Synthesis. OS-Genesis (Sun et al., 2024) and CuES (Mai et al., 2025) construct tasks from state transitions or interaction patterns obtained through environment exploration. AutoPlay (Ramrakhya et al., 2025) explores environment states and available functionalities before generating tasks from the resulting trajectories. WebExplorer (Liu et al., 2025) discovers interconnected facts through search and browsing before generating and iteratively refining questions. DIVE (Chen et al., 2026) alternates tool execution, evidence accumulation, and backward question construction. Yet these methods offer limited support for reusing computations and combining analytical branches. Our DataPraxis inherits and merges verified workspaces to uncover deeper questions beyond individual branches, while retaining traceable evidence for task evaluation.

Compositional and Evolutionary Task Synthesis. AgentSynth (Xie et al., 2025) alternates the generation and execution of state-dependent subtasks before integrating them into long-horizon instructions. TaskCraft (Shi et al., 2025) introduces task dependencies through depth expansion and combines independent subtasks through width expansion. Recursive Synthesis (Li et al., 2026b) extends reference solutions from verified seed tasks, jointly updates instructions, environments, and verifiers, and revalidates the resulting tasks. These methods increase task complexity by composing subtasks or extending existing solutions. While adding steps can increase task complexity, it does not necessarily reveal objectives that emerge through analysis. Our DataPraxis constructs tasks backward from verified findings and their dependencies, turning these emergent objectives into analytically demanding tasks with grounded evaluation criteria.

6 Conclusion

We introduced DataPraxis, a framework that uses verified workspace evolution to synthesize professional analytical tasks. By accumulating and combining analytical findings, it uncovers deeper questions that are difficult to formulate from raw materials alone. Verification draws on evolution evidence alongside solution evidence to establish task criteria and filter training trajectories. Experiments with DataPraxis-5K demonstrated improved agent performance across two model scales and multiple benchmarks. These findings support the value of accumulated workspace knowledge as a foundation for synthesizing deeper tasks. DataPraxis offers a promising approach to scaling task difficulty, opening new possibilities for pushing the boundaries of model capabilities. We illustrate limitations in Appendix D.

AI USE STATEMENT

We used generative AI tools to support research ideation and execution, retrieve and discover related literature, translate manuscript content, draft sections of the paper, and polish the manuscript's wording and presentation. We also used these tools to generate the synthetic dataset introduced in this work.

Within DataPraxis, language models profile workspaces, propose and execute analytical tasks, validate workspace changes, assemble task instructions, construct and validate rubrics, and generate training trajectories. The synthesis and reference models are specified in the experimental setup. We additionally use a language model to name behavioral clusters in Section 4.4. Generated analyses and trajectories undergo the execution and verification procedures described in Section 3, with examples in Appendix B. The authors take responsibility for the final text, claims, and research artifacts, including content prepared with AI assistance.

ETHICS STATEMENT

DataPraxis is constructed from publicly available datasets obtained from Kaggle, DataWorld, and GitHub repositories. We checked the license of each source dataset to ensure that its use in this work is permitted. The data used in this study do not contain sensitive personal information. Task instructions, evaluation rubrics, and solution trajectories are generated through the automated synthesis pipeline, without human annotation or participation by human subjects. RideBench uses simulated order, driver, and passenger records, as described in Appendix A. We plan to release DataPraxis-5K for research on analysis agents. The release will respect the licenses and redistribution conditions of the underlying source datasets.

REPRODUCIBILITY STATEMENT

Section 3 describes workspace initialization, the evolution modes, backward task synthesis, and verification for trajectory selection. The experimental setup specifies the synthesis and reference models, execution harness, training hyperparameters, benchmarks, and baseline construction. Appendix E provides the agent prompts and structured output formats used for task synthesis and verifier construction. Appendix A documents RideBench's simulation procedure, scenarios, and task distribution. Appendix B illustrates workspace evolution and rubric validation, while Appendix C defines the behavioral measurements and presents comparisons of model behavior before and after training.

REFERENCES

Anthropic. Claude Code. Official software repository. URL https://github.com/anthropics/claude-code. Accessed September 17, 2026.

Anthropic. Introducing Claude Opus 5. Anthropic Blog, 2026. URL https://www.anthropic.com/news/claude-opus-5.

A. Chen et al. DIVE: Scaling Diversity in Agentic Task Synthesis for Generalizable Tool Use. *arXiv preprint arXiv:2603.11076*, 2026. URL https://arxiv.org/abs/2603.11076.

DeepSeek. DeepSeek-V4 preview: Entering the era of affordable million-token context. DeepSeek Blog, 2026. URL https://deepseek.com/en/news/v4-preview/.

Tulsee Doshi. Introducing Gemini 3.6 Flash, 3.5 Flash-Lite, and 3.5 Flash Cyber. Google Blog, 2026a. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-6-flash-3-5-flash-lite-3-5-flash-cyber/.

Tulsee Doshi. Introducing Gemini 3.7 Flash. Google Blog, August 2026b. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/introducing-gemini-3-7-flash/.

A. Egg et al. DABstep: Data Agent Benchmark for Multi-step Reasoning. *arXiv preprint arXiv:2506.23719*, 2025. URL https://arxiv.org/abs/2506.23719.

Z. Fan et al. Toward Scalable Terminal Task Synthesis via Skill Graphs. *arXiv preprint arXiv:2604.25727*, 2026. URL https://arxiv.org/abs/2604.25727.

S. Hong et al. Data Interpreter: An LLM Agent For Data Science. *arXiv preprint arXiv:2402.18679*, 2024. URL https://arxiv.org/abs/2402.18679.

Y. Li et al. JobBench: Aligning Agent Work With Human Will. *arXiv preprint arXiv:2605.26329*, 2026a. URL https://arxiv.org/abs/2605.26329.

Z. Li et al. Recursive Synthesis for Long-Horizon Terminal Tasks. *arXiv preprint arXiv:2608.05466*, 2026b. URL https://arxiv.org/abs/2608.05466.

Y. Liang et al. SkillNet: Create, Evaluate, and Connect AI Skills. *arXiv preprint arXiv:2603.04448*, 2026. URL https://arxiv.org/abs/2603.04448v3.

Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, Jiayuan Song, Zhengmao Zhu, Wenhu Chen, Pengyu Zhao, and Junxian He. WebExplorer: Explore and Evolve for Training Long-Horizon Web Agents. *arXiv preprint arXiv:2509.06501*, 2025. URL https://arxiv.org/abs/2509.06501.

S. Mai et al. CuES: A Curiosity-driven and Environment-grounded Synthesis Framework for Agentic RL. *arXiv preprint arXiv:2512.01311*, 2025. URL https://arxiv.org/abs/2512.01311.

Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426*, 2018. URL https://arxiv.org/abs/1802.03426.

Moonshot AI. Kimi K3: Open frontier intelligence. Kimi Blog, 2026. URL https://www.kimi.com/en/blog/kimi-k3.

OpenAI. Codex CLI. Official software repository. URL https://github.com/openai/codex. Accessed September 17, 2026.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. OpenAI Blog, 2026. URL https://openai.com/index/gpt-5-6/.

S. Qiao et al. Scaling Generalist Data-Analytic Agents. In *ICLR*, 2026. URL https://arxiv.org/html/2509.25084v3.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026a. URL https://qwen.ai/blog?id=qwen3.6-27b.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026b. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Ram Ramrakhya, Andrew Szot, Omar Attia, Yuhao Yang, Anh Nguyen, Bogdan Mazoure, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling Synthetic Task Generation for Agents via Exploration. *arXiv preprint arXiv:2509.25047*, 2025. URL https://arxiv.org/abs/2509.25047.

D. Shi et al. TaskCraft: Automated Generation of Agentic Tasks. *arXiv preprint arXiv:2506.10055*, 2025. URL https://arxiv.org/abs/2506.10055.

Q. Sun et al. OS-Genesis: Automating GUI Agent Trajectory Construction via Reverse Task Synthesis. *arXiv preprint arXiv:2412.19723*, 2024. URL https://arxiv.org/abs/2412.19723.

Z. Sun et al. AgenticDataBench: A Comprehensive Benchmark for Data Agents. *arXiv preprint arXiv:2607.01647*, 2026. URL https://arxiv.org/abs/2607.01647.

A. Tamkin et al. Clio: Privacy-Preserving Insights into Real-World AI Use. *arXiv preprint arXiv:2412.13678*, 2024. URL https://arxiv.org/abs/2412.13678.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, et al. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *ICLR*, 2025. URL https://arxiv.org/abs/2407.16741.

J. Xie et al. AgentSynth: Scalable Task Generation for Generalist Computer-Use Agents. *arXiv preprint arXiv:2506.14205*, 2025. URL https://arxiv.org/abs/2506.14205.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*, 2023. URL https://arxiv.org/abs/2210.03629.

Z.ai. GLM-5.2. Z.ai Blog, 2026. URL https://z.ai/blog/glm-5.2.

Mario Zechner and contributors. Pi Coding Agent. Official software repository. URL https://github.com/badlogic/pi-mono. Accessed September 17, 2026.

S. Zhang et al. DeepAnalyze: Agentic Large Language Models for Autonomous Data Science. *arXiv preprint arXiv:2510.16872*, 2025a. URL https://arxiv.org/abs/2510.16872.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv preprint arXiv:2506.05176*, 2025b. URL https://arxiv.org/abs/2506.05176.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 400–410. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-demos.38. URL https://aclanthology.org/2024.acl-demos.38/.

A RIDEBENCH CONSTRUCTION

RideBench was developed internally by a company to evaluate business analysis in ride-hailing operations. It uses a simulator that models interactions among passenger demand, driver supply, and platform policies. The simulator first establishes a stable operating state. Scripted events then alter selected parameters, allowing their effects to propagate through the simulated business. The resulting data capture changes in operational metrics under known interventions.

The benchmark covers 11 scenarios involving external events and platform strategies, including competitive pricing, subsidies, driver shortages, and extreme weather. Some scenarios introduce a single event, while others combine interventions to support more complex attribution and strategy comparisons. Agents receive order records and daily driver and passenger tables. Internal simulation variables and event logs are retained separately to support reference analysis and evaluation.

The 89 tasks are organized into three difficulty levels according to the required analysis and scenario complexity. The 56 easy tasks focus on metric computation, anomaly detection, and basic attribution. The 23 medium tasks require more detailed attribution or inference about external events. The 10 hard tasks emphasize operational strategy comparison and complex attribution. Together, these tasks assess whether agents can connect observed business changes to their underlying drivers.

B CASE STUDIES

B.1 WORKSPACE EVOLUTION

Workspace evolution turns intermediate findings into both reusable analytical assets and grounds for subsequent questions. Figure 7 illustrates this process through a hotel pricing example. Two branches characterize different dimensions of the same booking records. Their composition then tests whether the relationships found on each dimension persist when the other is taken into account.

Initial workspace. The workspace contains `hotel_bookings.csv`, with 119,390 booking records for a City Hotel and a Resort Hotel, with arrival dates spanning July 2015 to August 2017. Available fields include the number of days between booking and arrival, the average daily room rate (ADR), hotel identity, and cancellation status. The initial snapshot also includes `data_profile.json`. The execution environment is represented by the task package's `Dockerfile`. We follow the two pricing branches and their composition, denoted A, B, and C in the figure.

Node A: discovering the relationship with booking lead time. The first branch groups room rates by booking lead time. Among completed stays with positive room rates and stay lengths, the Resort Hotel exhibits an inverted U: median ADR rises from EUR 61 for bookings made 1–7 days before arrival to EUR 90 for bookings made 121–180 days before arrival, then declines at longer lead times. The City Hotel appears comparatively flat, with a range of only EUR 8.1 across lead time bins from 0 to 365 days. The workspace retains `lead_time_analysis.py`, an enriched table `completed_stays.csv`, and grouped summaries in `lead_time_prices.csv`. It also establishes the cancellation filter and lead time bins used in the subsequent analysis.

Node B: identifying the seasonal context. A separate branch constructs arrival dates and examines monthly price patterns. In the complete calendar year of 2016, median ADR peaks in August for the Resort Hotel and September for the City Hotel. The same peak months remain when canceled bookings are excluded. This branch adds `seasonal_analysis.py`, which defines the calendar mapping, together with `monthly_prices.csv` and `price_peaks.json`. Its findings identify specific months in which to revisit Node A's pooled relationships.

Node C: composing the findings into a new question. The composition node inherits both branches' scripts and result tables. Its recorded objective explicitly connects their findings: does the Resort Hotel's lead time premium persist in August, and does the City Hotel's apparently flat relationship persist in September? The executor reuses Node A's filtering and lead time bins together with Node B's calendar definitions to construct a joint grid of lead time and arrival month

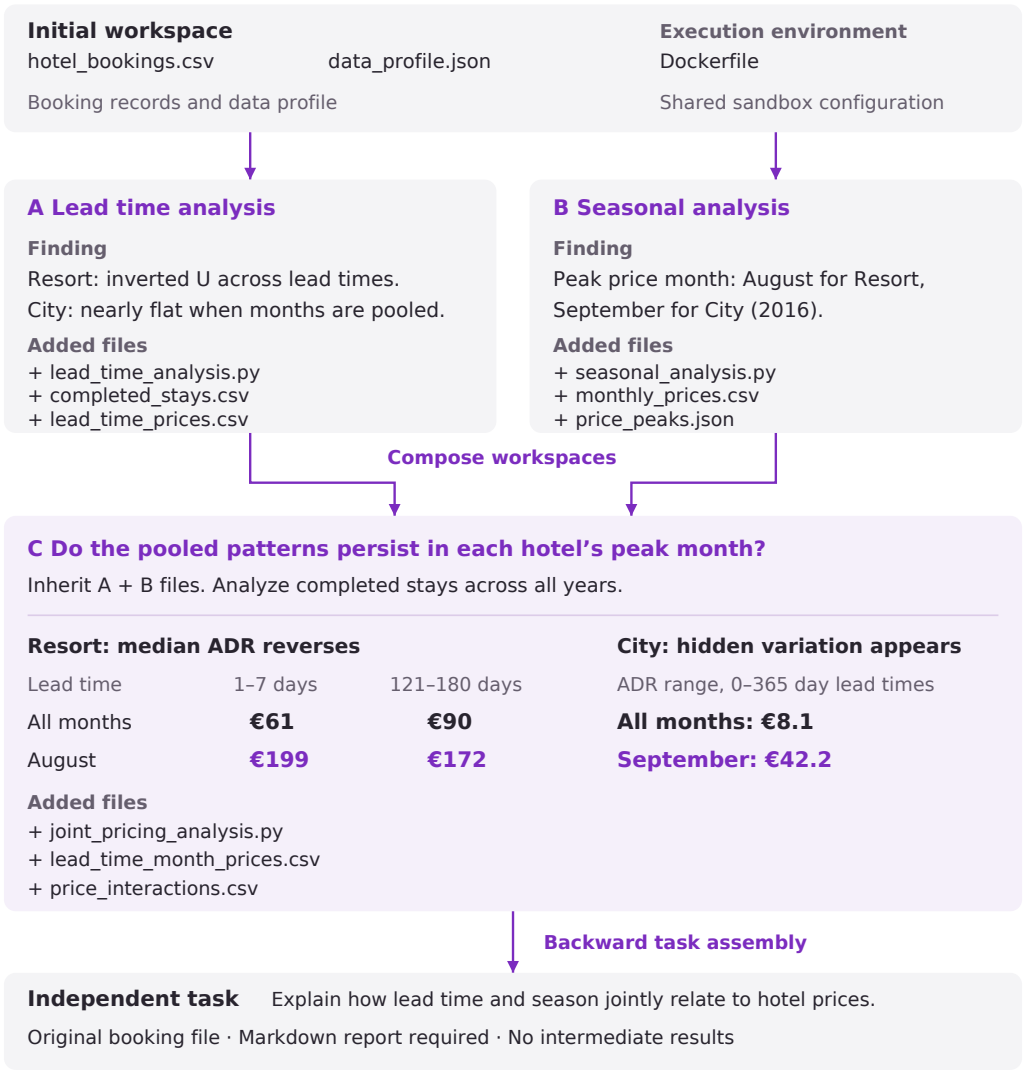


Figure 7: **Workspace evolution in hotel pricing analysis.** A–C denote selected evolution nodes. Arrows show workspace inheritance and subsequent task assembly. Descriptive filenames are used for readability.

across all available years. The resulting analysis changes the interpretation of both pooled patterns. In August, the Resort Hotel's median ADR is EUR 199 for the 1–7 day bin but EUR 172 for the 121–180 day bin, reversing the corresponding pooled comparison. For the City Hotel, the range across the same 0–365 day bins expands from EUR 8.1 when months are pooled to EUR 42.2 in September. The successor workspace adds `joint_pricing_analysis.py`, the joint grid `lead_time_month_prices.csv`, and tests in `price_interactions.csv`, making these conditional findings available for further evolution.

From evolved workspace to an independent task. The assembled task asks a solver to analyze how booking lead time and arrival season jointly relate to room rates, assess whether their relationships can be treated separately, and compare the two hotels in a Markdown report. The solver receives the original booking file, while construction scripts, intermediate tables, and discovered conclusions are withheld. The dependencies established during evolution thus become analytical work that the solver must reconstruct. This case illustrates how composition goes beyond collect-

ing separate results: a finding from one branch specifies where to interrogate another, producing a question and a conditional explanation that neither branch had established alone.

B.2 Rubric Synthesis and Validation

The dual-evidence verification process turns task requirements and analytical evidence into criteria that can be checked against a deliverable. A coffee retail task illustrates how validation separates the analytical requirement from a conclusion that depends on a particular choice of analysis population. The task uses `Coffee Shop Sales.xlsx`, containing 149,116 sales rows from three stores, and requests a Markdown report on product co-occurrence and bundle recommendations. A shopping basket is defined by transaction date, time, and store.

From evidence to candidate criteria. Evolution evidence from the workspace evolution record supplies verified findings about basket composition and product associations. Solution evidence from two independent reference trajectories is organized into evidence cards linking operations, computed values, and report conclusions. The rubric generator produces ten candidate criteria covering substantive conclusions, basket construction, association measures, recommendations, and report completeness. One criterion requires distinguishing frequent co-occurrence from strong association. Its initial satisfaction rule also requires the report to conclude that Bakery and Tea have lift greater than one (Table 4).

Table 4: Rubric refinement for coffee basket analysis. Criterion text is condensed and translated from the recorded rubric.

Stage	Selected criterion and supporting evidence
Candidate	Report both co-occurrence frequency and lift. Identify Bakery + Coffee as frequent but with lift below one, and Bakery + Tea as having lift above one.
Validation	Bakery + Tea appears in 7,983 baskets. Its lift is 0.920 among all 116,790 baskets and 1.197 among 28,904 baskets containing multiple categories. Bakery + Coffee has lift below one under both populations (0.951 and 0.857).
Final criterion	Report both frequency and lift, recognize that Bakery + Coffee is frequent without a positive association, and identify at least one category pair with lift above one. The required positive pair is no longer fixed to Bakery + Tea.

Validation establishes the scope of the expected conclusion. The rubric validator checks the referenced fields, the criterion's observability in the reports, and its conclusion against the workbook. For categories A and B, lift is $n_{AB}N/(n_A n_B)$, where N is the number of baskets, n_A and n_B count baskets containing each category, and n_{AB} counts baskets containing both. All counts refer to the same basket population. Recomputing this quantity shows why the initial requirement is too specific: Bakery + Tea has lift below one over all baskets but above one after restricting to baskets containing multiple categories. The two reference reports use these different populations. The validation record therefore revises the criterion to retain the distinction between frequency and association while removing the required direction for this particular pair. The revised criterion passes the subsequent validation check.

Preserving a substantive scoring requirement. The final criterion retains a substantive test: the report must distinguish frequent co-occurrence from positive association using both frequency and lift. Listing popular combinations alone is insufficient.

C Model Behavior Before and After Training

We compare Qwen3.6-27B and DataPraxis-27B on the same 65 JobBench tasks across 35 occupations. We examine two observable behaviors: organizing work into explicit subgoals and checking generated outputs.

More explicit task organization. We identify explicit subgoals from successful task creation records for analysis or deliverable construction, excluding final verification goals. The number

of tasks with explicit subgoals increases from 47 to 65 after training (Figure 8(a)). Among the 47 tasks with subgoals for both models, DataPraxis-27B introduces them after fewer tool calls in 44 tasks, excluding task management calls. The trained model therefore makes intermediate objectives explicit more consistently, with earlier introduction in most tasks where both models use subgoals.

More frequent self-verification. We examine self-verification through output inspection: tool calls that reopen or parse generated documents, spreadsheets, or data files to inspect their structure or contents. Checking only whether a file exists does not count. We compute inspection attempts per 100 nonmanagement tool calls for each task and average these rates equally across tasks. The mean increases from 6.64 to 10.36 per 100 tool calls, excluding task management calls (Figure 8(b)). This indicates that inspecting generated outputs occupies a larger share of tool use after training.

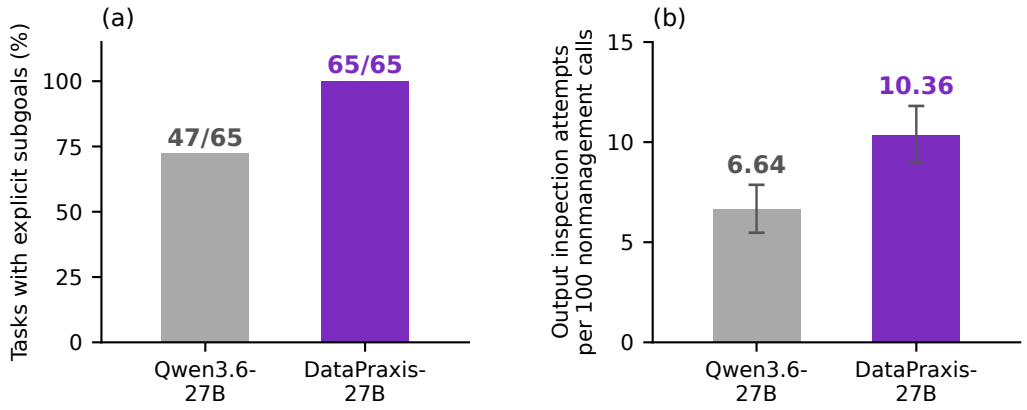


Figure 8: **Task organization and output inspection on JobBench.** (a) Number of tasks with explicit subgoals, expressed as a percentage of all 65 matched tasks. Final verification goals are excluded. (b) Mean output inspection attempts per 100 nonmanagement tool calls, with 95% bootstrap intervals over tasks. Output inspection is an observable proxy for self-verification. Both panels compare Qwen3.6-27B with DataPraxis-27B on the same task set.

C.1 Illustrative Cases of Behavioral Changes

We examine two tasks to illustrate the behaviors measured above. Each case compares Qwen3.6-27B and DataPraxis-27B on the same task: a VPN incident investigation illustrates explicit task decomposition, and workforce planning illustrates output inspection followed by revision (Figure 9).

Organizing a complex request into explicit subgoals. The VPN task requires reconciling gateway logs, incident records, and helpdesk tickets, then producing a report, a CSV containing two change requests, and a dashboard database. Qwen3.6-27B reads the sources and writes a script that generates all three deliverables, without registering explicit subgoals. After an initial inspection, DataPraxis-27B records four substantive subgoals: analyze the sources, prepare the reconciliation report, prepare the change requests, and build the dashboard. These goals retain requirements that matter to each output. For example, the change request goal specifies session and incident identifiers as evidence and requires the DNS rollback plan to account for tunnel routing discrepancies. The model then works through source analysis, report construction, change requests, and dashboard construction, updating the corresponding goal statuses as it proceeds. This makes the task's requirements explicit at the level of individual work stages, providing a concrete example of the increased subgoal coverage in Figure 8(a).

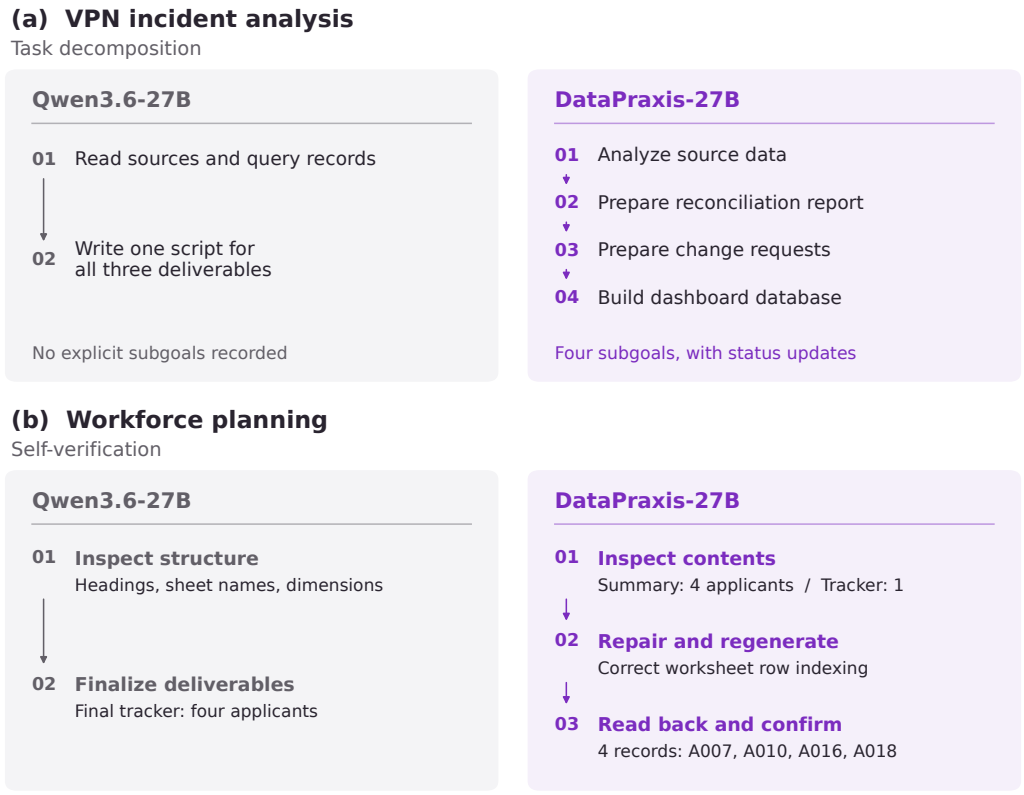


Figure 9: **Paired cases illustrating the two aggregate behavior patterns.** (a) Explicit subgoals organize VPN incident analysis and its deliverables. (b) Inspecting workbook contents reveals an omission and guides its repair. Text summarizes recorded operations.

Using output contents to guide revision. In the workforce planning task, the models analyze employee turnover and screen applicants, producing a report and a workbook with a license follow-up tracker. The trajectories show how output inspection informs revision. Qwen3.6-27B inspects document headings, worksheet names, and worksheet dimensions. DataPraxis-27B additionally reads worksheet rows and summary values. Its draft summary lists four applicants requiring license follow-up, but the tracker contains only one applicant record. The subsequent edit fixes the row indexing used to populate the tracker, preventing instruction text from overwriting candidate entries. After regenerating the workbook, the model reads the tracker again and confirms all four applicant IDs. Both models ultimately deliver trackers containing these four applicants. The distinctive evidence is the trained model's observed sequence of content inspection, targeted repair, and confirmation within its own trajectory. This sequence illustrates how output inspection, the behavior measured in Figure 8(b), can lead to a concrete correction before delivery.

D LIMITATIONS

DataPraxis relies on seed workspaces with sufficient data complexity and analytical scope to support continued exploration. Simple files with few variables or limited relationships may offer fewer opportunities to uncover deeper questions through workspace evolution. In addition, the difficulty of synthesized tasks and the quality of solution trajectories depend on the capabilities of the teacher models. Their ability to identify meaningful analytical directions and execute the required analyses shapes the resulting training data.

E AGENT PROMPTS

These condensed English templates retain the agents' core instructions and structured outputs. Angle brackets denote runtime inputs or values to fill in. A vertical bar separates allowed values. Use JSON null rather than a quoted string where specified, and empty lists when no entries apply. Each agent is shown once, with the Director's output selected by its operation mode.

E.1 PROFILER

You are a data profiler. Inspect the initial data and research the relevant domain to produce a structured profile for subsequent exploration. No analytical objective has been assigned yet.

```
Domain context: <task_description>
Documentation: <briefing_files>
Search and fetch tools: <search_tool>, <fetch_tool>
Output path: <output_path>
```

Inspect the current workspace using scripts and suitable readers. Consult documentation before inferring field meanings. Identify artifacts, entity keys, explicit and implicit relationships, and the valuable analytical questions these relationships support. For databases, inspect schemas and basic size information without performing downstream analyses.

Research professional activities, role descriptions, analytical frameworks, and methods using the supplied search and fetch tools. Use this research to broaden the domain's work types and methods. Skip online research only if the tools are unavailable. Keep research findings in the domain profile. Do not import external numerical benchmarks or metric definitions into the workspace profile.

List atomic professional work types that can be independently performed and checked and remain meaningful for other datasets in the same domain. Separately list methods with domain concepts and their reasoning mechanisms, not generic descriptive statistics. Do not prescribe pairings between work types and methods.

Record locally observed prerequisites as usage and silent analytical pitfalls as hazard. Do not invent issues. Use Python to write only data_profile.json to <output_path>, using the following structure. The artifact note is optional.

```
{
  "workspace_profile": {
    "overview": "<2-4 sentences describing the data and scale>",
    "artifacts": [{"ref": "<file or table>", "role": "<contents>",
                   "note": "<optional loading or usage note>"}],
    "relations": ["<objects, joining keys, and analytical value>"],
    "observations": [{"kind": "usage|hazard", "note": "<observation>"}]
  },
  "domain_profile": {
    "work_modes": [{"id": "<snake_case_id>", "intent": "<purpose>"}],
    "domain_methods": [{"name": "<method>", "logic": "<reasoning mechanism>"}]
  }
}
```

E.2 DIRECTOR

You are the exploration director. Plan valuable new analytical questions from the exploration graph. Follow the supplied expansion or composition instructions.

```
Domain context and profile: <source_task_hint>
Candidate parents: <candidate_list>
Exploration graph: <tree_overview>
Depth and limits: <depth_status>, <max_depth>, <per_call_cap>
Operation instructions: <operator_menu>, <required_operator_note>
Composition limit: <max_compositions>
Output path: <output_path>
```

Read the full graph to avoid repeating existing questions. Favor substantive questions with practical or research value and support in the available data. Specify what to investigate and why, without calculation steps, formulas, expected numbers, or instructions for avoiding data pitfalls.

For expansion, choose one eligible parent and propose up to <per_call_cap> distinct directions. At the data root, derive initial questions from the profile. Otherwise identify the specific verified finding each direction

depends on. A merely related topic is insufficient. Assign an allowed operator and a work_mode to every direction. Work types are an open set, not a coverage quota. If no candidate supports a valuable new direction, set can_expand to false, parent_node_id to null, and branches to an empty list.

For composition, propose up to <max_compositions> natural questions, each requiring at least two verified sources. Explain their relationship and what new conclusion their combination could support. Do not merely concatenate independent questions. Keep the question in objective and the reasoning about source findings in divergence_rationale. Return an empty list if no meaningful composition exists.

Use Python to write only director_direction.json to <output_path>. Use the expansion structure below, or the composition structure when composition is requested.

```
Expansion:
{
  "can_expand": true,
  "parent_node_id": "<eligible node ID, or null when stopping>",
  "branches": [{"next_focus": "<question>", "operator": "<allowed operator>",
    "depends_on": "<parent finding, or empty at the data root>",
    "work_mode": {"id": "<work type>", "intent": "<purpose>"},
    "insight": "<why this direction is valuable>"}]
}
Composition:
{
  "operator": "composition",
  "compositions": [{"source_node_ids": ["<verified ID>", "<verified ID>"],
    "relation": "<relationship>", "objective": "<single question>",
    "divergence_rationale": "<why multiple sources are needed>"}],
  "reasoning": "<rationale for the number of proposals>"
}
```

E.3 PROPOSER

You are a goal proposer. Translate the Director's direction into a clear analytical task for the Executor. Specify what to answer and deliver, not how to calculate it.

```
Data profile: <data_profile>
Domain context: <source_task_hint>
Verified predecessor findings: <history_summary>
Direction: <director_direction>
Operation instructions: <operator_note>
Output path: <output_path>
```

Understand the core question and write a self-contained instruction describing the analytical setting, objects, question, and expected output. Keep it at a moderate scale with typically one to three central conclusions that can be reliably executed and independently checked. Do not prescribe a calculation recipe or supply the answer.

Identify the specific predecessor finding on which the new objective depends. Prepare context_summary from history_summary alone. Include verified findings and their source nodes so that the Executor can reuse them directly. Numerical facts must come from verified history. The Director's proposal is a direction for future work, not evidence of a result. Do not perform the analysis yourself.

Use Python to write only goal_package.json to <output_path>, using exactly this structure.

```
{
  "current_goal": {
    "goal_text": "<setting, analytical question, and expected output>",
    "depends_on": "<specific predecessor finding>"
  },
  "context_summary": "<verified prior conclusions with source nodes>"
}
```

E.4 EXECUTOR

Complete the assigned analysis and leave reusable assets for subsequent nodes.

```
Goal: <goal_text>
Verified prior findings: <context_summary>
Workspace files: <workspace_files>
Data pitfalls: <data_traps>
Inherited scripts: <inherited_scripts>
Inherited artifacts: <inherited_artifacts>
Node index and output path: <round_idx>, <output_path>
```

Reuse or extend verified scripts and intermediate artifacts under sources/. Compute from raw data when an inherited result is unavailable. Write analyze_node<round_idx>.py as a self-contained script that reproduces the key numerical evidence from persisted inputs, without manual steps or transient state. Save new intermediate artifacts under generated/.

Derive directional and quantitative claims from computed values. If findings overturn an expectation, correct the script and rerun it. Do not patch the result JSON while leaving contradictory code. State conclusion as prose with its supporting numbers inline so subsequent nodes can reuse it directly.

Write analysis_log_node<round_idx>.md documenting the subproblems, assumptions and processing choices, calculation steps and intermediate values, changes prompted by the findings, and the evidence supporting each final claim. Include both the script and log in generated_files, together with reusable artifacts.

Write execution_result.json to <output_path> with all fields below. Set execution_status to success or failed. Use null for failure_reason on success. A successful conclusion must be nonempty prose, not a structured table.

```
{
  "execution_status": "success|failed",
  "observations": ["<finding with supporting values>"],
  "methodology": "<analytical method and processing choices>",
  "conclusion": "<conclusion with supporting numbers inline>",
  "generated_files": ["analyze_node<round_idx>.py",
    "analysis_log_node<round_idx>.md", "generated/<artifact>"],
  "failure_reason": null
}
```

E.5 VALIDATOR

You are an analysis validator. Try to falsify the main conclusion fairly, without inventing objections. Determine reliability and establish the verified conclusion.

```
Goal: <goal_text>
Execution result: <execution_result_json>
Workspace files: <workspace_files>
Output path: <output_path>
```

Check execution success, listed files, and substantive observations and conclusion. Failure of any of these checks means unsupported with a null ground_truth. Do not rescue a failed execution by independently completing it. Inspect the required script and analysis log. Rerun the script to check independent reproducibility, which does not itself establish correctness.

Identify the strongest plausible objection. Trace claims to calculations and assumptions, check the appropriateness of methods, and confirm that the analysis answers the assigned question. Judge the chosen analytical perspective on its own terms.

Return supported if the main claim withstands scrutiny. Return revised only after a bounded correction to the existing computation, such as repairing a formula or filter, and rerun the corrected script. If a reliable correction requires new analysis, return unsupported. Preserve sources/ and the original execution_result.json.

Write ground_truth in your own words with verified numbers inline. Record unresolved concerns in issues, and inherited assumptions or intermediate artifacts in condition_dependencies. Use Python to save validation_result.json at <output_path> with the structure below. Set ground_truth to null for unsupported. Include zero to three issues, using major, moderate, or minor severity. For revised, revisions must identify script changes and the recomputed values.

```
{
```

```
  "main_claim": "supported|revised|unsupported",
  "ground_truth": "<verified prose conclusion, or null if unsupported>",
  "revisions": ["<correction and recomputed evidence>"],
  "issues": [{"severity": "major|moderate|minor",
              "description": "<specific issue>"}],
  "condition_dependencies": ["<inherited assumption or artifact>"],
  "summary": "<what was checked and whether the conclusion holds>"
}
```

E.6 VERIFIER GENERATION

You are a read-only rubric generation agent. Derive criteria from the task, data, solving evidence, and candidate reference conclusions. Reference candidates may be wrong and must not be treated as guaranteed truth.

```
Instruction: <instruction>
Data files: <data_file_paths>
Evidence cards: <evidence_cards>
Reference candidates: <gt_candidate_rubrics>
```

Use Read, Bash, Grep, and Glob without modifying files. Read only supplied paths. Inspect the task, data schema, evidence cards, and reference candidates. Induce four to seven evaluation dimensions covering main conclusions, analytical methods, and presentation, with greatest importance given to the main conclusions. Assess presentation from extracted text, not rendered pixels.

Generate one to three criteria per dimension and at most ten overall. Each criterion must test one observable concern with a justified expected outcome. Drop correctness criteria whose expected outcome you cannot establish. Do not accept opposite answers, mere self-consistency, or answer presence as correctness. Allow genuinely open answers justified by the task and data, and completeness checks for explicitly required presentation.

Use filter for fixed primary conclusions and preference for method or presentation. Label provenance honestly as gt or generated. A downstream step caps gt criteria at 40 percent, dropping those with lowest confidence if exceeded. Deduplicate across all sources and record removals.

Return exactly one minified JSON object in the final reply, with the following fields. Do not use Markdown fences or additional prose. Tool output is not the final reply. Use null for source_field when no reference field applies.

```
{
  "rubric_id": "<ID>", "instruction_key": "<key>",
  "evaluation_dimensions": ["<dimension>"],
  "criteria": [{
    "criterion_id": "F01", "type": "filter|preference",
    "dimension": "<one evaluation dimension>",
    "criterion": "<declarative expected outcome>",
    "observable": "<evidence to inspect>",
    "satisfy": "<conditions for satisfying the criterion>",
    "violate": "<conditions for violating the criterion>",
    "confidence": 0.0, "source": "gt|generated",
    "source_field": "<ground_truth field, or null>",
    "source_note": "<reference conclusion or independently derived angle>"
  }],
  "rubric_warnings": [], "dedup_notes": ["<removed duplicate>"]
}
```

E.7 VERIFIER VALIDATION

You are a read-only rubric validation agent. Validate one criterion against the task, input data, solving evidence, and delivered output text.

```
Criterion and task: <criterion>, <instruction>
Data files: <data_file_paths>
Evidence: <evidence_cards>, <output_files_by_trajectory>
```

Use Read, Bash, Grep, and Glob without modifying files. Read only supplied paths and never read ground_truth files. Recompute from input data to establish correctness, independently of agreement among trajectories.

Check four angles. Grounding checks whether referenced fields and concepts exist. Observability checks whether supplied text and evidence reveal satisfaction or violation, excluding requirements that need rendered pixels. Formulation checks one clear declarative requirement with meaningful satisfy and violate conditions. Reject correctness clauses that accept opposite answers, mere self-consistency, or only answer presence. Allow justified open questions and explicitly required presentation completeness. Leave neutral clauses that do not exhibit these defects unchanged. Conclusion correctness checks whether the expected outcome holds, using as much recomputation as the claim requires.

Distinguish verified, unverifiable, over_specified, and refuted outcomes. Unverifiable means the data cannot settle the claim, not that a feasible check was skipped. An arbitrary demand for exact values or rankings is over_specified. Do not reject a criterion merely because all current trajectories satisfy it.

Leave scoring_tip empty unless equally valid methods reach the same primary conclusion and differ only in secondary details. Pass only if grounding is supported or ambiguous, observable is observable or partially_observable, well_formed is well_formed, and conclusion_correct is verified or unverifiable. Return exactly one minified JSON object with the following fields, without Markdown or additional prose.

```
{
  "criterion_id": "<ID>",
  "validation_angles": {
    "grounding": {
      "assessment": "supported|unsupported|ambiguous", "reason": "<evidence>",
      "data_evidence": [{"file": "<path>", "sheet": "<sheet>", "fields": ["<field>"]}]
    },
    "observable": {
      "assessment": "observable|partially_observable|unobservable", "reason": "<evidence>"
    },
    "well_formed": {
      "assessment": "well_formed|ill_formed", "reason": "<evidence>", "issues": []
    },
    "conclusion_correct": {
      "assessment": "verified|unverifiable|over_specified|refuted",
      "reason": "<evidence>", "verification": "<check, finding, and criterion claim>"
    }
  },
  "overall_pass": true, "failed_angles": ["<failed angle>"],
  "scoring_tip": "", "agent_confidence": 0.0
}
```