DATA WORLD MODEL: ENHANCING DATA AGENTS THROUGH WORKSPACE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based data agents must complete tasks within workspaces containing heterogeneous data assets. A workspace is initially opaque to the agent and is far too large to fit within a context window, posing significant challenges for data agents. Existing approaches either explore the workspace through costly trial-and-error interactions or construct a fixed semantic layer prior to execution, and both are limited to representing only the information of a workspace. We advance data agents from workspace representation to workspace modeling by introducing the *Data World Model* (DWM), an agent-first model for data workspaces. Following the three constituents of a classical data model, DWM integrates a measured state that captures workspace assets and their relationships, an operation model that predicts how an intended action transforms the current state, and a constraint model that determines whether a generated artifact meets the deliverable conditions the task states. DWM wraps the agent's function calling loop, monitors and processes the interactions between the agent and the workspace, and continuously updates itself based on the interaction content. Experiments across seven benchmarks demonstrate that DWM substantially improves data agents, with the largest gains observed in complex workspaces where direct observation is infeasible[1].

1 INTRODUCTION

Large language model (LLM) agents have evolved from answering isolated questions to autonomously completing complex tasks in real-world environments (Yao et al., 2023; Zhang et al., 2025). In data-centric applications, the environment where such tasks are executed is a *workspace*, which is a directory containing heterogeneous data assets including tables, documents, databases, and intermediate artifacts (Tang et al., 2026; Li et al., 2026). To enable agents to operate in these environments, recent studies have introduced *data agents* that leverage advanced LLMs to perform analytical tasks over heterogeneous data (Hong et al., 2025; Guo et al., 2024; Sun et al., 2025). These agents have shown promising capabilities on individual data tasks. However, as the scale of data expands from isolated datasets to complex workspaces, data agents face a fundamental challenge. To complete a task, an agent must first understand what a workspace contains, how its components are connected, and how potential actions may transform the workspace (Luo et al., 2026).

Understanding a workspace is fundamentally different from understanding an instruction. A natural language instruction is directly available to the agent, while a workspace is initially opaque. Its relevant information is distributed across many files, hidden in complex dependencies, and cannot be revealed through inspecting any single component. In real-world data tasks (such as those studied in AgenticDataBench (Sun et al., 2026)), a single workspace can reach 15 GB in size and contain hundreds of heterogeneous files. Such a scale exceeds the context capacity of current LLMs by orders of magnitude. Therefore, directly placing the entire workspace into the context is infeasible. Meanwhile, allowing an agent to explore the workspace freely forces it to consume most of its interaction budget on discovery before performing the actual task.

This challenge resembles a fundamental problem that the data management community has addressed for decades. When data volumes exceeded the ability of humans to inspect them directly, the field introduced the concept of a *data model*. A data model provides an abstraction that allows

[1] Code is available in the supplementary materials.

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.

users to reason about data without observing every individual record. As defined by Codd (1980), a data model consists of three essential constituents that describe data, transform data, and determine whether a resulting state satisfies given requirements. Such abstraction enables human analysts to operate on large databases without exhaustively examining their contents. As data consumers shift from humans to autonomous agents, we argue that agents also require a data model designed for their own interaction process. This model should not only describe the current workspace but also help agents reason about actions and verify their progress.

Existing data agents expose workspaces through two main paradigms. The first paradigm relies on direct exploration, where agents inspect files, execute operations, and iteratively gather information during task execution (Hong et al., 2025; Guo et al., 2024; Sun et al., 2025). Although flexible, this approach becomes increasingly inefficient as workspace complexity grows because agents repeatedly rediscover information through costly interactions. The second paradigm constructs predefined semantic layers by providing schema-centric representations that include table schemas, column samples, and retrieved descriptions before execution begins (Pourreza & Rafiei, 2023; Gao et al., 2023; Talaei et al., 2024; Cai et al., 2026). While these approaches improve workspace accessibility, they remain limited to representing the workspace at a particular moment. They describe what the data looks like but cannot predict how an action will change the workspace or determine whether the generated artifacts meet the conditions the task states.

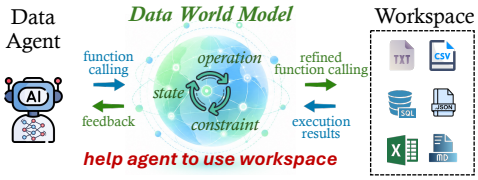


Figure 1: Data World Model helps data agent to use workspace.

In this paper, we advance data agents from workspace representation to workspace modeling by introducing the *Data World Model* (DWM), an agent-first model for data workspaces. DWM enables agents to reason about the current workspace state, anticipate the effects of their actions, and verify whether the final outcome satisfies the task objective. Following the principles of classical data models (Codd, 1980), DWM contains three components. The first component is a *state* that captures factual information about workspace assets, their relationships, and artifacts produced by the agent. The second component is an *operation* that predicts how an intended action transforms the current state into a future state. The third component is a *constraint* that determines whether a resulting state meets the deliverable conditions the task states. As illustrated in Figure 1, DWM is positioned between the data agent and the workspace. It mediates both the forward interaction from agent decisions to workspace operations and the backward feedback from workspace execution results. DWM is initialized by a builder agent that profiles the workspace programmatically and compiles the task request into the conditions the constraint will check. During execution, it continuously records interactions between the agent and the workspace and updates its internal state according to observed outcomes. With DWM, an agent no longer explores a workspace blindly. Instead, it operates through a continuously evolving model that provides guidance, prediction, and verification throughout task execution. In summary, our key contributions are as follows.

- **Data World Model.** We introduce DWM, the first agent-first model of a data workspace that adapts the principles of classical data models to autonomous agents operating under partial workspace observability.
- **Workspace Modeling.** We propose a self-evolving workspace modeling framework that predicts the consequences of agent actions, verifies generated artifacts against task requirements, and updates itself from interaction trajectories without labeled supervision.
- **Strong Performance.** Experiments on seven data agent benchmarks demonstrate that DWM substantially improves the performance of data agents, especially in complex workspaces where direct observation is infeasible and existing context-based approaches become ineffective.

2 Related Work

Data Agents. Data agents aim to complete data-centric tasks autonomously, with capability organized from single-step assistance to long-lived self-governing systems (Luo et al., 2026). Early methods fine-tune LLMs for tabular understanding (Li et al., 2023b), handling individual tasks without acting in an environment. A larger body of work builds workflow-based agents that prompt

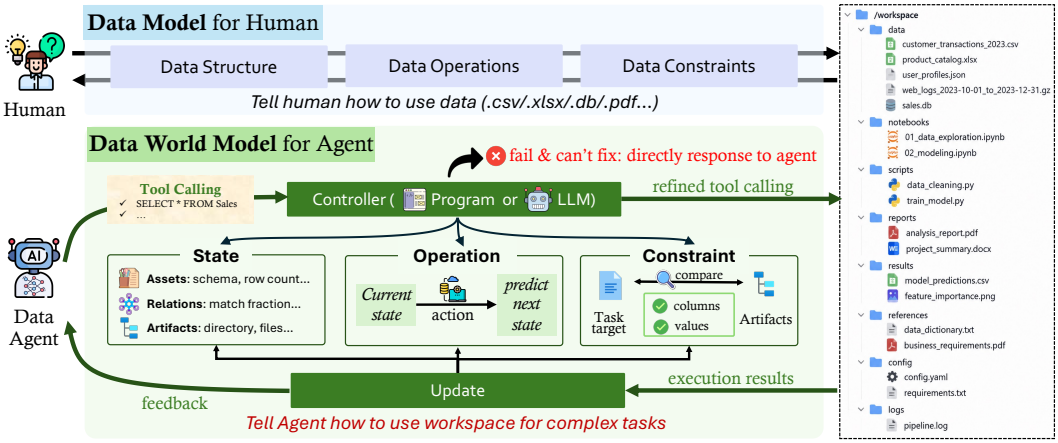


Figure 2: DWM wraps the agent's tool loop at two interception points, so a call whose expected outcome conflicts with the task is turned back before it runs, and a result that reveals a new workspace state updates the model before it reaches the agent.

strong LLMs to decompose a task and execute it step by step (Hong et al., 2025; Guo et al., 2024; Sun et al., 2025), and agentic training internalizes the whole pipeline into one model (Zhang et al., 2025). Benchmarks have moved in the same direction, from single tables toward complete workspaces with cross-file dependencies (Sun et al., 2026; Li et al., 2026; Tang et al., 2026; Wang et al., 2026). A recent survey notes that agents at the lower levels of this ladder still act in an open-loop manner and cannot perceive downstream validation of what they produce (Zhu et al., 2026), which is exactly the regime a workspace task lives in.

Context Construction for Data Agents. Since a workspace cannot be placed into the context, a line of work studies what to put there instead. Text-to-SQL systems select and serialize the schema elements that a question is likely to touch (Pourreza & Rafiei, 2023; Gao et al., 2023; Talaei et al., 2024), and agent frameworks generalize this idea into schema-shaped descriptions of an environment that are reusable across tasks (Cai et al., 2026) or into memory hierarchies that page content in and out of a limited window (Packer et al., 2023). These methods make a workspace readable, yet their common assumption is that the workspace is a static object to be described, so the constructed context is fixed before the first action and carries no statement about what an action will change or about whether the final artifact is acceptable.

World Models for LLM Agents. A world model predicts the next state of an environment given the current state and an action (Ha & Schmidhuber, 2018), and this formulation has been brought to LLM agents. RAP treats the LLM itself as a world model and searches over imagined states during reasoning (Hao et al., 2023), while WebDreamer simulates the outcome of each candidate action on a web page and scores the imagined outcomes before committing to one (Gu et al., 2025). A complementary line lets agents accumulate experience across episodes, through verbal reflection (Shinn et al., 2023), skill libraries (Wang et al., 2023), distilled insights (Zhao et al., 2024), and reusable workflows (Wang et al., 2024). Existing world models for agents describe environments whose state is small enough to be written into natural language, such as a web page or a game, so the state itself is never the bottleneck.

This work advances data agents from workspace representation to workspace modeling. DWM models the workspace with the three constituents of a classical data model and continuously updates its state from the agent's trajectory, so the model predicts how actions will change the workspace and checks the produced artifacts against the conditions the task states.

3 Data World Model

A data model is what allowed people to work with data they could not read exhaustively. An agent faces the same situation for a different reason. It cannot read the workspace exhaustively either, because the data is too large for a context window and completely unknown at the first step. The

Data World Model carries the three constituents of Codd (1980) over to an agent that acts under partial observation.

3.1 Architecture of Data World Model

Transferring the three constituents of a data model to an agent means defining each of them concretely when the underlying object is a workspace of files and the consumer is a language model working with a partial view. This subsection defines each constituent and the guarantee it provides, from a state that a partial view can trust, through an operation that speaks about actions before they run, to a constraint that decides completion without consulting the reference answer. Because the agent changes the workspace as it acts, a fourth mechanism keeps these constituents current throughout the episode (Section 3.3).

State. The state is a structured representation of the workspace built from the workspace itself, with no access to the task answer. A state s decomposes into three components,

$$s = (\mathcal{A},\ \mathcal{R},\ \mathcal{O}), \tag{1}$$

where each component captures a distinct facet of the workspace. *Assets* $\mathcal{A}$ are the data files, each annotated with its schema, row count, and per-column statistics including cardinality and value distribution. *Relations* $\mathcal{R}$ are the cross-asset links, each annotated with the key match fraction and the maximum key multiplicity. *Artifacts* $\mathcal{O}$ are the files the agent has produced, so the state evolves as the agent acts. The relations distinguish a state from a schema catalogue. A match fraction is unobservable from either table in isolation and requires a joint scan, and an inventory of assets alone leaves the agreement between two candidate keys unmeasured. Every entry carries its provenance. Schemas, row counts, and artifact contents come from a full pass over the file and are exact, while key match fractions and multiplicities come from compact sketches and are estimates whose direction of error is known by construction, since a sketch can miss agreeing keys and cannot invent them. Only an entry that is exact, or an estimate whose error direction makes the conclusion safe, is allowed to support a decision that blocks an action.

Operation. A fixed description of existing data carries no information about data an action would produce. The operation fills this gap by predicting the post-action state from the current state and the action the agent is about to execute,

$$\hat{s}_{t+1} = f_{\text{expect}}(s_t,\ a_t), \tag{2}$$

where $\hat{s}_{t+1}$ encodes the expected output schema, row count range, and granularity of the resulting artifact. The prediction is derived from the measured entries of s_t programmatically, with no generative step on this path. When the agent joins two assets, the match fraction f and the maximum key multiplicity m stored in their relation place the output cardinality in the interval $[f \cdot n_{\text{left}},\ m \cdot n_{\text{left}}]$. The two ends of this interval carry different force. A match fraction below one certifies that some left rows have no counterpart and will be dropped, which is a property of the measured keys and holds whatever the agent writes, so it can license a rejection. An upper bound above n_{left} only establishes that duplication is possible, so it is reported to the agent and never blocks the call.

Constraint. The constraint serves for a task the role that integrity rules serve for a database. A language model call compiles the task request into a *target specification* τ that covers the expected artifact type, column set, row count, and ordering, while leaving any dimension unspecified by the request unconstrained. Checking τ against the artifact component $\mathcal{O}$ together with the evidence e_t left by the execution that produced it yields a *residual*,

$$\rho_t = R(\tau,\ s_t.\mathcal{O},\ e_t), \qquad \text{accept}(s_t) \iff \rho_t = \varnothing, \tag{3}$$

where e_t is the text of the issued command together with what that command printed, and R is a deterministic program with two families of conditions. Delivery conditions read τ and $s_t.\mathcal{O}$ and ask whether a file of the required type, columns, shape, and ordering exists. Consistency conditions read $s_t.\mathcal{O}$ and e_t and ask whether the artifact agrees with the numbers the same execution printed. The residual ρ_t is empty when every condition holds and otherwise names the dimensions that failed. The scope of accept follows from what these conditions read. It certifies that the artifact is deliverable in the form the request asks for and that it does not contradict its own execution, and it makes no statement about whether the values inside are the values the task wanted, since τ is compiled from the request and $s_t.\mathcal{O}$ is measured from the produced file, so neither term ever consults the reference answer.

3.2 Integration of Data World Model

A workspace model is useful only when consulted before an action becomes irreversible or an artifact is accepted. DWM therefore wraps the tool loop with Equation equation 2 on the outbound pass and Equation equation 3 on the inbound pass.

Anticipation before Execution. Before each tool call, DWM predicts the post-action state using Equation equation 2. The verification target depends on the action's position in the episode. An intermediate step is evaluated only against invariants that no later action can undo, since a temporary artifact may satisfy no part of τ yet, so a join is rejected only when the measured statistics show that it will drop rows the task requires. The full target τ is enforced for deliverable-writing actions and termination requests, where completion is rejected if the required artifact is missing or $\rho_t \neq \varnothing$. A rejection is issued only when the measured entries of s_t certify the violation, and a prediction that merely looks unpromising is passed through with the concern attached. Rejected calls never reach the workspace, allowing the agent to revise without consuming execution steps.

Reconciliation after Execution. After execution, DWM intercepts the result and reconciles $\mathcal{O}$ with the workspace by registering new files, re-measuring modified artifacts, and removing missing entries. It then computes ρ_t from Equation equation 3 with e_t taken from the command and its printed output, which is where the consistency conditions do their work, since a value the file holds can disagree with the value the same execution printed while the file still carries the required type, columns, and shape. The agent receives the original result together with a structured reconciliation report describing the divergent dimensions, while retaining full control over subsequent actions.

Separation of Programmatic and Generative Computation. Both passes separate measurable conditions from semantic interpretation. Deterministic rules handle all executable checks, including join bounds, artifact existence, schema, row count, shape, ordering, and consistency between artifacts and reported values. The generative component resolves only conditions requiring interpretation, such as mapping natural-language requests to workspace fields, inferring unstated structural requirements, and describing detected inconsistencies. This division places every gate that can block an action on the programmatic side, so all blocking decisions are reproducible once τ and the measured state are fixed, and none of them depends on a sampled token. The guarantee is conditional on the correctness of τ, since compiling τ is itself a generative step, and τ is revised during the episode only in the monotone direction described in Section 3.3.

3.3 Continual Improvement of Data World Model

The mechanism in Section 3.2 relies on a state that remains aligned with the workspace throughout the episode. Initialization constructs the initial state before execution, and Equation equation 4 continuously refines it after each intercepted action and observation.

Initialization. Initialization performs a single profiling pass over the workspace and one generative step to compile the target specification τ. The profiling pass extracts measurable properties required for s_0 (Equation equation 1), while the builder agent maps the task request and observed state into τ, including requirements that cannot be inferred from measurement alone. Since the builder reasons only over the profiled state and issues no command against the workspace, initialization costs one scan per file and one language model call whatever the task asks for.

Update at Every Interception. As the agent changes the workspace, the model updates through

$$s_{t+1} = f_{\text{update}}(s_t, a_t, o_t), \tag{4}$$

where a_t is the intercepted action and o_t is its observed outcome. On the outbound pass, the update loads the measurements needed to evaluate the intended action, and on the inbound pass it brings the model back into agreement with the observed workspace changes. These state updates are programmatic, so every entry a gate reads traces back to a measurement.

The target specification τ is refined at the same interception points, and the refinement is monotone. A generative step may resolve a dimension the request left open or add a condition the request implies, and it may not weaken or remove a condition already compiled. A produced artifact therefore serves as evidence about what the request meant and never as grounds for accepting itself. The evolving state and specification are therefore refined along the same trajectory that guides the agent's actions.

Table 2: **AgenticDataBench.** Average score.

System	Score	Δ
Previous Systems		
DA-Agent + Kimi-K2.5	39.08	
Codex + Qwen3.5	41.56	
Claude Code + Sonnet 4.6	43.62	
Baseline (GPT-5.6-Sol)	41.82	
+ Semantic Layer	42.40	+0.58
+ DWM	**51.57**	**+9.75**
Baseline (Qwen-3.8-Max)	29.02	
+ Semantic Layer	29.50	+0.48
+ DWM	**34.72**	**+5.70**

Table 3: **DataSpace.** Task accuracy.

System	Score	Δ
Previous Systems		
DS-Agent + Claude Sonnet 5	32.93	
DS-Agent + MiMo-V2.5	39.27	
DS-Agent + Kimi K3	53.41	
Baseline (GPT-5.6-Sol)	48.33	
+ Semantic Layer	49.20	+0.87
+ DWM	**71.67**	**+23.33**
Baseline (Qwen-3.8-Max)	35.47	
+ Semantic Layer	36.03	+0.56
+ DWM	**49.80**	**+14.33**

4 EXPERIMENTS

4.1 BENCHMARKS

We evaluate DWM on seven data agent benchmarks that span diverse workspaces and scoring criteria. **AgenticDataBench** (Sun et al., 2026) ships workspaces of up to 15 GB across 14 domains and scores each task with an executable function. **DataSpace** (Li et al., 2026) requires a submitted table to match the reference cell by cell. **BIRD** (Li et al., 2023a) scores generated SQL by execution accuracy over real databases. **FDABench** (Wang et al., 2026) evaluates analytical queries over mixed structured and unstructured sources. **InsightBench** (Sahu et al., 2025) evaluates end-to-end business analytics where the agent formulates its own questions. **DDR-Bench** (Liu et al., 2026) evaluates long-horizon research through interdependent questions over annual filings. **Workspace-Bench** (Tang et al., 2026) scores rubric-graded multi-file tasks at three difficulty levels. Appendix A reports the workspace statistics of all seven benchmarks.

4.2 EXPERIMENTAL SETUP

Table 1: **Workspace-Bench.** Rubric pass rate.

System	Score	Δ
Previous Systems		
DeepAgent + GPT-5.4	36.20	
OpenClaw + Kimi-2.5	44.50	
Hermes + Kimi-2.5	49.10	
Baseline (GPT-5.6-Sol)	47.97	
+ Semantic Layer	48.50	+0.53
+ DWM	**55.10**	**+7.13**
Baseline (Qwen-3.8-Max)	35.62	
+ Semantic Layer	36.08	+0.46
+ DWM	**40.27**	**+4.65**

We conduct experiments on two frozen backbones to separate the workspace model's contribution from the language model's. `GPT-5.6-Sol` serves as the primary backbone, and `Qwen-3.8-Max` tests whether the gains transfer across backbones. For each backbone we compare the backbone alone (Baseline), the backbone augmented with a static semantic layer that serializes profiled schema into the prompt (+Semantic Layer), and the backbone wrapped by the full DWM (+DWM). The semantic layer receives the same profiled measurements as DWM but delivers them as a fixed prompt prefix unchanged throughout the trajectory, isolating the contribution of modeling itself. All language model calls inside DWM use the same backbone as the agent, so every difference between configurations comes from the workspace model. We adopt the official evaluation protocol of each benchmark without modification (Sun et al., 2026; Li et al., 2026; Tang et al., 2026), and every score in Tables 2–1 is the average of paired runs under identical settings. The task-level and episode-level counts reported in Section 5 come from a single reference run per configuration, since a count of how many episodes took a given path is defined for one run and not for an average of two. Reference systems from official leaderboards are listed for context.

4.3 MAIN RESULTS

Tables 1–7 report results on all seven benchmarks. DWM improves the frozen backbone on every benchmark, and the pattern holds on the weaker backbone `Qwen-3.8-Max`, where the gain is proportionally larger on the benchmarks that demand the most discovery. The semantic layer,

Table 4: **BIRD.** Execution accuracy.

System	Score	Δ
Previous Systems		
Llama3-70B-Instruct	40.80	
GPT-4	47.80	
TA + GPT-4	58.00	
Baseline (GPT-5.6-Sol)	51.00	
+ Semantic Layer	52.00	+1.00
+ DWM	**57.00**	**+6.00**
Baseline (Qwen-3.8-Max)	68.20	
+ Semantic Layer	68.80	+0.60
+ DWM	**69.40**	**+1.20**

Table 5: **FDABench.** Multiple-choice accuracy.

System	Score	Δ
Previous Systems		
Tool-Use + Gemini-2.5-Pro	15.20	
Tool-Use + Claude-Sonnet-4	15.30	
DeepSeek-V3.2	22.50	
Baseline (GPT-5.6-Sol)	10.30	
+ Semantic Layer	10.90	+0.60
+ DWM	**28.48**	**+18.18**
Baseline (Qwen-3.8-Max)	7.88	
+ Semantic Layer	8.48	+0.60
+ DWM	**18.79**	**+10.91**

Table 6: **InsightBench.** G-Eval score.

System	Score	Δ
Previous Systems		
Pandas Agent	39.13	
AgentPoirot	43.11	
DataSeer	51.17	
Baseline (GPT-5.6-Sol)	54.26	
+ Semantic Layer	54.80	+0.54
+ DWM	**58.18**	**+3.92**
Baseline (Qwen-3.8-Max)	36.73	
+ Semantic Layer	37.20	+0.47
+ DWM	**40.68**	**+3.95**

Table 7: **DDR-Bench.** Overall score.

System	Score	Δ
Previous Systems		
GPT-5.2	42.99	
DeepSeek-V3.2	49.12	
Claude 4.5 Sonnet	69.10	
Baseline (GPT-5.6-Sol)	66.32	
+ Semantic Layer	64.00	−2.32
+ DWM	**85.99**	**+19.67**
Baseline (Qwen-3.8-Max)	58.67	
+ Semantic Layer	59.00	+0.33
+ DWM	**69.33**	**+10.66**

which receives identical profiled measurements but serializes them once into the prompt, moves no benchmark by more than one point and is negative on DDR-Bench. The two configurations differ only in whether the measurements are delivered as a fixed description or maintained as a model that anticipates actions and checks artifacts, so the gap measures what modeling contributes beyond the measurements themselves. What an agent needs from its workspace is a model it can query, update, and check against, and a description of the same measurements leaves that need unmet.

The gain is largest where the workspace exceeds what the agent can place into its context, covering the benchmarks with the most sources per task and the largest files in Table 10, and it is smallest where the workspace is a single table the agent can read in full. Section 5.4 takes up this separation and the deployment criterion it yields.

5 Analyses

We conducted extensive analyses to understand the effectiveness of DWM. Unless otherwise specified, all analyses were performed using GPT-5.6-Sol as the backbone on DataSpace (DS for short) and Workspace-Bench (WB for short).

Table 8: Cumulative ablation on DataSpace and Workspace-Bench. Each row removes one further component from the row above, so each Δ is a marginal contribution along a single fixed order.

Configuration	DS	Δ	WB	Δ
Full DWM	**71.67**		**55.10**	
− Update	67.25	−4.42	53.70	−1.40
− Operation	62.50	−4.75	51.90	−1.80
− Constraint	52.95	−9.55	49.60	−2.30
− State	48.33	−4.62	47.97	−1.63

5.1 Ablation Study

Table 8 removes one component at a time from the full DWM on DataSpace and Workspace-Bench, the two benchmarks whose workspaces span the widest range of files and formats. The removal order is fixed by dependency, since the update maintains the state, the operation predicts from it, and the constraint checks artifacts registered in it, so each Δ reports what one component adds on top of the components still present below it. The constraint carries the largest single drop (-9.55 on DataSpace), because it is the only component that can prevent a

wrong submission, and scoring the submitted table against the reference penalizes that submission the moment it lands. The operation adds the ability to anticipate an outcome before an action runs, and the continual update mechanism keeps that anticipation current as the workspace changes. The state carries the last remaining structure. Removing it recovers the baseline, since these workspaces cannot be navigated without a measured starting point. One comparison falls outside this chain and is worth reading directly. With the constraint, operation, and update all removed, what remains is the state alone, which still clears the semantic layer by 3.75 points while both are built from identical measurements. The difference lies in access. The state answers a query at the moment the agent needs an answer, while the semantic layer commits to one serialization before the first action and therefore carries the measurements chosen before the task revealed which of them mattered.

5.2 Effect of DWM on Failure Composition

This analysis asks which failures the method removes and which it leaves in place. Figure 3 decomposes the 60 DataSpace tasks of one reference run by whether the agent produced a valid artifact and whether it matched the reference, placing each task into exactly one of three groups. The baseline ends 12 tasks with no valid artifact and DWM reduces this to 3, while wrong artifacts decrease from 19 to 14. Both failure groups shrink, with the no-artifact group accounting for most of the 14 additional correct submissions. A static representation cannot reach this failure mode, since a workspace description does not reveal what the agent has produced, and only a model that tracks artifacts as part of its evolving state can detect a missing output. The shift in the no-artifact group follows the designed mechanism, since a task that would have ended without a file is redirected with the missing artifact named, as illustrated by one episode in Appendix B. The boundary between the two failure groups marks the scope of the method. DWM addresses most omission failures and leaves a small residue of no-artifact cases, while computation failures change little and remain largely determined by the backbone.

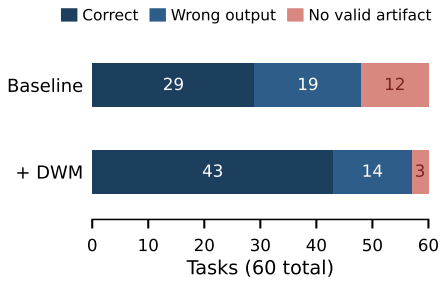


Figure 3: DWM converts most no-artifact failures into correct submissions, while wrong artifacts move far less.

5.3 Precision of Pre-Execution Anticipation

Table 9 isolates the operation constituent by measuring how often the pre-execution anticipation fires and what happens afterwards. On DataSpace, the anticipation rejects at least one call in most episodes, and shadow execution confirms 92.5% of those rejections to violate the condition the rejection cited, which is an invariant for an intermediate call and the full target specification for a deliverable write or a termination request. The high precision follows from the design. The anticipation checks cardinality bounds and schema consistency against measured relations, so its decisions are reproducible once the target specification is fixed, and a gate is thereby more reliable than the action it blocks. After a rejection, the agent modifies its approach and 84.2% of those episodes end with a correct artifact, compared to 44.7% for the same tasks under the baseline. Both figures are computed over the same episodes, which were selected by the anticipation having fired in them, so the comparison describes tasks where the operation model has something to say and does not estimate an effect on a task drawn at random. On Workspace-Bench the anticipation fires less frequently because the workspace structure is shallower, yet the precision and recovery rates remain comparable. The episodes that fail to recover after a correct warning are the ones whose remaining difficulty is computational, which Section 5.4 places outside the scope of the method.

Table 9: Pre-execution anticipation on DataSpace and Workspace-Bench. The three rates carry different denominators, namely all episodes for the first row, rejected calls for precision, and episodes containing at least one rejection for recovery. Precision is measured by shadow-executing rejected calls and checking whether the outcome violates the condition the rejection cited.

	DS	**WB**
Episodes with anticipation	63.3%	41.0%
Avg. rejections / episode	2.1	1.5
Rejection precision	92.5%	89.7%
Post-rejection recovery	84.2%	78.0%

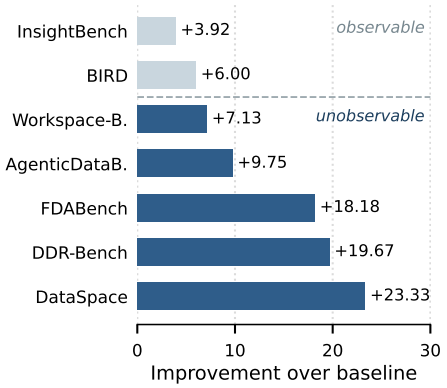


Figure 4: The gain tracks whether one inspection command can place the properties a task depends on into the agent's context.

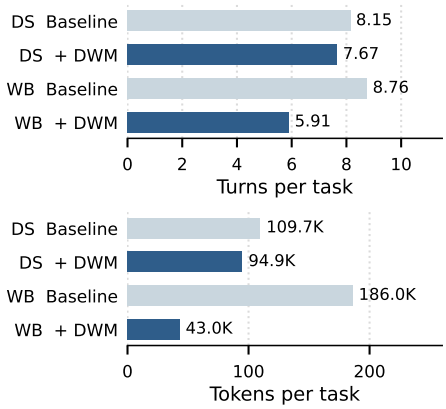


Figure 5: Modeling shortens the episode while raising accuracy, cutting both interaction turns and tokens.

5.4 Gain Scales with Workspace Unobservability

We group the seven benchmarks by whether one inspection command can place the properties a task depends on into the agent's context. A benchmark whose questions are answered from a schema belongs to the observable group even when its files are large, while a benchmark whose tasks turn on cross-file agreement or on artifacts scattered across a directory does not. As shown in Figure 4, the improvement separates sharply along this line, with the observable group carrying a small band and the unobservable group a substantially wider one. Within the unobservable group the gain does not track raw data volume, since the largest workspace is also the one whose remaining errors are dominated by computation alone. The quantity DWM addresses is therefore observability, which yields a criterion for deployment, since modeling is decisive when the workspace contains properties the agent cannot acquire from a single read. This distinction also explains why the semantic layer fails to close the gap. A static description grows linearly with the workspace until it no longer fits the context, and the very benchmarks where it overflows are the ones where modeling delivers the largest gain.

5.5 Reduction of Interaction Cost

Figure 5 reports interaction turns and token consumption per episode on DataSpace and Workspace-Bench. The average episode shrinks by a small margin on DataSpace and by roughly one third on Workspace-Bench, because the baseline agent spends most of its Workspace-Bench turns on discovery commands that DWM replaces with the initial state. Token cost falls more steeply than turn count, by more than three quarters on Workspace-Bench, because the turns DWM removes are the directory listings and file dumps whose outputs are the longest in the trajectory. The reported token counts already include every API call DWM makes, covering the builder agent, the anticipation checks, and the reconciliation reports, so the reduction is net of all overhead the model introduces and identifies discovery as the dominant cost in baseline trajectories. Accuracy and cost therefore move together, because the agent acts on measured structure and no longer spends its budget inferring that structure by trial.

6 Conclusion

We have advanced data agents from workspace representation to workspace modeling. The Data World Model (DWM) extends the three constituents (state, operation, constraint) of a classical data model to agents operating under partial workspace observability. By wrapping the agent's tool loop, DWM ensures that both outgoing actions and incoming observations are mediated by the model, which continuously refines itself from the agent's execution trajectory. Experiments across seven benchmarks show that DWM substantially improves the performance of data agents in workspaces, with the improvements becoming more pronounced as workspace complexity increases.

AI USE STATEMENT

In this work, we used generative AI tools only for final-stage language polishing and grammar refinement of the manuscript. We have not used generative AI tools for generating research ideas, designing methods, conducting experiments, implementing code, analyzing data, interpreting results, or producing scientific claims, and other generative AI-assisted tasks requiring disclosure are not applicable to this work. We have reviewed all AI-assisted revisions to ensure their correctness and consistency with the authors' intended content. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of this work by providing detailed descriptions of the proposed method, experimental settings, implementation details, and evaluation protocols in the main paper and supplementary materials. The supplementary materials include the source code required to reproduce our experiments, and we will publicly release the code after the review process to further facilitate verification and future research. The content of this paper can be completely reproduced.

REFERENCES

Kuntai Cai, Juncheng Liu, Xianglin Yang, Zhaojie Niu, Xiaokui Xiao, and Xing Chen. Autocontext: Instance-level context learning for llm agents, 2026. URL https://arxiv.org/abs/2510.02369.

E. F. Codd. Data models in database management. In *Proceedings of the 1980 Workshop on Data Abstraction, Databases and Conceptual Modeling*, pp. 112–114, 1980.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*, 2023.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your llm secretly a world model of the internet? model-based planning for web agents, 2025. URL https://arxiv.org/abs/2411.06559.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Automated data science by empowering large language models with case-based reasoning. *arXiv preprint arXiv:2402.17453*, 2024.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Yongxin Ni, Zhibin Gou, Zongze Xu, Yuyu Luo, and Chenglin Wu. Data interpreter: An LLM agent for data science. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19796–19821, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1016. URL https://aclanthology.org/2025.findings-acl.1016/.

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

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls, 2023a. URL https://arxiv.org/abs/2305.03111.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263*, 2023b.

Wei Liu, Peijie Yu, Michele Orini, Yali Du, and Yulan He. Hunt instead of wait: Evaluating deep data research on large language models, 2026. URL https://arxiv.org/abs/2602.02039.

Yuyu Luo, Guoliang Li, Ju Fan, and Nan Tang. Data agents: Levels, state of the art, and open problems, 2026. URL https://arxiv.org/abs/2602.04261.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*, 2023.

Gaurav Sahu, Abhay Puri, Juan Rodriguez, Amirhossein Abaskohi, Mohammad Chegini, Alexandre Drouin, Perouz Taslakian, Valentina Zantedeschi, Alexandre Lacoste, David Vazquez, Nicolas Chapados, Christopher Pal, Sai Rajeswar Mudumba, and Issam Hadj Laradji. Insightbench: Evaluating business analytics agents through multi-step insight generation, 2025. URL https://arxiv.org/abs/2407.06423.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. Lambda: A large model based data agent. *Journal of the American Statistical Association*, 121 (553):1–13, 2025. ISSN 1537-274X. doi: 10.1080/01621459.2025.2510000. URL http://dx.doi.org/10.1080/01621459.2025.2510000.

Zhaoyan Sun, Shan Zhong, Daizhou Wen, Jiaxing Han, Guoliang Li, Ying Yan, Peng Zhang, Yu Su, Xiang Qi, Baolin Sun, Chengyuan Yang, Tao Fang, and Huaiyu Ruan. Agenticdatabench: A comprehensive benchmark for data agents. *arXiv preprint arXiv:2607.01647*, 2026.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, GuoLiang Li, Jihua Kang, and Fan Wu. Workspace-bench 1.0: Benchmarking ai agents on workspace tasks with large-scale file dependencies, 2026. URL https://arxiv.org/abs/2605.03596.

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

Ziting Wang, Shize Zhang, Haitao Yuan, Jinwei Zhu, Wei Dong, and Gao Cong. Fdabench: A benchmark for data agents on analytical queries over heterogeneous data. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 9985–9996. ACM, August 2026. doi: 10.1145/3770855.3817454. URL http://dx.doi.org/10.1145/3770855.3817454.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. Deepanalyze: Agentic large language models for autonomous data science, 2025. URL https://arxiv.org/abs/2510.16872.

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

Yizhang Zhu, Liangwei Wang, Chenyu Yang, Xiaotian Lin, Boyan Li, Wei Zhou, Xinyu Liu, Zhangyang Peng, Tianqi Luo, Yu Li, Chengliang Chai, Chong Chen, Shimin Di, Ju Fan, Ji Sun, Nan Tang, Fugee Tsung, Jiannan Wang, Chenglin Wu, Yanwei Xu, Shaolei Zhang, Yong Zhang, Xuanhe Zhou, Guoliang Li, and Yuyu Luo. A survey of data agents: Emerging paradigm or overstated hype?, 2026. URL https://arxiv.org/abs/2510.23587.

A Detailed Introduction of Workspace Statistics

Table 10 summarizes the workspaces that each benchmark provides to the agent. The two quantities that matter for this paper are how much data a task ships and in how many sources it arrives, since together they decide whether an agent can place the workspace into its own context. For benchmarks that use databases, we count each table inside a database as a separate source, since the agent must discover and navigate each table independently. BIRD and FDABench draw on the same database collection, and FDABench adds per-task unstructured documents that the agent must locate and read alongside the tables.

Table 10: Workspace statistics of the seven benchmarks. Sources per workspace counts every file or database table the agent must navigate, and median rows is reported for the benchmarks whose sources are tabular.

Benchmark	Sources / workspace (# rows)	Largest file	File types
AgenticDataBench	1–18 (12.4K)	3.3 GB	CSV, JSON, Parquet, SQLite, XLSX
DataSpace	5–26 (2.0K)	82 MB	CSV, JSON, SQLite, PDF, Markdown
BIRD	3–14 tables (1.5K)	598 MB	SQLite
FDABench	3–14 tables and documents	598 MB	SQLite, TXT, Markdown
InsightBench	1 (0.5K)	210 KB	CSV
DDR-Bench	10 tables (3.0M)	1.1 GB	SQLite
Workspace-Bench	2,136–11,698 files	689 MB	XLSX, PDF, DOCX, CSV, JSON, TXT

B Case Study

We trace a single DataSpace task to show how the reconciliation pass catches a concrete error during execution. The task asks how many trading days a particular stock had total daily turnover exceeding 100 million yuan, and the answer is 42. The workspace contains 22 files spanning CSV tables, JSON documents, Markdown reports, and a SQLite database. Among the structured sources, a table named `qt_tradingcapitalflow` carries a schema that appears directly relevant to the question, yet it holds zero rows for the target stock. The answer can only be obtained from a 50 KB Markdown document whose numeric fields are prose-encoded. The target entity appears under five different surface forms across the sources, so no single join key resolves it. The turnover threshold sits within 0.5% of the minimum qualifying amount in the data, making the count sensitive to boundary handling.

The agent navigates the workspace competently. It checks the database first, discovers that the target stock has zero rows in the seemingly relevant table, reads the Markdown document in full,

[2]The original task and agent output are in Chinese. We translate the column headers and stock name here for readability; the numerical values are unchanged.

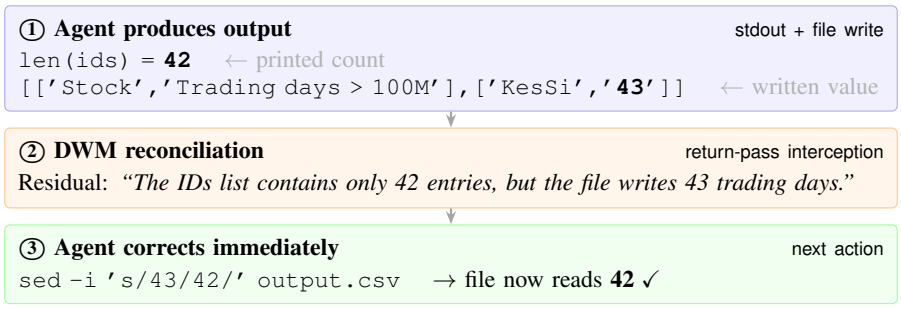


Figure 6: The reconciliation pass detects that the agent's computed value (42) disagrees with the value written to the output file (43) and attaches a one-sentence residual. The agent fixes the file in its next action.[2]

and extracts all 42 matching trading day identifiers with a Python regex. The error occurs in the same command that computes the correct count. The agent's script calls `print(len(ids))` and receives 42 on stdout, but then writes the output file with a heredoc that hard-codes the value 43. The agent's own assertion checks only the shape of the CSV (two rows by two columns) and does not inspect cell values, so the off-by-one passes silently.

This is the setting in which the reconciliation pass fires, since the return-pass interception (Section 3.2) triggers when a new artifact appears and the preceding command was non-read-only. The model computes the residual ρ_t via Equation equation 3, whose consistency conditions compare the registered artifact against the execution record e_t of the command that produced it, and detects that the numeric value in the file disagrees with the value printed during the same execution. A one-sentence residual is attached to the tool result the agent receives, and no other action intervenes between the residual and the fix shown in Step ③ of Figure 6.

This is the design intent of the return pass. The model does not re-derive the answer but checks whether what the agent wrote is consistent with what the agent computed. The anticipation pass stayed silent on all ten actions in this episode, and the submission guard verified that a file was present before allowing the submit call, so both other constituents contributed precision without interfering.