Towards a Context Layer for Self-Improving Data Agents

Till Döhmen
MotherDuck
Netherlands
till@motherduck.com

Jacob Matson
MotherDuck
USA
matson@motherduck.com

Jordan Tigani
MotherDuck
USA
jordan@motherduck.com

Abstract

Data agents that autonomously query and analyze structured data struggle with accuracy when they lack organizational context: which table is canonical, how metrics are defined, which join patterns are correct. We present results from a benchmarking study on the DABstep text-to-SQL benchmark, where providing domain documentation and targeted data modeling (views, macros) improved hard-question accuracy from 29.8% to 93.2% while reducing cost by 62%. Building on these findings, we describe a *context layer*: a data-platform-native knowledge store where agents both consume and contribute organizational knowledge, and which serves as the foundation for targeted data engineering that we envision can eventually be largely automated. We argue that the key challenges are bridging the gap between personal and organizational memory, building effective agentic retrieval beyond naive RAG, and enabling organizations to evaluate their agents against their own data through self-serving evals. As models converge in capability, the differentiator for data agents will increasingly be the context they have accumulated, not which model they call.

Keywords

data agents, context engineering, text-to-SQL, agentic memory, retrieval, self-improvement

1 Introduction

Today's data systems were designed for a small number of careful human operators writing hand-crafted SQL. A growing share of analytical workloads is now delegated to AI agents [13] that translate natural-language questions into SQL, build visualizations, and orchestrate data pipelines. This shift exposes a critical gap: agents lack the organizational context that human analysts accumulate over years. Which table is the source of truth? How is "revenue" computed? Which filters are always required? Yet even providing context is only effective to a certain degree; the last mile of accuracy requires targeted data modeling that encodes complex logic directly in the database.

The industry has recognized this context gap. a16z calls context "the key missing ingredient" for data agents [11]. OpenAI built a six-layer context system for their in-house data agent [18]. GitHub Copilot shipped cross-agent memory [9]. Yet these solutions are either proprietary, generic (unaware of data models and organizational permissions), or both.

This paper makes three contributions. First, we provide background on context for data agents, from semantic layers to agentic memory, and argue that the problem is not adequately served by existing approaches (§2). Second, we present benchmarking results showing that domain documentation and targeted data modeling have more impact on agent accuracy than the choice of model (§3). Third, we describe a possible design of a *context layer* and identify the key open problems: bridging personal to organizational memory in a multi-user setting, building effective agentic retrieval, and enabling domain-specific evaluation (§4). Finally, we outline a vision for the fully agentic data stack (§5).

SAO Workshop at CAIS'26, San Jose, California, USA
.

2 Background

2.1 Semantic Layers

Semantic layers act largely as an intermediate translation layer that guarantees semantic correctness of queries: business metrics, dimensions, and relationships are defined in a structured markup language, and a query engine translates queries formulated in the semantic layer's own query language into SQL (e.g. dbt Semantic Layer [6], Cube [5], Malloy [14]). Implementations vary and there is a push toward standardization [17], but the basic principle remains the same.

In practice, many questions in an agentic text-to-SQL context are exploratory and go beyond pre-defined metrics. Business definitions evolve continuously, and maintaining a semantic layer that reflects reality is cumbersome, as we have argued previously [16]. We argue that a better division of labor is: things that are known and stable should be modeled on the database layer itself (as views, macros, and comments that agents discover through standard schema exploration), while everything beyond should be captured in a self-improving memory system that we call a context layer, optimized for continuous learning and agentic retrieval.

2.2 Agentic Memory

The AI community has converged on the idea that agents need persistent memory, and the field has matured rapidly from 2024–2026. **General-purpose memory systems** like Mem0 [4], Hindsight [12], and Cognee [15] are converging on similar architectures: hybrid storage (vector + graph), multi-strategy retrieval with reranking, explicit temporal modeling, and active memory consolidation. Multi-strategy retrieval (semantic, keyword, graph traversal, temporal) is the single biggest performance differentiator, and temporal reasoning remains the hardest unsolved capability. General-purpose memory, however, does not adequately address the needs of data agents. For example, knowledge lifecycles are domain-specific: in code, a memory should expire when the cited file is deleted or move when it is renamed; in data, knowledge must stay in step-lock with schema evolution and decay when tables are deprecated. Access control is also domain-specific: code memory should mirror repository permissions, data memory should mirror data warehouse permissions. Generic memory systems have no mechanism for either.

We see domain-specific memory systems emerging in the SWE domain, like GitHub Copilot's cross-agent memory [9], which stores observations with citations to code locations and verifies them in real-time against the current branch, tying knowledge decay to the code objects themselves. And we see organization-specific solutions for data agent memory emerging across the industry. One widely covered example is OpenAI's data context layer [18], which combines table lineage, human annotations, code-level enrichment from pipeline source, institutional documents, a personal/global memory layer, and live runtime queries, all tightly coupled to their 70k-table warehouse and permissions model.

We see an open need for a memory architecture that is specific enough to address the domain needs of data agents (*object-scoped*, tied to databases, tables, and columns; *organizationally permissioned*; *trust-differentiated*, where an admin-endorsed definition outranks a personal note), yet generic enough to not be tied to a single organization's stack.

3 Benchmarking Context Engineering

We conducted a simple experiment, using a challenging text-to-SQL benchmark dataset, to motivate the need for both accurate domain context as well as targeted data modeling.

3.1 Experimental Setup

For the experiment, we used DABstep [7], a suite of 450 payment-processing questions[1] that comes with domain documentation and requires understanding of a complex fee-matching system with 9 join dimensions, wildcard joins, and hypothetical simulations. The original paper reports that "even the best agent achieves only 14.55% accuracy on the hardest tasks" with default setups.

All configurations use a benchmark-agnostic agent framework equipped with 3 MCP [1] tools (SQL execution, list tables, list columns) and a turn limit of 15 tool calls. We used a low-cost state-of-the-art LLM, Gemini 3 Flash [10], and varied two independent dimensions:

(1) **Context delivery:** How domain knowledge reaches the agent. We tested three approaches:
   (a) *Annotations*: hand-crafted column comments plus auto-generated statistical profiles, discoverable via the list-columns tool.
   (b) *BM25 retrieval*: BM25 search over chunked domain documentation and SQL pattern examples, injecting relevant fragments into the prompt at query time.
   (c) *Full-context prompt*: all domain documentation, term mappings, and SQL patterns injected into the system prompt in full.

(2) **Data modeling:** Whether complex join logic is pre-computed as database objects. Views encode the 9-dimension fee-matching join; DuckDB table macros encapsulate parameterized calculations as callable functions. These were developed iteratively through qualitative evaluation of benchmark results, with a human and agent jointly identifying the highest-leverage views and macros to add. This represents an upper-bound result: a realistic agent-assisted modeling workflow rather than a fully automated procedure. The effort involved approximately 10 iterations of inspecting failure cases and encoding the missing logic as database objects.

[1] DABstep contains 460 questions total, but we excluded the 10 development questions that ship with gold answers, as they are likely present in model training data.

3.2 Results

Table 1 shows accuracy across combinations of the two dimensions. The best configuration (full-context prompt + views/macros) achieves 93.2% on hard questions. While this would rank first on the public validated DABstep leaderboard as of March 2026, we acknowledge that while we aim to keep validation aligned with the official ranking, it is not a verified result at the time of writing.

Table 1: Accuracy on DABstep by context retrieval and data modeling (Gemini 3 Flash).

Context retrieval	Data modeling	Easy	Hard	Cost
None (baseline)	×	79.2%	29.8%	$11.35
Column annotations	×	88.9%	30.1%	$10.43
Full-context prompt	×	90.3%	82.7%	$6.39
Full-context prompt	✓	93.1%	93.2%	$4.35
BM25 retrieval	✓	91.7%	86.6%	$5.96

3.3 Key Findings

The first finding concerns context delivery. Column annotations alone are insufficient: per-column comments added only 0.3% on hard questions because they cannot express cross-table logic. Retrieving chunks of domain documentation via BM25 performs better but remains lossy: keyword matching sometimes filters out patterns the model would have found with full visibility, and BM25 retrieval (86.6%) trails the full-context prompt (93.2%) by 6.6 points while costing 27% more (due to a higher number of turns). The full-context prompt is effective because it lets the model make its own relevance judgments, but dumping all documentation into a single prompt is only feasible because this benchmark has a bounded domain. For real-world organizations with thousands of tables, more sophisticated retrieval methods become necessary.

The second finding is that *data modeling* makes a significant contribution to the results. The fundamental bottleneck is reconstruction: LLMs can read business rules in prose but cannot reliably assemble them into complex multi-table joins under turn pressure. Views and macros solve this by moving logic to DDL time (+10.5 points when adding macros to the full-context prompt). Without data modeling, prompt engineering hits a ceiling at 82.7% on hard questions regardless of how well instructions are written. An open question is how to arrive at signals that inform data modeling decisions. This motivates a notion of trust and canonicalness in the context layer, which can form the basis for automated data modeling.

4 A Context Layer for Data Agents

Our benchmark shows that both rich context and targeted data modeling significantly improve agent accuracy. Three specific gaps motivate the context layer design: (1) column annotations cannot express cross-table logic, suggesting that context must be richer

than per-object comments; (2) BM25 retrieval over documentation chunks is lossy, suggesting that retrieval must be agentic rather than keyword-based; and (3) the most impactful views and macros required iterative human insight, suggesting that the system should accumulate such insight over time. The question is how to establish context, make it retrievable, and do so in a way that can also inform data modeling decisions. An important starting point is to collect signals about how data is used and in which context it is used across the organization. In many cases, such knowledge is not explicitly written down and does not fit in a single prompt at scale. Inspired by the agentic memory systems surveyed in §2, we propose one possible design for a self-maintaining *context layer* that reframes context as a *discovery problem* rather than a *definition problem*. We suggest such a system requires the following characteristics:

- **Object-scoped and multi-tenant:** Context must be tied to data objects (tables, columns, schemas) and scoped to users, roles, or the organization, inheriting the database's access control model. The system must distinguish personal observations from organizational facts, with mechanisms to promote context across visibility levels.
- **Self-curating:** The system must consolidate, deduplicate, and validate raw context, accumulating from agent interactions and validating through evaluation to prevent noise and improve over time.
- **Agentically retrievable:** Delivery must go beyond pure search, supporting agentic retrieval methods like progressive disclosure and task-aware retrieval.

4.1 Fragments

The core unit of our envisioned context layer is the **Fragment**: a plain-text piece of knowledge with metadata including structured references to database objects (databases, schemas, tables, columns) and other artifacts. Fragments can represent anything from a formal metric definition to an informal observation about data quality. What distinguishes them from unstructured documents is that they are *object-scoped* and *access-controlled*: each fragment is anchored to a specific data asset or set of data assets (tables, columns, etc.) and scoped to a user, role, or organization. This enables both better surfacing during retrieval and easier maintenance as schemas evolve. For example, a fragment might carry the title "`eur_amount requires deduplication before aggregation`," with content explaining the business rule, a reference to `db.main.payments.eur_amount`, trust status *endorsed*, and organizational scope.

Fragments are created through multiple channels (Figure 1). They can be *explicitly authored* by humans (e.g. metric definitions, business rules). *Ambient capture* records observations from agent conversations: every time an agent discovers a join pattern, corrects a misunderstanding, or receives feedback, the observation is stored as a fragment. *Inferred context* is derived directly from the data platform: query history mining [8], statistical profiling, and schema change detection produce knowledge that no human would think to write. *External sources* such as dbt model documentation, Slack threads, and Notion pages contain organizational knowledge that already exists but is scattered and inaccessible to agents. In all cases, fragment creation is typically LLM-mediated: a synthesis step extracts structured knowledge from raw material and anchors it to the relevant database objects.

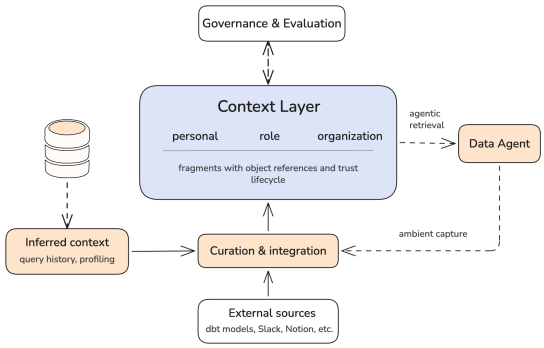

Figure 1: Architecture of the context layer. Raw observations from agent conversations (ambient capture), inferred context from the data platform (query history, profiling), and external sources (dbt models, documentation tools) all flow through a curation and integration step before entering the context layer. Human governance and evaluation form a bidirectional loop that promotes, endorses, and validates knowledge.

4.2 Self-Curation

All input channels produce raw material that must pass through *curation and integration* before becoming trusted context (Figure 1). We envision a trust lifecycle: raw observations are unverified (rumors), consolidated and deduplicated context has been validated (lore), and human-endorsed definitions carry organizational authority (canon). Without this distinction, the context layer would fill with noise and contradictions. Fragments are scoped to personal or organizational visibility, and a *promotion* mechanism moves context between scopes.

Curation operates at two levels. *Personal curation* deduplicates and consolidates a single user's fragments, primarily through LLM-driven consolidation [2], to filter out noise and duplicates and produce high-signal context. Because fragments are coupled to a live schema, they can also be flagged or decayed automatically when referenced objects are deprecated or renamed. *Org integration* determines whether personal fragments should be promoted to organizational scope. Beyond object-level permissions, additional measures may be needed to prevent PII and other sensitive information from leaking into organization-wide context. Each candidate is compared against existing organizational fragments using embedding similarity, reference overlap, and source type. If a close match exists, the system proposes merging rather than creating a duplicate; this is how independent observations from different users naturally converge. Each org fragment tracks how many independent observations contributed to it, providing a signal of confidence and consensus. High-usage org fragments are flagged for human review before update.

For example, if three analysts independently observe that a column should never be summed directly because rows duplicate per

fee rule, the system consolidates these into a single endorsed fragment anchored to that column. This is desirable because a single authoritative fragment is easier to maintain, reduces noise during retrieval, and carries a stronger trust signal than three unverified personal notes.

4.3 Agentic Retrieval

We observe a shift in the field toward retrieval systems that leverage structure in addition to mere text content, improving both retrieval accuracy and token efficiency [3]. The structure of fragments lends itself well to such richer retrieval methods. Object references enable *progressive disclosure*: when an agent queries a specific table, all fragments anchored to that table and its columns are surfaced automatically without embedding search; as the agent narrows its focus, more specific fragments appear. Relatedness between database objects and other fragment properties enables *agentic search*: an agent can actively search for related context by reference, source type, or semantic similarity when it encounters unfamiliar objects. And properties like trust status, recency, and source type enable *deterministic ranking* that goes beyond similarity scores: for example, endorsed fragments outrank personal observations, and frequently accessed fragments are prioritized.

These retrieval methods are also a natural fit for standard tool-use interfaces such as MCP [1], ensuring compatibility with any tool-calling agent. Beyond that, we are exploring the effectiveness of an agent that pre-compiles a task-specific guide from relevant fragments. Such compiled guides could be reused for similar questions, acting as a warm task-specific context layer. By keeping lineage to the original fragments, it is straightforward to track when a guide needs to be updated because an underlying fragment has changed.

4.4 Evaluation and Data Modeling

As the context layer grows, human oversight at the fragment level becomes impractical: there are too many fragments, they evolve too quickly, and judging individual fragment quality in isolation is difficult. We suggest leaving curation largely to agents, and providing oversight and observability through organization-specific **evaluations**.

Public benchmarks like DABstep measure agent capability in general, but cannot tell a user whether *their* agent understands *their* data. We call these **self-serving evals**: question-answer pairs written against a user's actual database, testing whether the agent returns correct results given currently available context. Instead of curating individual fragments, the user maintains a set of questions that are evaluated for correctness as the context layer evolves, giving visibility into how good the context layer is. An agent could also suggest question-answer pairs based on usage patterns in the organization. Self-serving evals serve as both accuracy measurements and regression tests when context is promoted from personal to organizational scope.

Beyond evaluation, fragment usage statistics (which fragments are retrieved, how often, for which query patterns) provide another signal for informing data modeling decisions. Frequently used fragments that encode complex join logic are natural candidates for, e.g., materialization as views or macros. This closes the loop between context and the data layer.

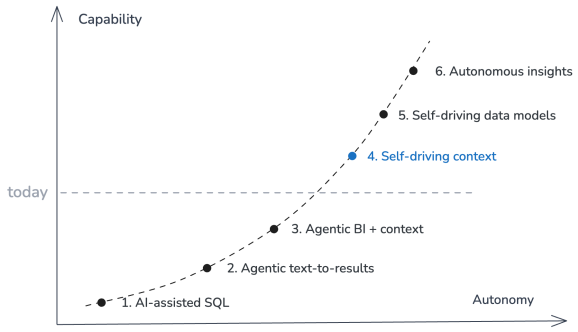

Figure 2: Stages of agentic analytics [20]: from AI-assisted SQL through agentic text-to-results and context-guided BI, toward self-driving context, self-driving data models, and autonomous insights. The dashed line marks roughly where the industry stands today. The context layer (§4) is our proposal for enabling stages 4 and 5.

5 Outlook: The Agentic Data Stack

LLMs are already proficient at writing SQL, and on our platform, agent-generated queries now exceed human-authored ad-hoc queries. When humans stop writing SQL, much of the traditional data stack loses its raison d'être: BI tools, transformation layers, and orchestration frameworks were built for human operators. AI blurs the swim-lanes between these categories, and anything that can be vibe-coded will be [19]. What remains is compute and *context*: the organization-specific map between raw data and the questions an agent can answer reliably. If context is the differentiator, the natural question is what the progression toward a fully autonomous data stack looks like.

Figure 2 outlines one such progression. We believe stages 1–3 are well-understood today. The context layer described in §4 is our proposal for enabling stage 4; stage 5 closes the loop by using accumulated context to inform data modeling automatically. While speculative, one plausible trajectory is that data engineers shift from writing SQL to curating context and writing evals, while the system progressively learns what questions matter and provides answers before a human asks. We further expand on this vision in [20].

6 Conclusion

Our benchmarks show that both domain context and targeted data modeling significantly improve agent accuracy, even with a low-cost model. Building on this finding, we propose a context layer that makes organizational knowledge a first-class, self-maintaining resource. Open problems that we are actively exploring include bridging personal and organizational memory, building retrieval that goes beyond naive search, and giving organizations evaluation tools to measure whether their agent actually understands their data. We believe that as the primary consumers of data systems shift from humans to agents, agentic maintenance of context and data models becomes increasingly important, and the human skill required will eventually shift from "can write SQL" to "knowing what the system should care about."

References

[1] Anthropic. 2024. Model Context Protocol. https://modelcontextprotocol.io/

[2] Anthropic. 2025. Claude Code Memory and Dream Consolidation. https://milvus.io/blog/claude-code-memory-memsearch.md Inspired by the Dream Consolidation Pattern.

[3] Ash Ashutosh and Edo Liberty. 2026. Pinecone Nexus: The Knowledge Engine for Agents. https://www.pinecone.io/blog/knowledge-infrastructure-for-agents/ Pinecone Blog.

[4] Prateek Chhikara et al. 2025. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. *arXiv preprint arXiv:2504.19413* (2025). https://arxiv.org/abs/2504.19413

[5] Cube Dev. 2026. Cube. https://cube.dev/

[6] dbt Labs. 2026. dbt Semantic Layer. https://www.getdbt.com/

[7] Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. 2025. DABstep: Data Agent Benchmark for Multi-step Reasoning. *arXiv preprint arXiv:2506.23719* (2025). https://arxiv.org/abs/2506.23719

[8] Evgeniia Egorova. 2025. *Automatic Metadata Extraction for Text-to-SQL.* Master's thesis. https://arxiv.org/abs/2505.19988 Master's thesis, in collaboration with MotherDuck.

[9] GitHub. 2025. Building an Agentic Memory System for GitHub Copilot. https://github.blog/ai-and-ml/github-copilot/building-an-agentic-memory-system-for-github-copilot/

[10] Google. 2026. Gemini 3 Flash. https://deepmind.google/technologies/gemini/

[11] Andreessen Horowitz. 2025. Your Data Agents Need Context. https://a16z.com/your-data-agents-need-context/

[12] Chris Latimer, Nicoló Boschi, Andrew Neeser, Chris Bartholomew, Gaurav Srivastava, Xuan Wang, and Naren Ramakrishnan. 2025. Hindsight is 20/20: Building Agent Memory that Retains, Recalls, and Reflects. *arXiv preprint arXiv:2512.12818* (2025). https://arxiv.org/abs/2512.12818

[13] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, Matei Zaharia, Alvin Cheung, Natacha Crooks, Joseph E. Gonzalez, and Aditya G. Parameswaran. 2026. Supporting Our AI Overlords: Redesigning Data Systems to be Agent-First. In *CIDR*. https://arxiv.org/abs/2509.00997

[14] Malloy. 2026. Malloy. https://www.malloydata.dev/

[15] Vasilije Marković, Lazar Obradović, Laszlo Hajdu, and Jovan Pavlović. 2025. Optimizing the Interface Between Knowledge Graphs and LLMs for Complex Reasoning. *arXiv preprint arXiv:2505.24478* (2025). https://arxiv.org/abs/2505.24478

[16] Jacob Matson. 2025. What If We Don't Need the Semantic Layer? *MotherDuck Blog* (2025). https://motherduck.com/blog/what-if-we-dont-need-the-semantic-layer/

[17] Open Semantic Interchange. 2025. OSI Core Metadata Specification. https://github.com/open-semantic-interchange/OSI Version 0.1.1.

[18] OpenAI. 2025. Inside Our In-House Data Agent. https://openai.com/index/inside-our-in-house-data-agent/

[19] Jordan Tigani. 2026. Future Casting the Modern Data Stack. *MotherDuck Blog* (2026). https://motherduck.com/blog/future-casting-the-modern-data-stack/

[20] Jordan Tigani. 2026. Water-Town: The Agent Swarm Data Stack. *MotherDuck Blog* (2026). https://motherduck.com/blog/water-town-agent-swarm-data-stack/