Towards Agentic Data Engineering: A Contract-Driven System with Self-Evolving Knowledge

Luan Prado[1], **Leonardo Guerreiro Azevedo**[1], **Adriano Veloso**[1,2]

[1]Kunumi Institute
30.130-138 – Belo Horizonte – MG – Brazil

[2]Computer Science Department, Federal University of Minas Gerais, Brazil

{luan.prado,leonardo.azevedo, adriano}@kunumi.com, adrianov@dcc.ufmg.br

Abstract. *Data engineering suffers from long, non-repeatable cycles due to the lack of persistent domain knowledge. This work proposes a contract-driven agentic architecture that externalizes knowledge into executable specifications across sessions. A Data Explorer Agent explores databases while a Data Transformation Agent translates validated contracts into pipelines. Contracts serve as specification, validation, and documentation, forming a feedback loop of recorded, generalized, reusable results. The evaluation of the architecture across diverse databases shows knowledge accumulation and reuse, detection of data quality issues, and the completeness of data pipelines, confirming that knowledge externalization is key to reliable data engineering automation.*

1. Introduction

Data engineering does not fail for lack of tools, but because the knowledge required to make data meaningful is tacit, fragmented, and constantly evolving. As a result, data exploration and engineering tasks fall into long, non-repeatable cycles in which assumptions are rediscovered and inconsistently encoded.

Data engineer task automation has evolved through several paradigms, from expert systems [Feigenbaum et al. 1969, Shortliffe 1976] and ETL frameworks [Inmon 1992, Kimball 1996], to machine learning [Sculley et al. 2015], data pipeline management [Munappy et al. 2020], and LLM-based interfaces [Liu et al. 2024]. Despite decades of advances, automation remains an open challenge [Munappy et al. 2020]. Existing approaches either assume stable schemas, operate statelessly, or fail to accumulate knowledge across interactions.

This work addresses those limitations by framing data engineering as a knowledge problem. We propose an agentic architecture that comprises two agents: the Data Explorer Agent (`DXA`), which performs database exploration, builds a knowledge base, and creates a roadmap composed of a list of contracts that describe database issues to be resolved; and the Data Transformation Agent (`DTA`), which translates contracts into data pipelines, and executes them to transform the database. The contracts separate semantic exploration from deterministic pipeline execution through persistent, inspectable artifacts, allowing human intervention where tacit knowledge can be externalized. This makes knowledge accumulation explicit, auditable, and reusable.

The novelty we present is: (i) separating knowledge construction from execution; (ii) using contracts as the boundary between both stages; and, (iii) treating the KB as the substrate for continuity and self-improvement. The approach addresses exploration, metadata, data quality, transformation, and pipeline automation over relational databases.

We evaluate the architecture on three databases of increasing complexity: NBA [Walsh 2026], MIMIC-IV [Johnson et al. 2024], and TOTVS Datasul ERP (commercial). Results show incremental knowledge accumulation, work continuity, stable operational costs, and automated generation of fully tested data pipelines.

The remainder of this work is organized as follows. Section 2 presents the background and Section 3 the related work. Section 4 details the architecture, and Section 5 presents the evaluation. Section 6 presents the conclusion and proposals of future work.

2. Theoretical Background

This section establishes the conceptual foundations of our work by synthesizing principles from information science and organizational learning. We first examine the DIKW hierarchy as a model for data abstraction (Section 2.1) and then discuss the SECI model and experiential learning as frameworks for the externalization and evolution of technical knowledge (Section 2.2).

2.1. From Data to Wisdom: The DIKW Hierarchy

The DIKW hierarchy formalizes the progression from raw data to actionable decision-making by organizing data, information, knowledge, and wisdom as successive levels of abstraction [Ackoff 1989, Rowley 2007]. Data corresponds to raw observations; information emerges when data is organized into meaningful relationships; knowledge captures validated patterns and rules; and wisdom concerns prescriptive judgment, *i.e.*, not only knowing what is true, but what should be done.

Although DIKW is widely adopted as a conceptual model, it has rarely been operationalized in autonomous software systems. Most data engineering tools concentrate at the lower layers, *e.g.*, query engines retrieve raw results, and schema catalogs organize metadata into structured descriptions [Kleppmann 2017]. Progression toward knowledge and wisdom still depends on human intervention, as engineers accumulate rules, anomalies, and business judgments through repeated exploration cycles, often without converting them into machine-readable artifacts [Munappy et al. 2020, Sculley et al. 2015].

Our work operationalizes this progression through a chain of versioned artifacts. The knowledge base produced by the architecture explicitly materializes metadata as information, validated query patterns and failure modes as knowledge, and a roadmap as a prescriptive transformation plan, *i.e.*, wisdom. Rather than treating DIKW as a descriptive hierarchy, we turn it into an executable, auditable process that links raw database observations to engineering decisions.

2.2. Knowledge Creation, SECI, and the Human in the Loop

A central challenge in data engineering is externalizing tacit knowledge, *i.e.*, converting what domain experts know but cannot readily articulate into explicit, reusable artifacts.

Nonaka and Takeuchi formalize this process in the SECI model, which describes knowledge creation as a spiral of four conversion modes: socialization, externalization, combination, and internalization [Nonaka and Takeuchi 1995].

This perspective is particularly relevant to data engineering because many crucial decisions depend on knowledge that is neither documented nor directly observable in the schema, like column semantics, historical anomalies, threshold meanings, and encoding conventions. In practice, such knowledge remains distributed across people and interactions, rarely reaching a form that pipelines or downstream tools can consume [Munappy et al. 2020, Sculley et al. 2015]. For instance, existing data catalog solutions support mainly the combination mode, since they organize preexisting explicit metadata, but they do not elicit, formalize, or accumulate tacit knowledge [Munappy et al. 2020].

Our architecture is designed around all four SECI modes. Socialization occurs during interactive exploration, when the agent surfaces ambiguities and asks targeted follow-up questions to the data engineer. Externalization is the key operational step. The `DXA` converts answers, decisions, and implicit judgments into persistent Markdown artifacts. Combination occurs when this externalized domain knowledge is synthesized with technical findings to generate a roadmap. Internalization happens bidirectionally: the engineer learns through the interaction, while `DTA` internalizes the roadmap into executable transformations. Each session in which the agent re-reads and updates its knowledge base corresponds to a new turn of the SECI spiral, allowing accumulated knowledge to deepen over time rather than being reconstructed from scratch.

This design also aligns with experiential learning [Dewey 1938, Kolb 1984]. The agents do not reason in isolation from evidence; they execute queries, observe results, register validated findings, and revise their understanding. In this sense, learning is grounded in iterative interaction with the data rather than one-shot inference.

3. Related Work

This section positions our work along the technical landscape of agentic systems for data engineering tasks, against which our contributions are contrasted.

3.1. Self-Improving Agents and Knowledge-Level Self-Modification

Recent AI research has increasingly focused on agents capable of improving themselves through experience. The Gödel Machine introduced the idea of a self-improving agent that rewrites itself only when a formal proof guarantees benefit, but this requirement made it impractical in real-world settings. Zhang *et al.* address this limitation with the Darwin Gödel Machine, which iteratively modifies its own code and maintains an archive of successful variants, enabling open-ended self-improvement through accumulated stepping stones [Zhang et al. 2025]. Hyperagents extend this line of work by combining task execution and self-modification within a unified editable program, allowing improvement to transfer across domains beyond coding [Zhang et al. 2026].

These works represent the state of the art in self-improving agents, but their approach operates at the code level: the agent rewrites its own implementation. This is effective in domains where the object of work and the substrate of evaluation coincide,

such as software generation. In data engineering, however, the main difficulty is not algorithmic but semantic and organizational. What must improve is not only the code the system writes, but the knowledge it accumulates about the data and the business context.

Our work pursues a complementary form of self-modification: knowledge-level self-improvement. Rather than rewriting code, the agents rewrite their own knowledge base, which serves as persistent memory. When `DXA` discovers that a previously validated recipe fails on new data, it updates the relevant error and recipe entries; when a business rule is corrected by the user, it overwrites the corresponding knowledge artifact. This is not code changing code, but knowledge changing future behavior. The improvement is mediated by the SECI cycle: the correction is socialized with the user, externalized into the artifact, and recombined into downstream execution. The result is a self-improving system grounded in organizational knowledge rather than benchmark optimization, which is more appropriate for open-ended data engineering tasks.

3.2. LLM Agents for Data Tasks: Reasoning, Acting, and Their Limits

Yao *et al.* introduced ReAct as a paradigm in which language models interleave reasoning and action, enabling grounded trajectories that update plans as they interact with external environments [Yao et al. 2023, Bezerra 2025]. This coupling of reasoning and acting reduces hallucination and improves interpretability. Our `DXA` follows this structural paradigm by alternating between hypothesis formation and SQL execution. However, ReAct remains fundamentally session-scoped: it does not produce persistent artifacts that survive beyond the current interaction. In our architecture, this limitation is addressed by the knowledge base, which functions as durable working memory across sessions.

Multi-agent frameworks for data tasks have also advanced rapidly. MAC-SQL uses collaborative agents to decompose and refine SQL generation [Wang et al. 2025], while D-Bot applies LLMs to database diagnosis through configuration analysis [Zhou et al. 2024]. These systems achieve strong results on curated benchmarks, but they assume a known and well-documented schema. They do not autonomously discover schema structure, elicit undocumented business rules, or accumulate persistent knowledge about ambiguous semantics and embedded encodings. More importantly, their output is typically limited to SQL queries or diagnosis results.

Other data-oriented agents target scientific and analytical workflows, such as feature engineering and model training [Wang et al. 2025, Hong et al. 2025], but they usually operate over fixed pipelines or pre-specified task descriptions. None of these approaches addresses the full, open-ended loop considered in this work: exploring an unknown database, externalizing organizational knowledge during that exploration, and transforming the resulting knowledge into a prescriptive engineering roadmap executable on orchestration platforms.

This is the gap our work addresses. The contribution is an architecture that couples exploration, knowledge accumulation, contract generation, and pipeline execution in a persistent loop.

4. The Agentic Architecture

The agentic architecture comprises (Figure 1): the `Data Explorer Agent` (`DXA`), which explores a business database, externalizes knowledge, and creates a roadmap; and,

the `Data Transformation Agent` (`DTA`) uses the roadmap to create an improved business database.

`DXA` and `DTA` are implemented as agents (LLM that uses tools) [Yao et al. 2023] with persistent, file-based memory. The user interacts with both agents in natural language via a prompt-based User Interface. The interaction with `DXA` primarily serves to answer business questions while exploring the database. The `DXA` may proactively reach out to the user whenever it encounters fields, business rules, or semantic ambiguities that require domain knowledge to proceed — such as clarifying the meaning of a column or confirming how a specific rule should be applied. Newly validated findings are written back into the knowledge base (KB). Once the exploration is complete, `DXA` requests the user's approval before generating a roadmap, which is created only upon explicit user consent. The roadmap items act as data contracts for `DXA` and `DTA` communication. The roadmap is consumed by the `DTA` to create data pipelines. `DTA` employs data tools to transform the business database using the data pipelines. The user interaction is optional during `DTA` execution.

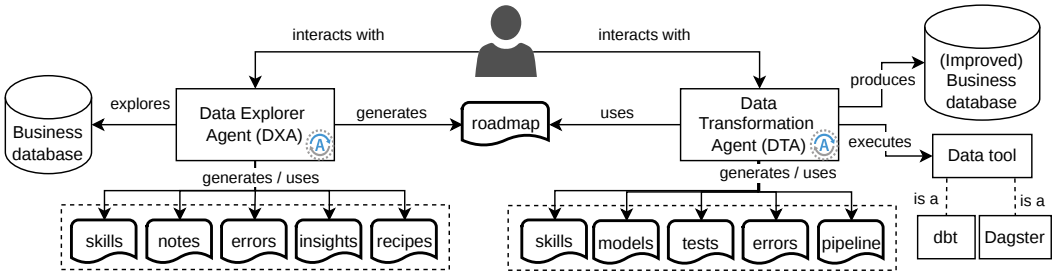

Figure 1. The Data Engineer Agentic Architecture.

An agent's execution episode over a target artifact corresponds to a session. The `DXA` session corresponds to a run until the roadmap is produced or updated. It stops when no new validated findings are needed for the roadmap. Then, the `DXA` asks if the user wants to create the roadmap. The user may also ask explicitly to generate the roadmap during the exploration, finishing the `DXA` session. The `DTA` session runs until the roadmap contracts are implemented in the transformed database. `DXA` and `DTA` also stop when execution is interrupted.

Due to space limitations, we provide metadata schema, examples of the consumed and generated files, examples of user interaction with the architecture UI, and traces of the agents' execution in GitHub [Prado et al. 2026].

4.1. Data Explorer Agent (DXA)

The `DXA` assists analysts, domain experts, and data engineers during database exploration and requirement elicitation. The current implementation targets relational databases and relies on SQL queries plus standard DBMS metadata facilities.

During database exploration, `DXA` builds a persistent knowledge base (KB) composed of four Markdown files:

- `NOTES.md`: schema metadata, descriptive statistics, and business rules.
- `ERRORS.md`: syntactic, semantic, and logical issues.

- `INSIGHTS.md`: business-relevant findings.
- `RECIPES.md`: query templates validated through execution.
- `SKILLS.md`: instructions to perform causal analysis, to plot charts, graphs, or other kinds of visualization, to execute data profiling.

Each `DXA` session begins by either creating the KB or loading it if it has already been created. Later sessions begin with accumulated organizational memory rather than rediscovering the same facts. `DXA` performs schema discovery, inspects the catalog, maps keys and types, estimates table sizes, and executes exploratory queries to identify nulls, encoded values, ambiguities, and candidate business rules. All queries and representative results are exposed to the user, making the exploration process transparent and auditable. As the KB grows, exploration becomes cumulative. `DXA` reuses previous recipes, checks prior assumptions, and asks targeted questions only where tacit knowledge is still missing.

Once exploration is complete, `DXA` produces a roadmap. Each roadmap item corresponds to a contract that specifies a data engineering issue validated by the `DXA` and to be resolved by the `DTA`. A contract is created only after a query that evidences the issue has been executed and validated. The roadmap, therefore, serves both as an executive summary for the user and as a machine-readable contract list for execution. Each roadmap contract contains:

- short issue description;
- artifact where the issue was found (*e.g.*, a table);
- issue characterization (*e.g.*, nulls, inconsistent types, denormalizations, ambiguous encodings);
- severity level of the issue (*critical*, *warning*, or *info*);
- query that evidences the issue and summary of its execution results (*e.g.*, number of query columns and returned rows, mixed data types in a column); and
- recommended action to solve the issue (*e.g.*, cast, impute, normalize, deduplicate).

4.2. Data Transformation Agent (`DTA`)

The `DTA` receives the roadmap and implements each roadmap contract as a data pipeline, *i.e.*, the `DTA` translates the roadmap contracts into models, tests, and orchestration assets to transform the business database. In the current implementation, the `DTA` produces `dbt`[1] models for transformation and uses `Dagster`[2] for orchestration.

`dbt` expresses transformations declaratively as SQL `SELECT` statements, while handling testing, documentation, lineage, and versioning. `Dagster` complements this by orchestrating asset execution, inferring dependencies, attaching observability metadata, and managing scheduling and recovery. Together, they enable `DTA` to translate validated specifications into executable, monitorable pipelines.

The `DTA` maintains its own KB, composed of:

- `ROADMAP.md`: the contracts received from `DXA`;
- `MODELS.md`: the created `dbt` models;
- `TESTS.md`: tests and results;
- `PIPELINE.md`: assets, dependencies, and schedules;

[1] `https://docs.getdbt.com/`
[2] `https://dagster.io/`

- `ERRORS.md`: transformation-time failures and workarounds;
- `SKILLS.md`: command references for `dbt` and `Dagster`.

On the first session, `DTA` initializes the `dbt` project, configures the target database, and processes the roadmap contracts sequentially. Then, `DTA` follows an execution loop. For each contract, it reads the specification, writes the corresponding model and tests, executes them, and either registers success or records the failure and its workaround. Once the full pipeline succeeds, `DTA` generates `Dagster` assets, runs the workflow, updates `PIPELINE.md`, and commits changes to `Git`[3], for version control. Failures are also recorded, generalized, and made available for reuse in later contracts and future sessions.

4.3. Architectural Rationale

The two-agent decomposition follows from three converging principles.

The first is *epistemic*, grounded in the SECI model [Nonaka and Takeuchi 1995]. Socialization and externalization operate over tacit, uncertain, context-dependent knowledge. Combination and internalization operate over explicit, structured, executable knowledge. `DXA` operates in externalization, while `DTA` operates in internalization. The roadmap marks the point at which tacit knowledge becomes explicit to be executed.

The second is *structural*, grounded in Dijkstra's separation of concerns [Dijkstra 1982]. `DXA` is responsible for knowing through schema discovery, business rule elicitation, evidence validation, and memory construction. `DTA` is responsible for doing transformation logic, testing, orchestration, and version-controlled execution. Mixing these concerns would weaken modularity, auditability, and failure diagnosis.

The third is *organizational*, grounded in Conway's Law [Conway 1968]. In practice, exploratory analysis and requirement definition are usually distinct from pipeline implementation and orchestration. Our architecture mirrors this division: `DXA` serves as an exploratory analyst, while `DTA` serves as an execution engineer. This allows human intervention at the most meaningful point, which is the roadmap, where requirements are explicit but execution has not yet begun.

5. Evaluation

We evaluate the proposed architecture from four complementary perspectives: cost, knowledge accumulation, continuity across sessions, and end-to-end execution capability. The current evaluation does not claim superiority over existing systems and does not provide a gold-standard comparison. Our goal is to assess whether the architecture accumulates reusable knowledge, reduces rework, improves through its own artifacts, and produces pipelines that improve databases of increasing complexity.

5.1. Experimental Setup and Evaluation Dimensions

We evaluate the architecture across three databases that span a spectrum of real-world complexity: schema size, domain opacity, and availability of external documentation.

NBA [Walsh 2026] is an open sports database with 16 tables covering games, players, teams, and draft history from 1946 to 2023 (easy complexity). Its schema is

[3]`https://git-scm.com/`

well-structured and extensively documented online, making it a controlled setting for evaluating baseline behavior. Even so, `DXA` sessions uncovered non-trivial quality issues.

MIMIC-IV Demo [Johnson et al. 2024] is a de-identified clinical database with 31 tables (medium complexity). Its complexity is primarily semantic rather than structural.

TOVS Datasul is a commercial ERP from a Brazilian import company with 48 tables, pervasive type inconsistencies, undocumented business rules (hard complexity), and data quality problems whose semantics cannot be recovered without domain expertise. It uses the Progress OpenEdge system, whose legacy schema conventions (*e.g.*, hyphenated column names) constitute structural outliers for LLM-based data tooling.

All agent sessions were conducted with the Claude Sonnet 4.6 [Anthropic 2026] model, treated as a fixed experimental condition with no prior knowledge of any target database. Model selection was guided by a benchmark for evaluating whether language models detect and reject broken premises rather than confidently elaborating on them [Gostev 2026][4]. This property is structurally important in our setting because if the model uncritically accepts undocumented business assumptions, it could corrupt the knowledge base rather than improve it.

We evaluate the architecture along four dimensions aligned with its design goals:

1. **Knowledge accumulation**: growth and stability of the KB across sessions.
2. **Work continuity**: ability to resume from and reuse prior KB without reprocessing validated information.
3. **Self-evolving behavior**: traceable instances of the agent detecting, correcting, and generalizing from its own errors.
4. **Pipeline completeness**: `DTA`'s ability to the pipeline from the roadmap.

5.2. Cost, Knowledge Accumulation, and Continuity

Table 1 summarizes session-level metrics, which consider the concepts of session, interaction, and turn. A session corresponds to a complete execution of the agent as defined in Section 4. Interactions are the human communications with the agent. A turn refers to one complete round-trip within the agentic loop. It encompasses three stages: (i) Thought (Planning): The agent analyzes the current objective and decides what needs to be done. (ii) Action (Tool Use): The agent performs an action, such as calling an API, performing a web search, or running Python code. (iii) Observation (Result): The agent reads the result returned by the tool.

Thirteen sessions were conducted: five `DXA` and one `DTA` sessions on NBA, two `DXA` and one `DTA` on MIMIC-IV, and one `DXA` and three `DTA` on the ERP, yielding 608 turns across 35 interactions. The low-cost values and low-cost increases suggest that contract-driven design bounds reasoning cost even as schema complexity increases.

All KB files are initialized as empty four-line stubs. Table 2 reports the final `DXA` KB state for all three databases, and we describe some examples of entries of the KB as follows. We do not present `DTA` KB state due to lack of space.

[4]Claude Sonnet 4.6 has a high performance on the benchmarks, demonstrating that the model is capable of realizing misleading questions or information, and stating to the user the problems instead of agreeing with false premises.

Table 1. Session metrics by database and agent.

Database	Agent	Sessions	Interactions	Turns	Cost (USD)	Cost per turn
NBA	DXA	5	12	165	$4.76	$0.029
NBA	DTA	1	1	51	$0.99	$0.019
MIMIC-IV	DXA	2	13	170	$7.54	$0.044
MIMIC-IV	DTA	1	2	94	$2.89	$0.031
ERP Datasul	DXA	1	2	45	$0.97	$0.022
ERP Datasul	DTA	3	7	128	$5.20	$0.041
Total	—	**13**	**37**	**653**	**$22.35**	—

Table 2. DXA Knowledge Base growth from empty initial state.

File	NBA		MIMIC-IV		ERP Datasul	
	Lines	Entries	Lines	Entries	Lines	Entries
NOTES.md	217	7 sections	212	5 sections	177	8 sections
RECIPES.md	119	6 recipes	119	6 recipes	135	9 recipes
ERRORS.md	21	2 entries	—	—	39	5 entries
INSIGHTS.md	—	—	—	—	—	—

In the NBA, NOTES.md documents all 16 tables. We highlight: the discovery that team_info_common is entirely empty (0 rows, 26 columns); that season_type conflates "All-Star" and "All Star" as distinct enum values, silently splitting the same game type across two keys; and that quarter-score columns in line_score mix VARCHAR and BIGINT types, making direct aggregation incorrect. These findings were also captured in ERRORS.md with two entries documenting root causes and corrective patterns, and later consumed by the DTA to guide deduplication logic.

In the MIMIC-IV, NOTES.md accumulates schema structure across all 31 tables, nullability profiles, and domain-specific operational caveats. Some examples of entries are: datetime columns are stored as VARCHAR; ICD-9 and ICD-10 diagnostic codes coexist in the same tables without a type discriminator; and, calendar years are shifted 80–100 years forward for de-identification, making temporal reasoning unreliable without domain awareness. Standard schema inspection cannot surface these constraints. They require a clinical context to interpret correctly, which makes this database a stress test for the KB's ability to accumulate domain-specific caveats.

In the ERP setting, NOTES.md records structural findings uncovered during exploration, such as: invoice_number alone is not a valid primary key, the correct one is (invoice_number, series); and, the product catalog mixes company products with canteen items, internal documents, and IT equipment SKUs (Stock Keeping Unit)[5]. Neither finding is recoverable from schema structure alone; both required iterative query execution and business-rule elicitation.

The RECIPES.md files also accumulate validated SQL templates annotated with domain-specific constraints. In the NBA, it includes a recipe for filtering season_type using a pattern guard to handle the All-Star/All Star inconsistency. In the MIMIC-

[5]It is a unique alphanumeric code assigned to each product or variation (size, color, model) to track inventory, prices, and sales.

IV, it includes join constraints such as the need to include `icd_version` when joining diagnosis tables. In the ERP case, it includes a series of normalization rules. These recipes later seed `DTA` contracts and `dbt` implementations, making the transition from exploration to execution explicit.

To realize the value of prior knowledge, *i.e.*, from cold exploration to reuse without re-exploration, we conducted tests in sessions. In the NBA processing, in the first session, there is no prior knowledge, only empty files. As the agent explores the data, the files are populated, and knowledge is acquired. We realize continuity in the third session of the NBA processing. With the KB populated, the agent completes 20 turns of analysis and visualization without adding any data to the KB. This indicates that the agent persists the correct data, eliminating the need to spend additional tokens[6] on exploration. All queries are retrieved or adapted from `RECIPES.md`, and all execute correctly on the first attempt. The Session 3 cost of $0.57 ($0.029/turn) indicates that KB reuse does not introduce overhead proportional to KB size.

5.3. Self-Evolving Behavior and Pipeline Completeness

Self-evolving behavior appears as traceable KB updates in which the agent corrects, generalizes, or supersedes previously recorded knowledge. We highlight the following cases.

First, a type-related failure observed during exploration was not treated as an isolated error. The root cause and workaround were documented, and the lesson was generalized as a structural observation in the knowledge base.

Second, the agent detected a silent semantic failure. Applying `MIN/MAX` to a date column stored as text produced lexicographically ordered results without raising an exception. The agent documented both the incorrect and corrected approaches, generalized the lesson into the KB, and reused the corrected pattern in subsequent queries.

Third, a parsing workaround identified during exploration was elevated into a reusable `dbt` macro during pipeline implementation. This provides a concrete example of knowledge moving from exploratory error handling to a reusable execution artifact.

A fourth case emerges from the NBA `DTA` session. The `ERRORS.md` entry for `season_type` normalization (recorded during `DXA`) had documented the `"All-Star"/"All Star"` inconsistency as a filtering hazard. When `DTA` implemented the corresponding staging model for `game`, it discovered that normalizing `season_type` upstream caused 56 apparent `game_id` duplicates. These records were ingested twice with different enum values. The agent traced the root cause to the same KB entry, resolved it with a `QUALIFY` deduplication clause, and confirmed that 5/5 tests passed. This case illustrates how a `DXA`-recorded error can propagate into a `DTA` structural decision, closing the loop between exploration and execution.

Pipeline completeness is summarized in Table 3, which presents the following metrics. *Roadmap items* correspond to the roadmap contracts generated by `DXA` and consumed by the `DTA`. *Roadmap items implemented* are the contracts for which `DTA` generates corresponding code to transform the business database. `DTA` generates `dbt` *models* and *tests*. A `dbt` model is a text file that contains a 'SELECT' statement used to

[6]A token is the basic unit of text that AI language models use to read and generate language, *e.g.*, in English, 1 token corresponds to approximately 4 characters or $\frac{3}{4}$ of a word.

state how database artifacts, like a table, should be created. A `dbt` test is an automated validation that ensures the data generated by `dbt` models follows the expected rules, *e.g.*, a test to verify that a column is unique and never empty. Tests may pass or fail during the database transformation. The *staging, intermediate, and mart*[7] models correspond to specific categories or roles of the `dbt` models. *Reusable macros* act as programming functions that enable dynamic SQL code generation and logic encapsulation, preventing code duplication across different layers of the data lineage.

Table 3. `DTA` pipeline output summary.

Metric	NBA	MIMIC-IV	ERP
Roadmap items (total)	13	18	18
Roadmap items implemented	3*	11	18
dbt models produced	6	20	14
dbt tests passing	5	94	96
dbt tests failing	0	0	0
Staging models	5	20	8
Intermediate models	1	0	3
Mart models	0	0	3
Reusable macros	0	0	3

* Session terminated mid-execution; 10 remaining items not yet started.

The NBA `DTA` session represents a partial execution of 13 roadmap items identified during `DXA` exploration, 3 were fully implemented (covering the empty-table guard, `season_type` normalization, and `line_score` type casting), with a fourth under active development when the session concluded. Hence, the 6 models produced and 5 passing tests constitute a structurally valid but incomplete pipeline, offering a concrete instance of incremental execution that can be resumed from the current KB state without reprocessing prior work.

For MIMIC-IV, `DTA` produced 20 staging models for the 11 roadmap items implemented. They cover all 31 tables, with typed columns, derived boolean flags (*e.g.*, `is_inpatient`, `came_from_ed`), and automated tests. From the total, seven roadmap items were not implemented. These items were left open because they require external ontologies or domain-specific reference tables. They were documented explicitly rather than silently omitted.

For the ERP, all 18 roadmap items were implemented across staging, intermediate, and mart layers. Three reusable macros unlocked 8 additional tables corresponding to 2,591,866 rows, *i.e.*, the macros allowed us to easily generate 8 additional tables in the pipeline without writing code from scratch for each one. The resulting mart layer exposes three analytical outputs: fiscal notes (164,429 rows, 18 quality flags), product catalog (5,439 rows), and sales target attainment (2,645 rows per representative).

These results make the DIKW progression observable as an artifact chain rather

[7]The transformation pipeline is structured into three conceptual layers: Staging (ingestion, cleaning, and standardization of raw data in a one-to-one relationship with the source), Intermediate (an isolated transition layer for data joins and the application of complex business logic), and Mart (the final presentation layer, structured dimensionally or denormalized, and optimized for analytical consumption).

than a purely conceptual framing. Raw query results become information in `NOTES.md`, reusable patterns in `RECIPES.md` and `ERRORS.md`, and finally a severity-ranked transformation plan in `ROADMAP.md`. Each level is stored as a versioned artifact that can be inspected, corrected, and reused by the engineer.

5.4. Discussion

Three main findings are supported by the results.

Cost remains stable across complexity levels. This suggests that contract-driven design localizes reasoning to a single contract at a time, preventing global schema size from dominating token consumption.

Self-evolving behavior is empirically distinguishable from stateless LLM use. In all documented cases, the same pattern appears: an execution failure is detected, analyzed, recorded in the KB, generalized into a reusable rule, and later reused. The KB is a storage through which self-improvement becomes persistent and inspectable.

The human remains in the loop by design rather than by limitation. Domain judgments, ontology decisions, and unresolved business rules are explicit open items, not resolved autonomously – socialization requires a human interlocutor (SECI model).

6. Conclusion

This work proposed a self-evolving multi-agent architecture that operationalizes the DIKW hierarchy and the SECI knowledge spiral as concrete computational steps for data engineering. The architecture comprises two agents: Data Explorer Agent (`DXA`) and Data Transformation Agent (`DTA`). The `DXA` externalizes tacit organizational knowledge into versioned artifacts, while the `DTA` transforms that accumulated knowledge into data pipelines that run over a database to improve it.

The proposal was evaluated across three databases of increasing complexity, demonstrating: (i) stable per-turn cost; (ii) knowledge accumulation through KB growth and content stabilization; (iii) ability to resume from and reuse prior knowledge without reprocessing validated information; (iv) traceable self-evolving behavior through recorded error correction and reuse; (v) pipeline completeness through roadmap items implemented, `dbt` models produced, and `dbt` tests passing. The human data engineer remains in the loop by design. The domain judgments are externalized, validated, and preserved in the KB. The results suggest that reliable automation in data engineering depends less on stateless reasoning than on the externalization of persistent knowledge.

A current limitation is the lack of a ground truth, which prevents the use of precision and recall metrics in the evaluation. Future work will address this through post-hoc expert annotation of a held-out database, enabling partial recall estimation against a curated gold standard. Other future work include extension to semi-structured sources, and an integrated AI layer that allows running predictive models directly via SQL commands.

Declaration on Generative AI: Claude Sonnet 4.6 [Anthropic 2026] model was used in the experiments. Claude and Grammarly were used to help structure the text, polish writing, improve conciseness, and check grammar.

Acknowledgement: This work was supported by Kunumi Institute. The authors thank the institution for its financial support and commitment to advancing scientific research.

References

Ackoff, R. L. (1989). From data to wisdom. *Journal of Applied Systems Analysis*, 16:3–9.

Anthropic (2026). Claude sonnet 4.6. Available at: https://www.anthropic.com/news/claude-sonnet-4-6. URL date: June 15th, 2026.

Bezerra, E. (2025). Introduction to llm-based agents. In *Topics in Data and Information Management: SBBD 2025 Short-Courses*, Topics in Data and Information Management, pages 1 –30. SBC, Porto Alegre, RS.

Conway, M. E. (1968). How do committees invent? *Datamation*, 14(4):28–31. Available at: https://www.melconway.com/Home/pdf/committees.pdf.

Dewey, J. (1938). *Experience and Education*. Kappa Delta Pi / Collier Books, New York.

Dijkstra, E. W. (1982). On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66. Springer-Verlag, New York. Original manuscript EWD447, 1974.

Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J. (1969). On generality and problem solving: A case study using the DENDRAL program. *Machine Intelligence*, 6:165–190.

Gostev, P. (2026). Bullshitbench: A benchmark for detecting nonsense in ai model responses. Available at: https://github.com/petergpt/bullshit-benchmark. URL date: June 15th, 2026.

Hong, S., Lin, Y., Liu, B., Liu, B., Wu, B., Zhang, C., Li, D., Chen, J., Zhang, J., Wang, J., et al. (2025). Data interpreter: An llm agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19796–19821.

Inmon, W. H. (1992). *Building the Data Warehouse*. QED Technical Publishing Group / John Wiley & Sons, New York.

Johnson, A., Bulgarelli, L., Pollard, T., Gow, B., Moody, B., Horng, S., Celi, L. A., and Mark, R. (2024). MIMIC-IV. *PhysioNet*. Version 3.1. Available at: https://doi.org/10.13026/kpb9-mt58. URL date: June 15th, 2026.

Kimball, R. (1996). *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, New York, 1st edition.

Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, Sebastopol, CA.

Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development*. Prentice Hall, Englewood Cliffs, NJ.

Liu, X., Shen, S., Li, B., Ma, P., Jiang, R., Luo, Y., Zhang, Y., Fan, J., Li, G., and Tang, N. (2024). A survey of NL2SQL with large language models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*.

Munappy, A. R., Bosch, J., and Holmström Olsson, H. (2020). Data pipeline management in practice: Challenges and opportunities. In *Product-Focused Software Process Improvement (PROFES 2020)*, volume 12562 of *Lecture Notes in Computer Science*, pages 168–184, Cham. Springer.

Nonaka, I. and Takeuchi, H. (1995). *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, New York.

Prado, L., Azevedo, L. G., and Veloso, A. (2026). Agentic Data Engineering. Available at: https://github.com/kunumi/agentic-data-engineering. URL date: June 15th, 2026.

Rowley, J. (2007). The wisdom hierarchy: Representations of the DIKW hierarchy. *Journal of Information Science*, 33(2):163–180.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems (NIPS 2015)*, volume 28, pages 2503–2511. Curran Associates.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*. Artificial Intelligence Series. Elsevier, New York.

Walsh, W. (2026). NBA database. Version 3.1. Available at: https://www.kaggle.com/datasets/wyattowalsh/basketball. URL date: June 15th, 2026.

Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Zhang, Q.-W., Yin, D., Sun, X., and Li, Z. (2025). MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*, pages 540–557, Abu Dhabi, UAE. Association for Computational Linguistics.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhang, J., Hu, S., Lu, C., Lange, R., and Clune, J. (2025). Darwin Gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*.

Zhang, J., Zhao, B., Yang, W., Foerster, J., Clune, J., Jiang, M., Devlin, S., and Shavrina, T. (2026). Hyperagents. *arXiv preprint arXiv:2603.19461*.

Zhou, X., Li, G., Sun, Z., Liu, Z., Chen, W., Wu, J., Liu, J., Feng, R., and Zeng, G. (2024). D-Bot: Database diagnosis system using large language models. *Proceedings of the VLDB Endowment*, 17(10):2514–2527.