Architecting the AI-Powered Agentic Data Cloud

Yeounoh Chung, Thibaud Hottelier, Cosmin Arad, Brenton Milne, Per Jacobsson,
Sam Idicula, Fatma Özcan, Alon Halevy, Yannis Papakonstantinou
Google Cloud, USA
{yeounoh, tbh, carad, bmil, pjacobsson, samidicula, fozcan, halevy, yannispap}@google.com

Abstract

This paper describes Google Cloud's paradigm shift from traditional, siloed data management services into a unified, AI-powered Agentic Data Cloud driven by large language models (LLMs) integration. Rather than treating AI as an auxiliary service that sits on top of isolated database silos, we are embedding LLM intelligence directly into the core data cloud services. To realize this vision, we detail several foundational components in our Agentic Data Cloud designed to support production-scale autonomous applications and agents, including AI functions, natural language interfaces to data, and a modern search stack. We further claim that the reliability of these AI-integrated solutions is largely dependent on a rigorous semantic foundation, and describe our universal metadata catalog. Bringing these analytics and data processing services with rich semantic foundation creates a scalable AI Agentic Data Cloud ecosystem, enabling intelligent and autonomous agents and applications.

1 Introduction

Google Cloud has been building the AI-powered Agentic Data Cloud, transitioning traditional data cloud products from isolated storage and database silos into unified, intelligent engines. For decades, the data management landscape was defined by specialized, decoupled services (e.g., analytical data warehouses, operational databases, ETL tools, standalone data science or ML pipelines) bundled under unified billing. In this legacy paradigm, data science often lived in a separate workbench, requiring tedious orchestration to move data from storage into specialized environments for modeling and inference. Also, applications were issuing manually-crafted, deterministic, logic-oriented SQL queries to databases. The primary catalyst for a fundamental paradigm shift in this landscape is the emergence of GenAI foundation models, with Large Language Models (LLMs) and embedding generator models being very important to the paradigm shift [1–3] Foundation models can make semantic sense of the data and interpret complex intents. Furthermore, the reasoning and multi-step planning capabilities of LLMs enable them to function as autonomous controllers that can select tools, and execute workflows [4, 5]. This drives both business intelligence and database applications to a transformation towards the "agent-first" paradigm [6] and, in turn, the AI-native database. In business intelligence the primary consumers of data are no longer limited to human analysts performing retrospective discovery and actions. In this new landscape, the primary consumers are autonomous applications and agents that leverage data for analytical insights. This involves complex analytical tasks, such as strategic forecasting. Similarly, operational database applications provide a natural language interface that allows agents to interact with the database, decipher user intent, and execute the resulting queries.

Rather than treating AI as an auxiliary service, and agents accessing underlying storage services, we are integrating LLMs directly into our data analytics, operational engines, and data science workflows [7–9]. By embedding these capabilities into the data plane, we provide the scalability and performance required for production-scale AI agents and applications. At the same time, raw scalability and direct

model integration does not straightforwardly translate into reliable agents and autonomous applications. For these systems to transition from experimental demos to reliable enterprise agents executing actions independently, we must shift our focus from mere connectivity to the quality and verifiability of their actions and outputs. To bridge this gap, we claim that the success of AI-powered data cloud is strongly predicated on a rigorous metadata foundation [10–13]. While LLMs provide the reasoning catalyst, they cannot compensate for poor or missing metadata – a misconception that training a better model would magically compensate for this. Without a unified semantic model to ground non-deterministic reasoning in deterministic data, AI-integrated products fail to provide the trustworthiness required for enterprise applications.

In this paper, we present several foundational components of our AI-powered data cloud architecture. First, we introduce AI Functions, which shift AI from external ML pipelines into core database primitives (Section 2). By enabling high-quality, declarative, in-place inference, we significantly reduce the operational complexity associated with traditional external AI pipelines. This capability is deeply integrated with a modern search stack that seamlessly combines structured data filters with advanced vector search to handle complex queries transactionally: AlloyDB AI-powered search [14]) and grounded retrieval-augmented generation (RAG) (Section 3). Next, we claim that the reliability of AI-integrated products is largely dependent on a rigorous semantic foundation (Section 4). By anchoring non-deterministic LLM reasoning in a deterministic layer of universal metadata catalogs and rich semantic models, the system provides enterprise-grade context and verifiability. Next, we introduce natural language interfaces for our data services that leverage and move beyond simple natural language-to-SQL/Code (NL2X) generation (Section 5). These include conversational analytics and agentic interfaces, where agents solve complex business problems through multi-turn reasoning, supported by the emerging Model Context Protocol (MCP) [15] and the Antigravity [16] ecosystem. The semantic models are also important for proper governance. By bridging these functional areas with a robust semantic foundation ([1], Google is creating a scalable AI Agentic Data Cloud ecosystem, enabling intelligent and autonomous agents and AI applications (Section 6).

We will conclude this paper (Section 7) with a discussion on active research areas and open challenges to fully realize the power of data agents and AI applications.

2 AI Functions

Relying on AI agents layered over databases is insufficient for handling data at scale. To truly integrate structured and unstructured data while delivering both precise and qualitative insights, we need a fundamental architectural shift: evolving database internals to embed AI primitives directly into the core processing engine. By doing so, these engine-level primitives act as a foundation for agents, allowing them to delegate complex ambiguities and 'fuzzy' logic to the underlying database. Recently, several academic projects also emerged that follow a similar paradigm, including Lotus[17], Palimzest[18], DocETL [19], ThalamusDB[20].

Imagine an agent that is given a database of house listings and is tasked to *"Find houses in bay area neighborhoods with good public schools that cost less than 1M"*. The cost criterion can be formalized into a classic SQL filter expression without problem. However, whether a neighborhood has good public schools or not is a subjective notion that needs to be evaluated by an LLM. If the database lacks a *school_rating* column, a traditional database engine is stuck. The `AI.IF` function allows the agent to perform semantic filtering on every row, here for every house address. `AI.IF` uses LLM's parametric knowledge about the world to come up with a common sense understanding of "good schools" in the area. In effect, these operators break the closed-world assumption of databases, as the answer to the

[1] https://cloud.google.com/dataplex

question is no longer only contained in the database, but can be inferred from LLM's pre-acquired world knowledge. Furthermore, this examples also highlights a requirement for dynamic grounding, where the agent or the engine may need to fetch real-time external information to ground its semantic evaluation in current facts.

BigQuery, AlloyDB and Spanner all have introduced a suite of new SQL functions, as summarized in Table 1, which take natural languages instructions as input and are evaluated using LLMs. Unlike most SQL functions, AI functions are multi-modal and can operate on semi-structured data (e.g., logs) or unstructured data (e.g., images, documents). Listing 1 and 2 show two typical use-cases: LLM-based classification over free-form text, and matching text with images.

Table 1: The growing list of AI functions available in AlloyDB and BigQuery.

AI Function	Definition	Example Use Case
`AI.GENERATE`	General purpose projection	Translate or summarize text Generate a caption for an image
`AI.IF`	Binary (true/false) classification	Filter out negative movie reviews Join paper abstracts with supporting claims
`AI.CLASSIFY`	Classification using user-provided classes	Bin support tickets into defined categories
`AI.SCORE`	Assign a numerical score based on rubric	Rate movie reviews by helpfulness
`AI.RANK`	Semantic ranking	Order search results based on relevance
`AI.AGG`	Semantic aggregation over many rows	Summarize user actions from session logs Find common complaints on many reviews

Listing 1: Categorizing customer support tickets (over 30 classes in the full query)

```
SELECT
  AI.CLASSIFY(ticket_body, [
    "Billing inquiries",
    "Quota limitations",
    "Performance degradation",
    "Other"]) AS category,
  COUNT(*) AS ticket_count
FROM support_cases
GROUP BY category;
```

Listing 2: Finding products from a catalog displayed in advertisement images with a multi-modal text-image join

```
SELECT p.sku_id
FROM advertisement_images a
JOIN products p
WHERE
  AI.IF(("The advertisement shown in",
          a.img_uri,
          "contains the product described in",
          p.text_description))
```

A key design tenet behind AI functions is that the engine — not the user — owns the cost, quality, and trust behind engine-level execution decisions. Unlike most other functions which have a single correct result, AI functions have a range of acceptable outputs. Optimizing the execution of AI functions is almost always a cost-quality trade-off. Furthermore, users need enough observability to trust the results.

For queries using AI functions, the cost of LLM calls dominates all other computation, to the point where it becomes economically unfeasible on millions of rows. BigQuery and AlloyDB can approximate LLM inference with smaller proxy models running directly on database workers [21]. Certain AI functions,

such as AI.IF or AI.Rank, can be approximated by using simple linear regression or other ML models. By labeling a small subset of sample rows from the table, a linear regression model can be trained on the embeddings of the unstructured fields, and these models can be used for the rest of the rows. If the resulting model doesn't satisfy accuracy constraints, then the engine falls back to the full LLM inferencing. In AlloyDB, proxy models are trained offline, as part of a `PREPARE` statement. In BigQuery, proxy models are trained on-the-fly by inserting sampling, labeling, and training steps in the query plan. We found that for a wide variety of AI functions prompts, the proxy model approximation trades off a small amount of quality for two orders of magnitude of reduction in costs and latency. On a query classifying the sentiment of 10 million product reviews, proxy models achieve a 329x speed up over LLM inference with on-the-fly training in BigQuery, and a 991x speed up with offline training in AlloyDB. The total cost including database compute and LLM inference is reduced by approximately 700x in both cases [21].

While current proxy model implementations focus on scalar or point wise transformations, we are investigating AI joins as the next promising optimization opportunity. For example, one might want to link product images to their description, as shown in Listing 2, or link support call transcripts to structured claim reports. Unlike scalar AI functions, LLM joins introduce a quadratic inference overhead. Even with proxy models, a naïve nested-loop evaluation across two 10,000 row tables requires 100-million inferences. Others have tried to tackle this problem using pre-filtering, semantic pruning mechanisms or using block nested loops with batching [17, 22, 23]. Despite the promise of these early approaches, significant challenges still remain. The efficacy of these methods is bound by the fidelity of the embedding or the feature space capturing the relationship between the tables and join conditions. Furthermore, if the join selectivity is too low, the semantic pruning mechanisms may not prune enough data and the system still spends massive resources processing "near-relevant " items. We also note that AI joins can be used for many different use cases, as also identified by [22]. As such, we envision that there is not going to be one algorithm for AI joins, but rather a suite of them covering different use cases and tradeoffs. Just as a traditional query optimizer chooses between Hash Join, Merge Join, and Nested Loop Join based on table size and indexing, a semantic query engine will need to choose from a library of AI join algorithms tailored to different data and task characteristics.

While AI-integrated data engines are in their infancy, their evolution will undoubtedly reveal new use cases and challenges. Benchmarks like SemBench [24] are created to accelerate this development. As these systems mature, they will transcend basic tasks like classification to unlock entirely new reasoning capabilities. While some users currently favor the simplicity of AI functions over the rigorous guarantees of classic machine learning, the true breakthrough is architectural. By replacing the fragmented, multi-stage pipeline of extracting and cleaning unstructured data with a single, integrated engine, AI engines offer a streamlined, single-stop solution for the next generation of data analysis.

3 The Modern Search Stack

AI-powered semantic search has been the engine behind billion-user experiences like Google Search and YouTube for years. We have brought that same vector search technology into the Data Cloud's data services. At the same time, Data Cloud customers pose multiple novel requirements that induce respective research challenges and opportunities:

- The searches combine unstructured and multimodal data with structured data. This enables complex, real-world queries, such as, in the Target.com (which is a Data Cloud customer [2]) search

[2]https://cloud.google.com/blog/topics/retail/from-query-to-cart-inside-targets-search-bar-overhaul-with-alloydb-ai?e=48754805

scenario, finding a "black stylish turtleneck" (semantic search on image and text) that is also "available at a nearby store" (structured data filter).

- Customers have varying requirements for when they want to be on the quality/cost tradeoff, which is closely related to the quality/speed tradeoff. This, in turn, invites multiple search-related technologies.
- Transactional consistency between the unstructured/multimodal data and their vectors is a requirement, serving both quality and ease-of-use.

Best-in-class database vector search for each use case Database vector search has to provide transactional consistency. Furthermore (unlike popular vector search libraries) it has to work even when the index cannot be fully cached in main memory. Naturally, this leads to utilizing the paging system of each cloud service. But then we have a third requirement: If there is enough memory to cache the index, the performance should be commensurate with the top main memory libraries, and if there is not enough memory, it should degrade gracefully. This requirement led us utilizing the AlloyDB Columnar Engine for storing the index to avoid page overheads.

The wide diversity of Data Cloud customer databases induces a fourth challenge by our commitment to provide the best database vector search for each use case: Vector search has been dominated by two families of indices and respective algorithms: Graph-based ones, where AlloyDB, CloudSQL PostGres and MemoryStore feature HNSW[25], and tree-based ones, where AlloyDB, BigQuery, Spanner, CloudSQL MySQL feature ScaNN[3] - the index that is behind Google Search and Youtube. Notice AlloyDB offers both and for good reason: Each index performs better in different use cases. Roughly, ScaNN is much better for “low dimensionality" (eg, 128d) vectors, across the board, is better for updates and build-time and is also better for scaling-out. At the same time, the query speed minded customer with high dimensionality vectors (e.g., 1576d) that can cache the index can get better performance with AlloyDB's pgvector HNSW that uses the Columnar Engine. Research is needed on how to (a) automate/advise on proper indexing and (b) a declarative interface that hides the differences between the two indices.

Combine structured filters with vector search In Data Cloud services you get top-tier retrieval over mixed data with a single, familiar SQL query that performs filtered vector search. Such queries seamlessly combine filters, joins, and top-k vector search.

Furthermore, AlloyDB AI's query processor spearheaded performance and quality improvements for such mixed queries. While filtered search is convenient, ensuring fast performance is a hard query optimization problem. The challenge lies in deciding between three methods:

- Pre-filtering: Best when few rows match the filter.
- Post-filtering: Best when many rows match the filter.
- Inline-filtering: Best for mid-selectivity queries by simultaneously processing filters and vector rankings.

The problem is that estimating which method is best is difficult[26], especially because what truly matters is the filter's selectivity on the neighborhood of the query vector - rather than the selectivity of the filter in the entire dataset. AlloyDB AI solves this with *Adaptive Filtering,* which optimizes the query plan once it learns the actual filter selectivity as it accesses data, and then can appropriately switch between the filtered vector search methods.

[3] github.com/google-research/google-research/tree/master/scann

AI functions for filtering and ranking and optimizing the quality/cost tradeoff As we discussed in Section 2, AI functions provide a novel ability for filtering and ranking, i.e., for the two core aspects of search. With AI functions the recall and precision can hit levels that are impossible with plain vector search or hybrid search (vector search + text search). However, running AI functions is linear in the number of rows, while Approximate Nearest Neighbor (ANN) search is roughly logarithmic. Furthermore, a quality/cost tradeoff is applicable to both AI functions and ANN vector search: As we saw in Section 2, proxy models deliver massive cost optimization for AI Functions. At the same time, ANN vector search is inherently a recall/cost (or speed) tradeoff even for fixed vector size. Adding the availability of Matryoshka [27] embeddings by Gemini, where it is up to the customer what prefix (i.e., how many dimensions) of the generated embedding to use, a challenging optimization problem appears: Optimize for quality and cost in a funnel that combines vector search, as its low cost method, with AI Functions being the high cost/high quality methods.

4 Universal Catalog & The Semantic Foundation

Agentic data workflows are reminiscent of federated and data integration systems[28] that also attempted to answer queries by combining data from multiple sources. To understand the components and challenges of the agentic enterprise, it is instructive to compare it to these older architectures. Data integration systems relied heavily on descriptions of data sources in order to reformulate a user query to the available sources. These descriptions were written in formal languages, which were usually some version of SQL with additional markup. They, then, used a carefully crafted algorithm to reason about these descriptions and select the relevant data sources [29]. Today, we are in a much more exciting era, where data source descriptions can be much richer. In particular, the descriptions can use natural language to describe any necessary nuance about the semantics of the data, even those that were not expressible in the formalisms of the past. This is possible because the reasoning algorithm is the LLM that powers the agent, which is capable of reasoning about natural-language descriptions. The collection of metadata we have about the underlying data is stored in a metadata catalog. Google Cloud offers a metadata catalog called Dataplex[4] which we describe below, but in principle, an AI-enabled enterprise should be able to access multiple catalogs that cover all of the customer's data estate.

It is crucial to emphasize the importance of rich source descriptions. Even as the LLMs become more powerful, they will not be able to discover and process the underlying data correctly unless they have rich metadata about the data semantics. The fundamental reason that, unlike text documents that are mostly self-describing, the semantics of tables are often opaque and not encoded in the table itself. More specifically, the table itself does not describe reasons that the table was created, its intended use and some of the assumptions underlying it. Even the aspects of the semantics that are explicit, like the table and column names, can be opaque and and often use cryptic jargon. As a simple example of where nuanced understanding of tables matters, we often see cases where we have many tables whose schema looks the same, but the differences between them are subtle, making it very easy for an agent to choose inappropriate data.

The catalog contains any kind of signal that tells us more about the data. This includes the schema and the text descriptions of the table and its columns. In addition, the catalog contains the lineage of tables, any signal about their quality or how frequently they are updated. Recent usage statistics also tell us whether the table is a useful one or not. Some important information can be farther flung. For example, there may be text in corporate documents that refer to a table and tell us exactly why it was created and under what assumptions.

[4] https://cloud.google.com/dataplex

Conceptually, it is useful to distinguish multiple components of Dataplex or other systems like it. The *harvesting* component collects all the metadata it can find from a variety of sources. For example, the harvesting component may use an LLM to enrich the table and column description by leveraging a glossary of business terms for the business. The harvester will fetch the lineage of the tables and keep track of its usage patterns, as well as navigate to any documents that may contain useful information. Query histories also provide rich semantic information about the data assets, and key business metrics. The harvester can identify join paths by analyzing query histories, or extract examples to use for in-context learning.

The *storage* component of the catalog offers a set of APIs for accessing and updating the metadata. We note that Dataplex stores metadata about relational tables, but also about other collections of unstructured data that users upload to Google's object storage. Metadata is kept up-to-date in near real time with the mutations to the datasets it refers to, via a streaming Pub/Sub interface. Data storage and database systems publish notifications about newly created or just modified data assets to a Pub/Sub topic and a Dataplex subscriber effects the appropriate metadata updates, automatically keeping metadata consistent[5] with the evolution of a customer's data estate.

The *search* component of the catalog offers a semantic search engine that given a query in natural language, will return a set of tables (or other data resources) that a relevant to the query. The search works by embedding the metadata about each resource as a vector and performing nearest-neighbor search with the embedding of the query.

The last component of Dataplex, the *context* engine (often called a *metadata reasoner*), is the one that is relatively nascent. The goal of the context engine is the following. Given a task given by the agent (such as a natural-language query), the engine should return a set of sources that together can be used to address the task. The context engine relies on the underlying search engine to produce candidate resources, but adds a layer of reasoning. For example, executing a task typically requires a combination of data sources rather than a single one. The context engine guarantees that the collection of data sources together covers all the entities and columns needed for the task, and that the data sources can be combined (e.g., via joins or unions) to yield the correct result. The context engine autonomously decides which slices of metadata in Dataplex to consult in order to guarantee the coverage and quality of its answer. For instance, in one case it may consult a tool that can determine whether there is a join path between two tables, while in another it may consult a tool that verifies that a table has been updated recently.

A discussion of metadata would be incomplete without a mention of semantic models. A semantic model is typically a graph representation of sets of entities and the relationships between them. Semantic models vary in their expressive power. The most common ones are those that identify common join paths (and hence relationships) among tables and define precisely how to compute metrics for the business from the available tables. The latter is critical for proper governance of the enterprise. More expressive semantic models borrow features from ontology languages, such as concept and relationship hierarchies and constraints on relationships between entities. A single semantic model doesn't necessarily cover the entire landscape of data of a customer, and multiple models may exist side by side. The semantic models available are also used by the context engine to improve the interpretation of queries.

Historically, one of the challenges to adoption of semantic models has been the human effort required to build and maintain them, especially as the data landscape in a corporation is constantly evolving. However, it is possible to at least bootstrap the process of creating a semantic model with the help of AI. The benefit of semantic models is that they are closer to the conceptual model in which people, and thereby possible agents, think about their data estate, because they make domain relationships more explicit. While several vendors offer tools for creating semantic models, and they are clearly valuable,

[5] eventual consistency with low convergence latency

Table 2: Natural Language to SQL example

Natural Language Task	Translated SQL

Which budget allowed the most money for water, chips, and cookies?	`SELECT T2.budget_id`
	`FROM expense AS T1`
	`INNER JOIN budget AS T2`
	`  ON T1.link_to_budget = T2.budget_id`
	`WHERE T1.expense_description =`
	`  "Water, chips, cookies"`
	`ORDER BY T1.cost DESC NULLS LAST`
	`LIMIT 1`

the precise impact of these models on agentic workflows is a topic that will be studied in the upcoming years.

5 SQL Query and Code Generation & Conversational Analytics

In this section, we describe natural language (NL) interfaces used to interact with and analyze the enterprise data assets. We first discuss SQL generation, followed by code generation to other languages, such as Python, and complete with the description of our conversational analytics agents.

SQL Query Generation: Today's AI agents interact with the data cloud via natural language (NL), which necessitates robust mechanisms to bridge the gap between NL input and the query languages supported by data cloud systems. The fundamental building block here is translation from NL to SQL – commonly referred to as NL2SQL — the primary and most established approach for relational databases and analytical platforms. There has been quite a lot of work on NL2SQL in recent years[30, 31] revived by the emergence of powerful LLMs [32]. But the scope extends beyond standard SQL covered in these works. Modern data architectures allow for full-text and vector search queries, graph queries, and the AI functions for unstructured data described in Section 2. Moreover, NoSQL databases offer proprietary APIs and query languages, and developers write Python code that interact directly with databases via libraries like Pandas. The fundamental technical challenges in achieving high-quality translation from natural language remain largely the same across all these query types.

Table 2 illustrates a typical NL2SQL task where a user's question is translated to a database query which provides the answer. This examples comes from the BIRD-Bench [33] "Student Club" dataset.

The typical approach used at Google for solving NL2SQL problems uses large language models as the main SQL generators, depending on the use case taking an agentic approach to compose various subtasks [32, 34]. A variety of RAG techniques is used to retrieve the relevant context the LLM needs to make the right interpretation of the question – schema information, data samples, query examples, descriptions and guidelines etc. These systems need to address a series of challenges:

- Syntactic correctness: The first requirement is that the LLM produce executable SQL. With state of the art large language models, we typically observe that the models are very good at understanding SQL syntax and generating syntactically correct SQL, similar to other code generation skills. Where they fall short is typically when it comes to newer, non-standard features, such as vector search queries, that look subtly different across different SQL dialects. This can be addressed by incrementally including more diverse training data into training of the large language model, or by

using in-context learning or RAG techniques pulling in examples or documentation for how to write the specific type of query at runtime.

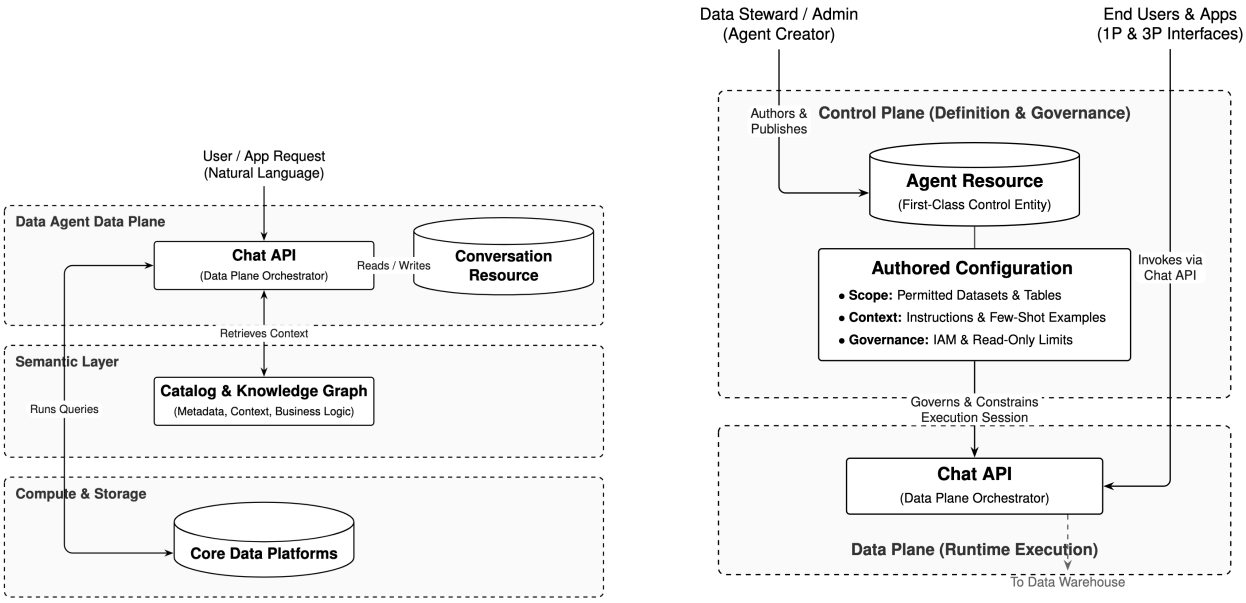
- Semantic correctness: While syntactic correctness is a relatively straightforward problem with a deterministic correct answer, having the model understand the user intent and the semantics of how that intent maps to the data is more difficult. In the example above the SQL generator needs to understand that "allow the most money" maps to the largest value in *expense* table *cost* column. Additionally, the model needs to figure out how to join the *expense* and *budget* tables. This gets particularly challenging in enterprise applications where business terms may have very specific definitions that don't align well with the world knowledge of the models. The solution to this problem is to build up additional context of the data, explaining to the model what can't be conveyed through the traditional schema concepts. This rich semantic information comes from the universal catalogs and semantic models, as discussed in Section 4. Context can be *descriptive*, for example a natural language comment explaining what is stored in a particular table or column, or *prescriptive*, a templated SQL query showing exactly how to answer a specific task. It can be free-form text or more formalized as explicit semantic models attached to the database objects.

- Disambiguation: Unlike SQL, natural language is inherently ambiguous. In the example above, the caller is asking for "which budget" without a detailed explanation for exactly what they are looking for. The system's interpretation of this as an id key column is a reasonable choice, although not clearly specified. To provide correct answers the SQL generator needs to be able to detect ambiguities and either ask the user for clarification when needed, or explain the intent behind its interpretation of the question to avoid giving a misleading answer.

The example above also highlights the need for NL2SQL system to inspect the data itself, not just metadata. This is especially needed for *value linking*: the process to match concepts and entities in the natural language query (ẅater, chips, and cookies) to the correct SQL literals (`T1.expense_description = Ẃater, chips, cookies´`). To solve this problem, the SQL generator needs to have a way to explore the actual database content, which may require pre-processing, such as creating custom indexes or sampling data. The right technique depends on the data and the application, and is not easily generalizable.

Having good benchmarks is critical for understanding these challenges, how they interact and how to best evolve the NL2SQL systems. Google uses a broad mix of benchmarks, from academic standards like BIRD, and its more advanced descendants like BIRD-INTERACT [35], to custom in-house developed benchmarks that focus more on enterprise and specific NL2SQL use cases.

Code Generation: While SQL remains the most frequently used interface for accessing relational data, data analysts and scientists heavily rely on imperative programming (e.g., Python) for advanced analytics and predictive modeling. Consequently, NL2X must seamlessly extend to natural language to code (NL2Code) translation, targeting frameworks such as Python's Pandas, PySpark, and Scikit-Learn [36–38]. Unlike SQL, a declarative interface, data science code is procedural. The underlying agent must map user intent to the correct API calls and also reason about execution state and data flow across multiple steps. Critically, similar to NL2SQL, it also must understand the underlying data semantics (e.g., real-world meaning of columns, implicit relationships and business logics) to ensure the generated steps correctly fulfill the user's intended analytical goals. The challenge here is that the model often predicts the next token rather than understanding the semantically accurate structures of the code and the data. This can lead to code that compiles, but does not fulfill the user's goals [39]. This procedural requirement aligns with our conversational data analytics pattern, "investigate, then report" agentic workflow. Translating natural language to complex procedural code remains an active area of



(a) Data Plane

(b) Control Plane

Figure 1: Conversational Analytics Architecture Overview

research [40–43]. While LLMs demonstrate remarkable coding proficiency, understanding data semantics and accurately evaluating the quality and the fidelity of the generated code introduces significant challenges. Unlike SQL generation, where semantic correctness can often be verified by comparing query execution result sets, data science code is open-ended; a single analytical task can be correctly solved using many different approaches or library/API calls combination.

Conversational Analytics: While traditional NL2SQL systems map user intent to single executable queries, real-world analytics requires a dynamic approach known as conversational analytics [6] – an ongoing process of iterative exploration, hypothesis testing, and visualization. To support this, our conversational analytics architecture evolves the discrete NL2SQL translation step into a continuous, autonomous loop using an "investigate, then report" agentic workflow pattern. Rather than simply returning an SQL string, the conversational analytics agent formulates a plan, executes queries against the engine, and observes the results. This enables the agent to autonomously explore, backtrack, and self-correct—such as actively probing the database to disambiguate string values—before exposing fully vetted results to the user. Beyond tabular data, this open-ended agentic loop integrates specialized tools. It leverages sandboxed code execution environments to dynamically render charts, and consults up-to-date documentation to construct queries using advanced engine capabilities like BigQuery Machine Learning (BQML [7]) or Graph Query Language (GQL [8]). Once the investigation concludes, the "report" phase synthesizes the executed queries, data payloads, and generated charts into a cohesive, narrative-driven final response, effectively mirroring the workflow of a human data analyst.

The operational execution of these autonomous workflows relies on a multi-layered data plane (See Figure 1a) designed for scalability and reliability. At the foundation are the underlying source systems, such as BigQuery or AlloyDB, which provide the raw storage and compute capabilities. Above this

[6] https://docs.cloud.google.com/bigquery/docs/conversational-analytics
[7] https://cloud.google.com/bigquery/docs/bqml-introduction
[8] https://docs.cloud.google.com/spanner/docs/reference/standard-sql/graph-intro

sits the semantic layer—incorporating Dataplex and enterprise knowledge graphs—which provides the critical metadata, context, and business logic needed to ground the agent's reasoning. The topmost layer is the data agent data plane itself, exposed primarily via the Chat API and managed through persistent conversation resource. The Chat API acts as the central orchestrator for user interactions. When a natural language request is received, the data plane retrieves the relevant scope from the semantic layer and initiates the agent's investigative loop. It translates the agent's reasoning into read-only queries that are executed securely against the source systems, buffering intermediate results for analysis. Simultaneously, the data plane manages multi-turn state by reading and writing to conversation resource. This ensures that conversational context is maintained across interactions while keeping execution histories, user prompts, and queried data securely isolated.

While users can chat directly with their organization's raw data, enterprise scale requires more structured governance. Customers need the ability to provision, specialize, and monitor role-specific data agents—such as a "financial forecasting agent" or a "supply chain diagnostic assistant" – control them with fine-grained Identity and Access Management (IAM) policies, and deploy them to surfaces both within Google Cloud Platform and via external APIs. The Conversational Analytics control plane (See Figure 1b) fulfills this need by elevating data agents to first-class, governable resources within the data cloud. Furthermore, this provides organizations a safe environment to experiment with and customize agent behavior in a scoped workspace, completely separate from the organization's broader ground-truth data. Crucially, this control plane ensures that autonomous agents operate strictly within the enterprise's security and governance boundaries. Because the agent executes actions directly against the database on behalf of the user, it is constrained by the same IAM and context-aware access policies as a human operator. To prevent malicious or accidental data mutation, execution tooling utilizes database dry-runs prior to execution, structurally enforcing that generated queries are read-only (e.g., restricted to `SELECT` statements) and that all accessed tables fall within the user's permitted scope. This platform-centric approach ensures that Conversational Analytics is ubiquitous across the user ecosystem. These specialized, governed agents are embedded directly into graphical interfaces, such as BigQuery Studio, allowing users to interact with their data seamlessly within existing workflows. Additionally, the control plane exposes these capabilities via stateless and stateful APIs, enabling enterprise developers to programmatically invoke governed data agents and inject natural language analytical reasoning directly into custom applications.

6 Data Agents & Agentic Ecosystem

In this section, we introduce the suite of specialized built-in data agents designed to handle complex workflows, and describe the broader agentic ecosystem that supports their customization and orchestration.

Data Agents: Besides the Conversational Analytics agents mentioned in Section 5 above, Data Cloud systems embed a number of other built-in data agents, designed to assist users with data-intensive tasks such as data science, data engineering, or data governance.

The *data engineering* agent[9] automates data preparation and data cleaning tasks. It supports complex data transformations and AI-assisted generation, modification, troubleshooting, and optimization of data pipelines.

The *data governance* agent was designed to assist with proactive governance: instead of waiting for compliance gaps, the system actively suggests classification, access controls, lineage tracking, and policy updates. The data governance agent also carries out some of the tasks of the harvesting component of

[9]https://cloud.google.com/blog/products/data-analytics/exploring-the-data-engineering-agent-in-bigquery

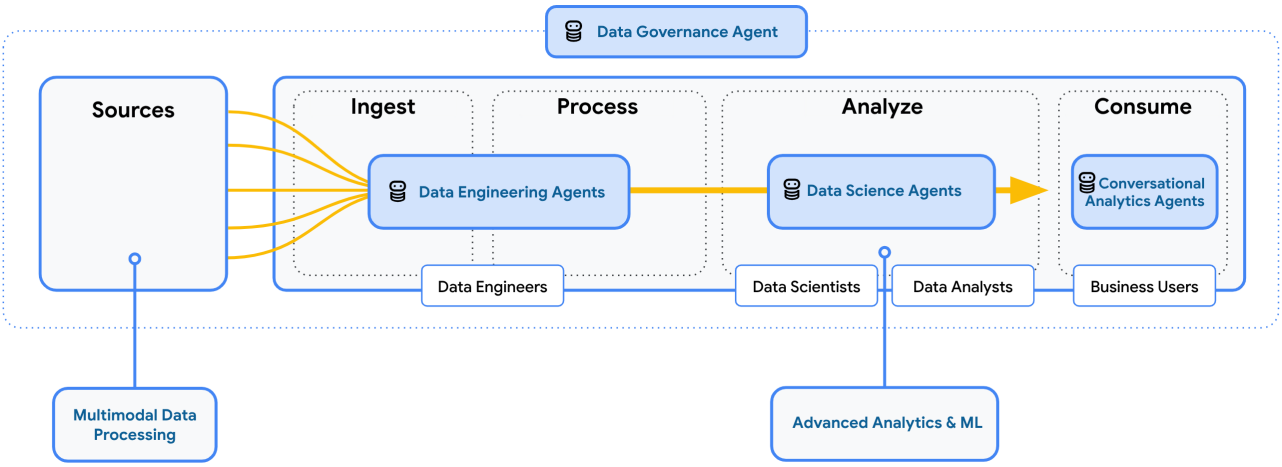


Figure 2: Built-in data agents in Google Data Cloud.

the Dataplex catalog, described in Section 4 above. These include automatic metadata generation such as table and column descriptions, finding common join patterns in the query history, or profiling the data in support of *value linking*.

The *data science* agent[10] helps streamline manual and repetitive data science tasks. It supports exploratory data analysis, highlighting patterns in the data via rich visualizations, as well as development and evaluation of statistical, timeseries, and predictive ML models. The data science agent leverages the full power of Colab notebooks via Python code generation and makes use of the rich variety of libraries available in the Python ecosystem for data science, machine learning, and advanced visualizations. To enable scaling to very large datasets, the data science agent can generate PySpark data pipelines and automatically provision serverless Spark clusters for distributed parallel execution. Alternatively, the data science agent can generate BigFrames[11] code, which transpiles to SQL and executes on the scalable BigQuery execution engine. The data science agent can effectively generate BQML[12] code to both take advantage of advanced built-in models (e.g., TimesFM for time series analysis and forecasting) as well as train custom models from the user's datasets and use them for inference. Improving the quality of the data science agent is an area of active research[44].

Agentic Ecosystem: The true power of any agentic ecosystem lies in its extensibility, allowing users to deeply integrate and customize their own data agents alongside the platform's built-in offerings. Users can tap into this ecosystem through several key components:

- *Custom data agent creation & hosting:* Developers can build their own fully customized data agents using the Agent Development Kit (ADK[13]) and the Antigravity[14] ecosystem, and they can deploy, manage, and scale agents in production with Agent Engine[15]. A low-code alternative to developing custom data agents, is to package specialized context and system instructions in custom Conversational Analytics agents using the control plane API introduced in section 5 above (figure 1b).

[10] https://cloud.google.com/blog/products/ai-machine-learning/ai-first-colab-notebooks-in-bigquery-and-vertex-ai
[11] https://docs.cloud.google.com/bigquery/docs/bigquery-dataframes-introduction
[12] https://docs.cloud.google.com/bigquery/docs/bqml-introduction
[13] https://google.github.io/adk-docs
[14] https://developers.googleblog.com/build-with-google-antigravity-our-new-agentic-development-platform
[15] https://docs.cloud.google.com/agent-builder/agent-engine/overview

- *Tool integration:* Developers can equip their custom agents with Data Cloud functionalities by utilizing the MCP (Model Context Protocol) Toolbox for Databases[16], which provides a suite of prebuilt and custom tools that enable AI agents to securely and efficiently interact with over 40 different databases, including AlloyDB, Cloud SQL, Spanner, BigQuery, MySQL, PostgreSQL, Neo4j, MongoDB, and more.

- *Agent delegation:* Through the A2A[17] (Agent-to-Agent) protocol, users can connect their custom agents to Data Cloud's built-in specialized agents, allowing them to smoothly delegate complex data tasks. Looking forward, this highly interoperable ecosystem paves the way for complex agent meshes and networks, where built-in Data Cloud agents can seamlessly act as specialized sub-agents for custom data agents across the enterprise. Effective and efficient orchestration of multi-agent systems is an active area of research[45–50].

While disambiguation capabilities help agents clarify user requests, repeatedly forcing users to answer the same clarifying questions can quickly become frustrating. To mitigate this, agentic data systems must learn from their user interactions over time, thus *memory* is a critical mechanism for all data agents[51]. Crucially, the system synthesizes the semantic context gleaned from individual user conversations and makes it available to all users within a customer's organization. This shared organizational memory ensures that new users, or those who are less experienced with the specific nuances of the company's data estate, are effectively assisted from their very first interaction, eliminating the cold start problem.

An area of current and future work regards enhancing autonomous decision-making via specialized agentic skills[18]. Realizing this vision, however, introduces several significant technical challenges, particularly in how skills are autonomously discovered, composed, and continuously updated. Recent research has increasingly focused on dynamic skill acquisition and hierarchical reasoning trees to move beyond static, handcrafted workflows; yet, elevating LLM capabilities from simple API tool-calling to the robust composition of generalized, reusable skills remains a complex hurdle [52]. Furthermore, automating the discovery of these new skills in open-ended environments—without relying on predefined task boundaries or human demonstrations—poses a major exploration challenge that recent systems attempt to address through exploration-first strategies [53]. Additionally, there is the ongoing technical challenge of continuously evolving and self-improving these skill libraries, with active research exploring recursive reinforcement learning methods to iteratively build and refine more complex agentic behaviors over time [54].

Within an enterprise data ecosystem, these challenges extend to accurately grounding these skills in complex data environments. Using the enterprise catalog's knowledge graph, the system could create semantic mappings between unstructured or tabular data assets and specific recipes for performing common tasks on these datasets. However, the remaining technical difficulty lies in enabling agents to dynamically and selectively retrieve and inject only the most relevant task recipes or playbooks directly into their limited context window, which is critical for significantly improving task reliability without degrading model performance.

7 Conclusion

In this paper, we present several foundational components of Google's AI-powered Data Cloud architecture. In particular, the AI functions, the modern search stack, universal catalog and semantic models are core building blocks for developing intelligent agents and AI applications. In this agents-oriented system,

[16] https://github.com/googleapis/genai-toolbox
[17] https://developers.googleblog.com/developers-guide-to-ai-agent-protocols
[18] https://agentskills.io/home

natural language is the new way to interact with the services, while the system complexities are expected to be hidden from the customers. We believe the future will be agentic and the ease of use will be a key differentiator.

In this paper, we also identify several research challenges that need to be addressed for this vision. AI functions enable operational databases and analytical platforms to extract information from structured, unstructured and multimodal data and execute deep analytics and powerful semantic search. We presented our list of new AI functions, but we believe there are opportunities for new functions packaging LLM abilities in novel ways that compose better and unlock new use-cases. Each of these new functions can benefit from novel implementations that use the LLMs in most efficient ways. Moreover, these new functions, and mixing them with relational operations, further complicate query optimization. The optimization problem is no longer just minimizing latency, but also maximizing accuracy. This new dimension creates a pareto frontier that requires new optimizers [55]. The optimization of the quality/cost tradeoff is particularly accurate for search that combines structured data and unstructured/multimodal data. The modern search funnel includes ANN vector search as its low cost / medium quality layer and AI Functions for filtering and ranking as the medium cost or high cost, but also high quality, layer. Furthermore, it is important that we make SQL (as) declarative (as possible) again and get rid of the knobs [56].

Understanding the data semantics and business logic remains the main technical challenge for NL2X and conversational analytics agents. Universal catalogs and semantic models that capture these enterprise-specific information is critical to map the user intent into the correct set of tables and columns, and identify correct literals to use in predicates. How to collect and organize the most effective metadata, and how to reason about it efficiently at scale are still challenging problems. While models have evolved to correctly generate standard SQL found in many benchmarks, they need additional context, and tuning to generate recent SQL extensions, including vector search queries, graph queries and queries with AI functions.

Agents are transforming the enterprise architectures to solve many enterprise data problems, including data engineering tasks, data science tasks, and data governance tasks. For all data agents, skills are emerging as the new mechanism to solve many subtasks. Figuring out how and when to use skills effectively and efficiency is an open challenge. While data science agents are solving critical problems, they still have room for improving their accuracy. Finally, many agentic solutions are composed of multiple agents, and effective and efficient orchestration of multi-agent systems is an area that warrants more research.

References

[1] F. Özcan and Y. Chung and Y. Chronis and Y. Gan and Y. Wang and C. Binnig and J. Wehrstein and G. Kakkar and S. Abu-el-haija, "LLMs and Databases: A Synergistic Approach to Data Utilization," *IEEE Data Eng. Bull.*, vol. 49, no. 1, pp. 32–44, 2025.

[2] S. Mishra and Y. Papakonstantinou, "Alloydb ai drives innovation for application developers," Google Cloud Blog, April 2025, accessed: 2026-02-17. [Online]. Available: https://cloud.google.com/blog/products/databases/alloydb-ai-drives-innovation-from-the-database

[3] A. Verma and J. Burr. (2024, Apr.) SQL reimagined for the AI era with BigQuery AI functions. Google Cloud Blog. Accessed: Feb. 27, 2025. [Online]. Available: https://cloud.google.com/blog/products/data-analytics/sql-reimagined-for-the-ai-era-with-bigquery-ai-functions

[4] S. Yao *et al.*, "ReAct: Synergizing reasoning and acting in language models," in *International*

Conference on Learning Representations (ICLR), 2023, foundational for reasoning and tool use integration.

[5] T. Schick *et al.*, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023, details how LLMs learn to call APIs for external execution.

[6] S. Liu, S. Ponnapalli, S. Shankar, S. Zeighami, A. Zhu, S. Agarwal, R. Chen, S. Suwito, S. Yuan, I. Stoica *et al.*, "Supporting our ai overlords: Redesigning data systems to be agent-first," *arXiv preprint arXiv:2509.00997*, 2025.

[7] Y. Zhu, L. Wang, C. Yang, X. Lin, B. Li, W. Zhou, X. Liu, Z. Peng, T. Luo, Y. Li *et al.*, "A survey of data agents: Emerging paradigm or overstated hype?" *arXiv preprint arXiv:2510.23587*, 2025.

[8] M. Rahman, A. Bhuiyan, M. S. Islam, M. T. R. Laskar, R. Mahbub, A. Masry, S. Joty, and E. Hoque, "Llm-based data science agents: A survey of capabilities, challenges, and future directions," *arXiv preprint arXiv:2510.04023*, 2025.

[9] P. Liskowski, B. Han, P. Aggarwal, B. Chen, B. Jiang, N. Jindal, Z. Li, A. Lin, K. Schmaus, J. Tayade *et al.*, "Cortex aisql: A production sql engine for unstructured data," *arXiv preprint arXiv:2511.07663*, 2025.

[10] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE intelligent systems*, vol. 24, no. 2, pp. 8–12, 2009.

[11] Q. Zhang, C. Hu, S. Upasani, B. Ma, F. Hong, V. Kamanuru, J. Rainton, C. Wu, M. Ji, H. Li *et al.*, "Agentic context engineering: Evolving contexts for self-improving language models," *arXiv preprint arXiv:2510.04618*, 2025.

[12] L. Mei, J. Yao, Y. Ge, Y. Wang, B. Bi, Y. Cai, J. Liu, M. Li, Z.-Z. Li, D. Zhang *et al.*, "A survey of context engineering for large language models," *arXiv preprint arXiv:2507.13334*, 2025.

[13] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 7, pp. 3580–3599, 2024.

[14] Google Cloud, "What is AlloyDB AI?" Google Cloud Documentation, February 2026, accessed: 2026-02-17. [Online]. Available: https://docs.cloud.google.com/alloydb/docs/ai/what-is-alloydb-ai

[15] R. D. Amit Ganesh. (2025, Feb.) Powering the next generation of agents with google cloud databases. Google Cloud Blog. Describes the integration of Model Context Protocol (MCP) to standardize agent-database interactions. [Online]. Available: https://cloud.google.com/blog/products/databases/managed-mcp-servers-for-google-cloud-databases

[16] Google Antigravity Team. (2025, Nov.) Build with Google Antigravity, our new agentic development platform. Google Developers Blog. Announcement of a task-oriented IDE designed for autonomous agent orchestration. [Online]. Available: https://developers.googleblog.com/build-with-google-antigravity-our-new-agentic-development-platform/

[17] L. Patel, S. Jha, M. Pan, H. Gupta, P. Asawa, C. Guestrin, and M. Zaharia, "Semantic operators and their optimization: Enabling llm-based data processing with accuracy guarantees in lotus," *Proc. VLDB Endow.*, 2025. [Online]. Available: https://doi.org/10.14778/3749646.3749685

[18] C. Liu, M. Russo, M. J. Cafarella, L. Cao, P. B. Chen, Z. Chen, M. J. Franklin, T. Kraska, S. Madden, R. Shahout, and G. Vitagliano, "Palimpzest: Optimizing ai-powered analytics with declarative query processing," in *15th Conference on Innovative Data Systems Research, CIDR 2025, Amsterdam, The Netherlands, January 19-22, 2025*. www.cidrdb.org, 2025. [Online]. Available: https://vldb.org/cidrdb/2025/palimpzest-optimizing-ai-powered-analytics-with-declarative-query-processing.html

[19] J. Zhang and et al., "Docetl: Agentic query rewriting and evaluation for complex document processing," *Proceedings of the VLDB Endowment*, vol. 18, no. 9, pp. 3035–3048, 2025. [Online]. Available: https://arxiv.org/abs/2410.12189

[20] S. Jo and I. Trummer, "Thalamusdb: Approximate query processing on multi-modal data," *Proc. ACM Manag. Data*, vol. 2, no. 3, p. 186, 2024. [Online]. Available: https://doi.org/10.1145/3654989

[21] Y. Chung, R. Desai, J. He, Y. Xiao, T. Hottelier, Y.-L. K. Samo, P. Kadilkar, X. Chen, S. Idicula, F. Özcan, A. Halevy, and Y. Papakonstantinou, "100x cost & latency reduction: Performance analysis of ai query approximation using lightweight proxy models," https://arxiv.org/abs/2603.15970, 2026, accepted at SIGMOD 2026, Bengaluru, India.

[22] S. Zeighami, S. Shankar, and A. Parameswaran, "Featurized-decomposition join: Low-cost semantic joins with guarantees," *arXiv preprint arXiv:2512.05399*, 2025.

[23] I. Trummer, "Implementing semantic join operators efficiently," *arXiv preprint:2510.08489*, 2026.

[24] J. Lao, A. Zimmerer, O. Ovcharenko, T. Cong, M. Russo, G. Vitagliano, M. Cochez, F. Özcan, G. Gupta, T. Hottelier *et al.*, "Sembench: A benchmark for semantic query processing engines," *arXiv preprint arXiv:2511.01716*, 2025.

[25] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, 2020.

[26] D. Lu, H. Caminal, M. Chatzakis, Y. Papakonstantinou, Y. Chronis, V. Jain, and F. Özcan, "An in-depth study of filter-agnostic vector search on a postgresql database system," https://arxiv.org/abs/2603.23710, 2026, accepted at SIGMOD 2026, Bengaluru, India.

[27] A. Kusupati and et.al., "Matryoshka representation learning," 2022. [Online]. Available: https://arxiv.org/abs/2205.13147

[28] M. Stonebraker, I. F. Ilyas *et al.*, "Data integration: The current status and the way forward." *IEEE Data Eng. Bull.*, vol. 41, no. 2, pp. 3–9, 2018.

[29] A. Doan, A. Y. Halevy, and Z. G. Ives, *Principles of Data Integration*. Morgan Kaufmann, 2012. [Online]. Available: http://research.cs.wisc.edu/dibook/

[30] A. Quamar, V. Efthymiou, C. Lei, and F. Özcan, "Natural language interfaces to data," *Found. Trends Databases*, vol. 11, no. 4, pp. 319–414, 2022. [Online]. Available: https://doi.org/10.1561/1900000078

[31] G. Katsogiannis-Meimarakis, M. Xydas, and G. Koutrika, "Natural language interfaces for databases with deep learning," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3878–3881, 2023. [Online]. Available: https://www.vldb.org/pvldb/vol16/p3878-katsogiannis-meimarakis.pdf

[32] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. T. Kakkar, Y. Gan, A. Saberi, F. Özcan, and S. O. Arik, "CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2025. [Online]. Available: https://openreview.net/forum?id=CvGqMD5OtX

[33] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo *et al.*, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[34] Y. Chung, G. T. Kakkar, Y. Gan, B. Milne, and F. Özcan, "Is long context all you need? leveraging llm's extended context for nl2sql," *Proceedings of the VLDB Endowment*, vol. 18, no. 8, p. 2735–2747, Apr. 2025. [Online]. Available: http://dx.doi.org/10.14778/3742728.3742761

[35] N. Huo, X. Xu, J. Li, P. Jacobsson, S. Lin, B. Qin, B. Hui, X. Li, G. Qu, S. Si *et al.*, "Bird-interact: Re-imagining text-to-sql evaluation for large language models via lens of dynamic interactions," *arXiv preprint arXiv:2510.05318*, 2025.

[36] J. Nam, J. Yoon, J. Chen, J. Shin, S. O. Arık, and T. Pfister, "Mle-star: Machine learning engineering agent via search and targeted refinement," 2025. [Online]. Available: https://arxiv.org/abs/2506.15692

[37] P. Yin, W. Ding, C. Xia, L. Wang, C. Carroll, J. Li, G. Neubig *et al.*, "Natural language to code generation in interactive data science notebooks," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 141–173.

[38] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[39] Z. Wang, Z. Zhou, D. Song, Y. Huang, S. Chen, L. Ma, and T. Zhang, "Towards understanding the characteristics of code generation errors made by large language models," 2025. [Online]. Available: https://arxiv.org/abs/2406.08731

[40] J. S. Chan, N. Chowdhury, O. Jaffe, J. Aung, D. Sherburn, E. Mays, G. Starace, K. Liu, L. Maksin, T. Patwardhan *et al.*, "Mle-bench: Evaluating machine learning agents on machine learning engineering," *arXiv preprint arXiv:2410.07095*, 2024.

[41] F. Shu, Y. Wang, R. Wu, B. Liu, Z. Yao, Y. He, and F. Yan, "Dare-bench: Evaluating modeling and instruction fidelity of llms in data science," 2026. [Online]. Available: https://arxiv.org/abs/2602.24288

[42] D. Zan, B. Chen, F. Zhang, D. Lu, B. Wu, B. Guan, W. Yongji, and J.-G. Lou, "Large language models meet nl2code: A survey," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 7443–7464.

[43] Z. Zhang, C. Wang, Y. Wang, E. Shi, Y. Ma, W. Zhong, J. Chen, M. Mao, and Z. Zheng, "Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 481–503, 2025.

[44] J. Nam, J. Yoon, J. Chen, R. Sinha, J. Shin, and T. Pfister, "DS-STAR: Data science agent for solving diverse tasks across heterogeneous formats and open-ended queries," *arXiv preprint arXiv:2509.21825v4*, 2026.

[45] A. Adimulam, R. Gupta, and S. Kumar, "The orchestration of multi-agent systems: Architectures, protocols, and enterprise adoption," 2026. [Online]. Available: https://arxiv.org/abs/2601.13671

[46] P. Drammeh, "Multi-agent llm orchestration achieves deterministic, high-quality incident response," 2025. [Online]. Available: https://arxiv.org/abs/2511.15755

[47] G. Yu, "Adaptorch: Task-adaptive multi-agent orchestration in the era of performance convergence," 2026. [Online]. Available: https://arxiv.org/abs/2602.16873

[48] Y. Dang, C. Qian, X. Luo, J. Fan, Z. Xie, R. Shi, W. Chen, C. Yang, X. Che, Y. Tian, X. Xiong, L. Han, and M. Sun, "Multi-agent collaboration via evolving orchestration," 2025. [Online]. Available: https://arxiv.org/abs/2505.19591

[49] W. Zhang, L. Zeng, Y. Xiao, Y. Li, C. Cui, Y. Zhao, R. Hu, Y. Liu, Y. Zhou, and B. An, "Agentorchestra: Orchestrating multi-agent intelligence with the tool-evolutionary agent (tea) protocol," 2025. [Online]. Available: https://arxiv.org/abs/2506.12508

[50] K. Agrawal and N. Nargund, "Neural orchestration for multi-agent systems: A deep learning framework for optimal agent selection in multi-domain task environments," 2025. [Online]. Available: https://arxiv.org/abs/2505.02861

[51] S. Ouyang, J. Yan, I.-H. Hsu, Y. Chen, K. Jiang, Z. Wang, R. Han, L. T. Le, S. Daruki, X. Tang, V. Tirumalasetty, G. Lee, M. Rofouei, H. Lin, J. Han, C.-Y. Lee, and T. Pfister, "ReasoningBank: Scaling agent self-evolving with reasoning memory," *arXiv preprint arXiv:2509.25140*, 2025, accepted to ICLR 2026. [Online]. Available: https://arxiv.org/abs/2509.25140

[52] R. Xu *et al.*, "SoK: Agentic skills – beyond tool use in LLM agents," *arXiv preprint arXiv:2602.20867*, 2026. [Online]. Available: https://arxiv.org/abs/2602.20867

[53] A. Authors, "EXIF: Automated skill discovery through exploration-first strategy," *arXiv preprint arXiv:2506.04287*, 2025. [Online]. Available: https://arxiv.org

[54] Y. Wang *et al.*, "SkillRL: Evolving agents via recursive skill-augmented reinforcement learning," *arXiv preprint arXiv:2602.08234*, 2026. [Online]. Available: https://arxiv.org

[55] M. Russo, S. Sudhir, G. Vitagliano, C. Liu, T. Kraska, S. Madden, and M. Cafarella, "Abacus: A cost-based optimizer for semantic operator systems," *arXiv preprint arXiv:2505.14661*, 2025.

[56] M. Chatzakis, Y. Papakonstantinou, and T. Palpanas, "DARTH: declarative recall through early termination for approximate nearest neighbor search," *Proc. ACM Manag. Data*, vol. 3, no. 4, pp. 242:1–242:26, 2025. [Online]. Available: https://doi.org/10.1145/3749160