TextQL Technical Overview

Arman Raayatsanati, Matthew Abate, John B. Mains, and Mark Hay*

1 Introduction

This paper describes TextQL, an automated insights discovery platform for enterprise-level data analysis tasks. TextQL is designed to address five key challenges in deploying enterprise AI:

- **Huge numbers of data assets.** Modern enterprises often have tens of thousands of tables and a diversity of warehouses, formats, and storage locations. Even single business units or workflows may require understanding hundreds of tables and thousands of columns, as well as the ability to reason about physical properties such as specific categorical values, cardinality, join validity, etc.
- **Business-to-data context mapping.** Companies typically have large amounts of unstructured context on high-level strategic goals, current initiatives, key entities such as products, customers, or events, and more. However, even if this data exists and is accessible by a model, there is almost never a mapping of this knowledge to structured data and systems of record in a way that minimizes query ambiguity and expert human-in-the-loop guidance.
- **Autonomous, open-ended operation** Most existing systems are presented in the form of a chat interface directly with the deployed model. However, the number of useful analyses for an organization at a given point in time likely exceeds the number of humans who have a question or task they want addressed. An effective system should therefore be able to generate hypotheses and create useful insights without being fed questions from a human.
- **Reliable and transparent multi-step planning and reasoning.** Useful analyses tend to require planning and execution of multiple subtasks, taking into account available context and correcting trajectories based on mistakes or unexpected information. Moreover, these trajectories should be easily verifiable by human users.
- **Industry-grade robustness and versatile compute deployment.** Models that perform extraction and analysis on structured data require a secure computing environment that can be deployed across the variety of clouds, hardware, and network topologies seen across enterprises.

TextQL is underpinned by a comprehensive framework consisting of three core layers the Ontology Layer (TextQL Semantic Layer), the Agent Layer, and the Compute Later, as discussed in Section 2.

By integrating these innovations, TextQL significantly enhances structured data interactions, enabling precise, context-aware analytics for complex, real-world applications. This tailored solution meets the analytical and operational demands of modern enterprises, ensuring adaptability and reliability in a variety of computing environments.

2 Architecture

The TextQL platform is a comprehensive data exploration and analysis platform to support autonomous workflows over complex data sources. There are three non-trivial layers that enable these capabilities within the platform:

*A. Raayatsanati, M. Abate, J. B. Mains, and M. Hay are with TextQL Inc., 225 Broadway, New York City, NY, 10007, USA { arman, matt.abate , jbm, mark } @ textql.com

- The **Ontology Layer** provides a semantic framework for data modeling and SQL generation.
- The **Agent Layer** uses a state-machine framework to organize agent behavior and manage complex workflows.
- The **Compute Layer** handles data management, execution of arbitrary code, and orchestration of system components.

The Ontology Layer, Agent Layer, and Compute Layer function as an integrated system, enabling the transformation of natural language questions into actionable data insights through interactions between specialized subsystems.

By isolating semantic mapping in the Ontology Layer, natural language understanding to the Agent Layer, and secure execution in the Compute Layer, TextQL reliably handles tasks that would be impossible for any single system. This multi-layer design additionally addresses a fundamental challenge in agentic systems: maintaining both flexibility in user interaction and precision in computational execution. By compartmentalizing these concerns, TextQL achieves reliability that would be unattainable in end-to-end approaches where language models directly control computation, while preserving the intuitive interface that makes complex analysis accessible to non-technical users. Furthermore, the architecture supports progressive refinement, where initial exploratory queries lead to deeper analysis as the system identifies patterns in the data.

3 Ontology Layer

The TextQL Ontology is a representation layer and associated query language (OQL) that bridges the conceptual gap between raw, heterogeneous data sources and end-user analytical queries. The Ontology is designed to have the following properties:

- **Language Model Legibility.** Objects, attributes, and metrics should have names and descriptions corresponding to their business definitions, as opposed to potentially hard-to-read table names. Language models can then be easily mapped to queryable assets, AI-generated queries can be easily verified with minimal technical expertise, and even trivial context retrieval methods (i.e. trigram similarity or BM25) yield very high performance.
- **System Agnostic Querying.** Ontologies can be queried (by a model or human) in OQL without knowledge of underlying storage, permission systems, or specific SQL dialects. For a given OQL query, the query compiler can deterministically select the correct query engine, inject roles and permission-based filters where needed, and generate a syntactically correct query in the correct dialect. Links between objects in different storage systems can be defined and queried against if there is a deployed query engine that supports joining across them. Otherwise, the two objects are presented as opaquely unjoinable.
- **Abstraction and Encapsulation.** Mapping business concepts to corresponding data assets can require the use of advanced concepts such as window functions, level-of-detail queries, or functionality specific to the underlying system that language models may not write correctly or consistently. Ontology assets (particularly metrics and derived attributes) can be defined with formulae that encapsulate complex logic and used in OQL directly.
- **Ease of Generation and Verification.** An initial ontology can be easily generated for a single storage system with only a relational ERD. The TextQL platform also provides features for automatically creating or modifying ontology assets from natural language feedback or SQL query histories.

3.1 Components of the Ontology

The ontology is defined by four primary constructs:

- **Object Types:** These form the nodes of the ontology graph and represent foundational entities pertinent to business operations and analytical processes. Object types typically correspond to normalized tables in a customer data warehouse or similar entity-like assets (i.e. materialized SQL views, blob storage buckets, or REST API Queries).
- **Attributes:** Each object type is associated with a set of attributes corresponding to columns or fields in the underlying storage.

- **Links:** Represented as annotated edges, links define the joinable relations between object types. For a link, the join keys specify the attributes in each object used to establish the relationship, while the cardinality (one-to-one, one-to-many, or many-to-many) constrains the permissible join paths. Links are curated to include only plausible relational paths (for example, there cannot be a link across disjoint storage systems), ensuring computational efficiency and semantic relevance.
- **Metrics:** Metrics are quantitative measures associated with object types, containing a unique identifier, as well as formulaic expression applied to one or more attributes (e.g., involving window functions or level-of-detail calculations) in the associated object type. Metrics quantify business-relevant properties and are integral to analytical queries.

Formally, the ontology is defined as a tuple $\mathcal{O} = (O, A, L, M)$, where:

- O is a set of object types,
- A is a set of attributes
- L is a set of links,
- M is a set of metrics.

The components O and L together constitute a directed graph $G = (O, L)$, wherein O represents the nodes—each $o \in O$ corresponding to a fundamental business entity (e.g., customers, orders, products)—and L denotes the set of annotated edges, termed links, which encode the relational structure among object types. Each link $l \in L$ is represented as (o_i, o_j, k, c), where $o_i, o_j \in O$ are the source and target object types, $k = (a_i, a_j)$ is the join key comprising a pair of attributes $a_i \in A_{o_i}$ and $a_j \in A_{o_j}$, and $c \in \{1:1, 1:N, N:1, N:N\}$ specifies the cardinality of the relationship. The metrics M constitute a collection of quantitative measures, each $m \in M$ defined as a tuple (o, f), where $o \in O$ is the base object type and f is an aggregated formula (e.g., `SUM`, `AVG`, or complex expressions involving window functions) that references attributes intrinsic to o (i.e., A_o) or joinable attributes accessible via paths in G defined by L.

3.2 Ontology Query Language (OQL)

The Ontology Query Language (OQL) is a declarative formalism designed to express analytical queries over the ontology $\mathcal{O} = (O, A, L, M)$. An OQL query is formally represented as a tuple $Q = (M_Q, D_Q, F_Q, R_Q)$, where:

- $M_Q \subseteq M$ is a set of metrics.
- $D_Q \subseteq A \cup (A \times \mathcal{G})$ is a set of dimensions, comprising either raw attributes or attributes paired with granularity operators $\mathcal{G} = \{\texttt{DAY}, \texttt{WEEK}, \texttt{MONTH}, \texttt{YEAR}, \ldots\}$,
- $F_Q \subseteq (A \cup M) \times \mathcal{O} \times V$ is a set of filters, where $\mathcal{O} = \{\texttt{=}, \texttt{>}, \texttt{<}, \texttt{BETWEEN}, \ldots\}$ is the set of comparison operators and V is the domain of permissible values (e.g., scalars, ranges),
- $R_Q \subseteq (A \cup M) \times \{\texttt{ASC}, \texttt{DESC}\}$ is a set of ordering criteria, specifying attributes or metrics with their sort directions.

An OQL query Q operates over the graph $G = (O, L)$ and is compiled into executable SQL through a deterministic process that ensures system-agnostic querying. The formal semantics of OQL are defined as follows:

- **Metrics (M_Q):** Each element $(m, agg) \in M_Q$, where $m = (o, f) \in M$ and $agg \in \mathcal{A}$, specifies an aggregated computation rooted at the base object o. If f is a simple attribute reference, the result is $agg(f)$; if f is a formula, it is evaluated prior to aggregation.
- **Dimensions (D_Q):** Each $d \in D_Q$ is either an attribute $a \in A$ or a tuple (a, g), where $a \in A$ and $g \in \mathcal{G}$ applies a granularity transformation (e.g., truncating a timestamp to its year, or bucketing to the millions). Dimensions define the grouping structure when $M_Q \neq \emptyset$, otherwise serving as direct selections.
- **Filters (F_Q):** Each $(t, op, v) \in F_Q$ constrains the result set, where $t \in A \cup M$ is the target, $op \in \mathcal{O}$ is the operator, and $v \in V$ is the value. Filters targeting attributes ($t \in A$) are applied pre-aggregation, while those targeting metrics ($t \in M$) are applied post-aggregation.

- **Orders (R_Q):** Each $(r, dir) \in R_Q$, where $r \in A \cup M$ and $dir \in \{\texttt{ASC}, \texttt{DESC}\}$, specifies the sort order of the result set, applicable to either attributes or aggregated metrics.

The compilation of an OQL query Q into SQL adheres to the following principles:

- For each $(m, agg) \in M_Q$, the base object o of m is designated as a fact object, and a query is rooted at its backing store.
- Dimensions, filters, and orders referencing attributes in A induce dimensional objects, connected to fact objects via paths in G defined by L. Joins are constructed as left outer joins along these paths, respecting cardinality constraints to prevent row multiplication.
- The resulting SQL includes M_Q in the `SELECT` clause (with formulaic metrics expanded), D_Q in both `SELECT` and `GROUP BY` (if $M_Q \neq \emptyset$), F_Q split between `WHERE` (for attributes) and `HAVING` (for metrics), and R_Q in `ORDER BY`.

This formal definition ensures that OQL queries are both expressive and portable, abstracting the complexity of underlying systems while leveraging the ontology's graph structure and metadata for precise, verifiable query execution.

4 Comparative Analysis of the TextQL Ontology

The TextQL Ontology, formalized as a tuple $\mathcal{O} = (O, A, L, M)$ with its associated Ontology Query Language (OQL), provides a robust framework for data abstraction and querying in enterprise environments. This section compares the ontology against two prevalent paradigms—direct SQL writing (including fine-tuning language models against historical queries) and semantic layers such as Cube and LookML—demonstrating its superior design and operational efficacy.

4.1 Comparison with Direct SQL Writing

Direct SQL writing encompasses approaches where queries are authored manually or generated automatically, often via large language models (LLMs), including those fine-tuned against historical query datasets to translate natural language into SQL. These methods face significant limitations:

- **Dialect and Complexity Challenges:** Direct SQL writing demands proficiency across all supported dialects (e.g., PostgreSQL, Snowflake, BigQuery), a challenge exacerbated for LLMs, with proprietary systems likely underrepresented in pretraining data. Complex operations—such as window functions, level-of-detail queries, or intricate joins—must be rewritten for each query, introducing non-determinism and the risk of errors (e.g., hallucinations in LLM outputs). OQL abstracts these complexities, encapsulating formulae in defined metrics and compiling queries deterministically into the appropriate dialect.
- **Lack of Abstraction:** Both manual and fine-tuned SQL operate at the physical layer, requiring knowledge of storage systems, permissions, and schema details. This lack of abstraction hinders scalability and usability for non-technical users. OQL provides a system-agnostic interface, enabling queries without such expertise.
- **Dependency and Reliability Issues:** Fine-tuning relies on the quality and representativeness of historical queries; incomplete or biased datasets degrade performance. Changes to the underlying schema—such as adding tables, modifying columns, or restructuring relationships—render fine-tuned models obsolete.

The TextQL Ontology surpasses direct SQL writing by offering a stable, abstract, and deterministic querying mechanism that mitigates the pitfalls of schema changes, dialect variability, and LLM inconsistencies.

4.2 Comparison with Semantic Layers (Cube and LookML)

Semantic layers like Cube and LookML provide abstractions over raw data, yet the TextQL Ontology surpasses them in generality, composability, and comprehensiveness:

- **Limited Scope and Curation:** Traditional semantic layers rely on curated artifacts—Cube's multidimensional views (cubes) and LookML's explores (model definitions)—which constrain their scope to preselected dimensions, measures, and relationships. Cube, for instance, requires predefined cubes optimized for OLAP, while LookML depends on manually crafted explores within Looker's ecosystem. The TextQL Ontology, defined by $\mathcal{O} = (O, A, L, M)$, is comprehensive, encompassing all object types, attributes, links, and metrics in a single, unified graph G, without requiring selective curation.
- **Support for Advanced Computations:** Unlike Cube and LookML, which often necessitate ephemeral cubes or tables to handle complex operations (e.g., level-of-detail queries, window functions), the ontology embeds such functionality directly within its metrics M. Each $m = (o, f) \in M$ can encapsulate sophisticated formulae referencing A_o or joinable attributes via L, eliminating the need for temporary structures and enhancing composability across queries.
- **Generality and Portability:** Cube's multidimensional focus and LookML's Looker-specific syntax limit their adaptability to diverse systems. The ontology's OQL, supported by a system-agnostic compiler, operates seamlessly across heterogeneous storage environments, leveraging G to define joinable paths only where supported, while treating disjoint systems as unjoinable.
- **AI Integration:** Semantic layers like Cube and LookML lack native optimization for language model legibility, whereas the ontology's business-aligned naming and metadata enhance AI-driven query generation and verification.

The TextQL Ontology's generalized graph-based formalism, comprehensive metric definitions, and composable query language distinguish it from the more restrictive, curated approaches of Cube and LookML, offering a more powerful and adaptable semantic layer.

4.3 Advantages of the TextQL Ontology

The TextQL Ontology's superiority arises from its integrated strengths:

- **Robustness to Change:** Schema evolution is managed seamlessly, unlike the invalidation faced by direct SQL and fine-tuning.
- **Comprehensive Abstraction:** By encapsulating complex logic in metrics and abstracting system specifics in OQL, it exceeds the capabilities of both direct SQL and semantic layers.
- **Deterministic Precision:** Its structured approach ensures error-free, portable queries, contrasting with the non-determinism of LLM-generated SQL.
- **AI-Optimized Design:** Metadata richness and business-aligned naming enhance AI usability, surpassing all alternatives.

In conclusion, the TextQL Ontology outperforms direct SQL writing (including fine-tuning) and semantic layers like Cube and LookML by providing a flexible, comprehensive, and AI-optimized framework, making it uniquely suited for modern enterprise data analytics.

5 Compute Layer

Enterprise data workflows require a stateful environment that extends beyond mere SQL execution and supports advanced data manipulation, visualization, and analysis using libraries such as Pandas and Polars. This is evidenced by the widespread adoption of Jupyter Notebooks for interactive data science and engineering tasks. The computational foundation of the TextQL platform is the Compute Layer, which provides agents on the platform access to such an environment while meeting two key constraints. The first is that the code is executed in a manner such that it can be treated as untrusted without creating a security risk. The second is that datasets can be efficiently loaded into the environment from external sources.

We provide this with the Sandbox Environment, an element of the Compute Layer that creates and manages Python sandboxes that provide secure, isolated environments for executing analytical code. The Sandbox Environment implements Textables, the other key element of the Compute Layer , a

high-performance abstraction for working with tabular data from diverse sources built on Apache Arrow (1). The integration of Arrow enables the Sandbox Environment to efficiently work with large datasets in a way that would not be possible with off-the-shelf solutions such as E2B (2).

5.1 Sandbox Environment

The Sandbox Environment forms a critical component of the TextQL Compute Engine, providing secure and isolated Python execution environments where user code can run safely without compromising system integrity or exposing sensitive data.

At the core of this environment is the sandbox manager, which oversees the complete lifecycle of Python sandboxes. When a new sandbox is required, the manager initializes a fresh Python environment with a carefully curated set of dependencies—including pandas, numpy, and matplotlib—enabling users to perform sophisticated data analysis without manual package installation.

To optimize performance and resource utilization, the sandbox manager employs a “Least Recently Used" (LRU) cache strategy, maintaining a pool of pre-initialized sandboxes that can be quickly assigned to incoming tasks. This approach significantly reduces latency by eliminating the need to create new sandboxes for each request. When the pool reaches capacity, the least recently used sandboxes are recycled, ensuring efficient resource management even under heavy load.

Security is paramount in the sandbox design, with multiple protection layers implemented:

- Containerization technologies isolate each sandbox from others and from the host system, preventing cross-contamination between user sessions,
- Strict resource limits on CPU usage, memory consumption, and execution time prevent runaway processes,
- Network access is restricted by default, with explicit allowlisting required for external connections, and
- Code validation occurs before execution to detect and block potentially malicious operations.

The sandbox allocation process utilizes advanced heuristics that consider resource requirements, task priority, user ownership, and system availability. This ensures tasks receive appropriate resources while maintaining system balance and preventing bottlenecks.

During task execution, the system provides comprehensive monitoring and management features, including timeout handling for long-running tasks, error recovery mechanisms with meaningful error messages, and result streaming to provide partial results as they become available. Upon task completion, the system ensures proper resource cleanup, releasing sandboxes back to the pool for reuse.

5.2 Textables

The Textables system forms an integral component of our Sandbox Environment, providing a high-performance abstraction to work with tabular data across diverse sources. Built on Apache Arrow, Textables establishes a unified interface that standardizes interactions with tabular data regardless of its origin, whether from SQL databases, CSV files or business intelligence tools.

The architecture of Textables is built around a modular connector system, where each data source has a specialized connector that handles the specifics of data extraction and transformation. These connectors implement a common interface that allows the rest of the system to interact with them uniformly regardless of the underlying data source. This approach provides flexibility and extensibility, making it easy to add support for new data sources as they become relevant. This capability is essential for supporting the advanced data manipulation, visualization, and analysis needs of enterprise workflows within the computational foundation of our platform.

Apache Arrow integration is a cornerstone of the Textables system, providing a high-performance foundation for data representation and transfer. Apache Arrow is a cross-language development platform for in-memory data that specifies a standardized language-independent columnar memory format for flat and hierarchical data. This format is particularly well-suited for analytical workloads, where operations often need to process many values from a single column.

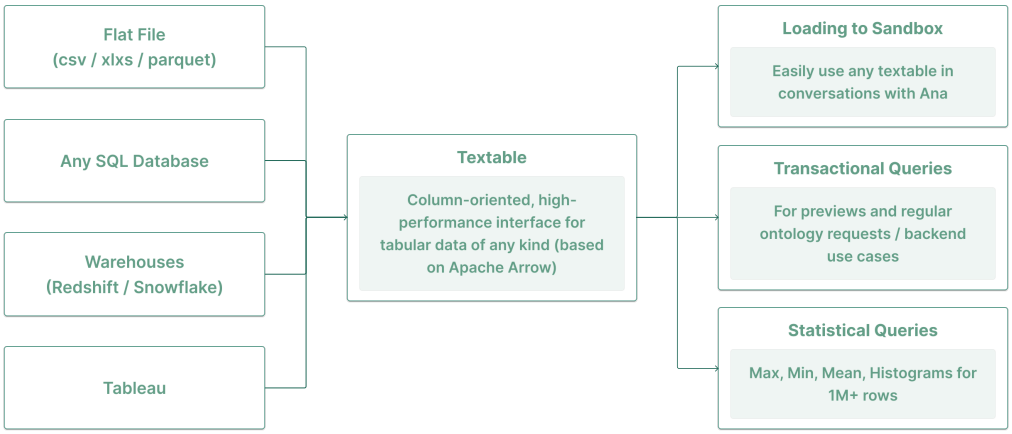

Figure 1: TextQL Textables: a high-performance interface for tabular data

TextQL leverages Apache Arrow for several critical functions within the Textables system. Textables:

- Uses Arrow's columnar format for efficient data representation, storing data in a way that maximizes cache locality and enables vectorized processing.
- Implements zero-copy data transfer mechanisms, allowing data to move between different components without the overhead of serialization and deserialization.
- Ensures cross-language compatibility, enabling seamless data sharing between the Go-based compute engine and Python-based analysis code.
- Utilizes Arrow's support for vectorized processing, enabling high-performance operations on columnar data that can utilize modern CPU features like SIMD instructions.

6 Agent Layer

6.1 Dakota State Framework

The Dakota State Framework serves as the foundational architecture within the Ana agent, employing a dynamic state-machine paradigm to manage and control agent behaviors and tool interactions at runtime. Dakota enables Ana to dynamically switch contexts, adjust prompts, and integrate external information, providing a flexible yet structured approach to handling various user queries and analysis tasks. This dynamic flexibility is essential in ensuring that Ana can adapt seamlessly to evolving user demands, complex business contexts, and diverse analytical scenarios.

Dakota is organized into distinct modes, each representing a specific operational state of the Ana agent:

Core Modes:

- **Idle.** Ana anticipates user input. This mode ensures that Ana is responsive and immediately ready to engage.
- **Help.** Ana requires clarification or additional information from the user to better understand or complete a query, ensuring precise and helpful responses.
- **Meta.** Ana provides information regarding the TextQL system and its functionalities to the user.
- **Query.** Ana actively searches for relevant data resources such as metrics, tables, and dashboards that best match the user's question.
- **Plan.** Ana formulates a structured plan or approach to address complex analytical questions, defining clear steps and methodologies for analysis.

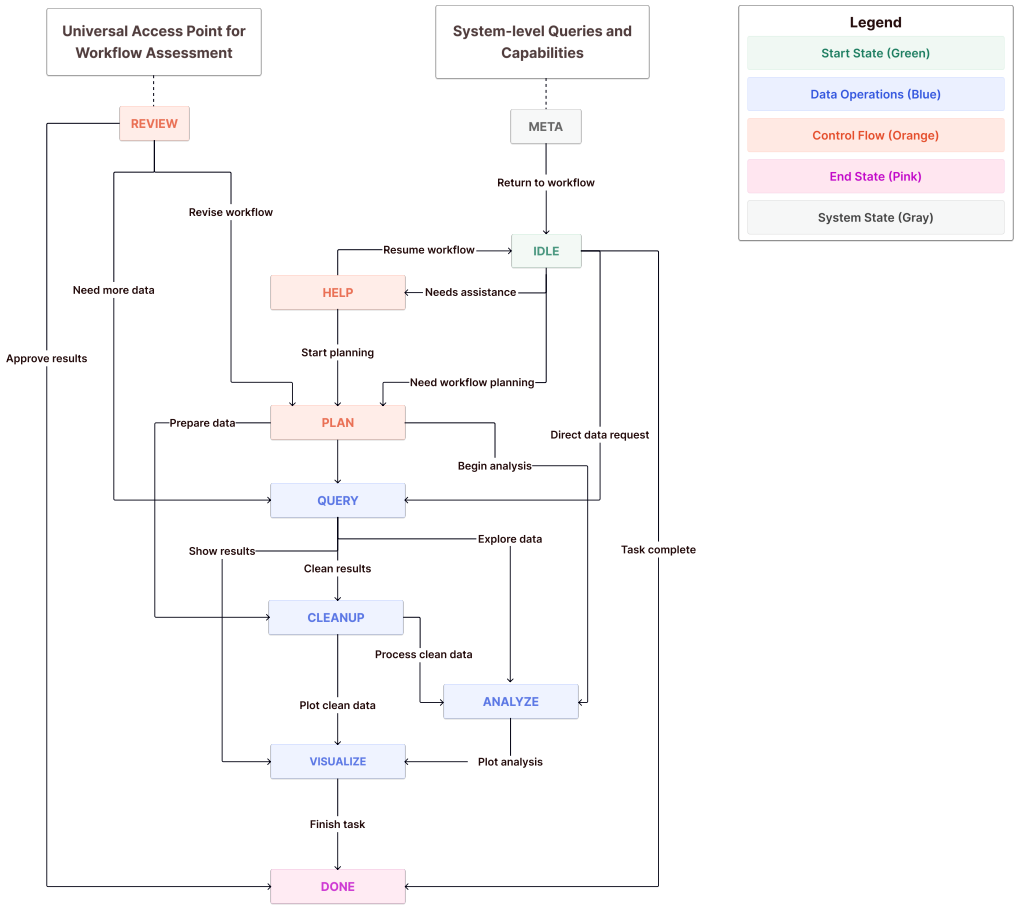

Figure 2: Dakota State Framework: Modes and Transitions.

- **Done.** Ana presents the completed analysis, answers user queries, and provides any resultant visualizations or summarized insights.

Analysis Modes:

- **Cleanup.** Ana prepares, cleans, and formats data using Python scripts, ensuring accurate and reliable subsequent analysis by transforming datasets into appropriate structures.
- **Analyze.** Ana executes Python-based analytical tasks to extract meaningful patterns, statistics, and insights from the cleaned datasets.
- **Visualize.** Ana transforms analysis outcomes into informative visual representations such as charts and graphs, significantly enhancing interpretability and insight communication.
- **Review.** Ana evaluates the retrieved datasets to ensure they contain sufficient and relevant information necessary for accurately addressing user queries.

Dakota defines clear transition pathways between these modes, ensuring logical and structured operational flows:

1. **Question Flow:** Idle → Query → Plan → Analyze → Done
2. **Data Preparation Flow:** Query → Review → Cleanup → Analyze
3. **Visualization Flow:** Analyze → Visualize → Done
4. **Clarification Flow:** Any Mode → Help → Return to Previous Mode

Figure 2 explicitly outlines these transitions and mode interactions. Our robust mode-based structure ensures that Ana reliably addresses a wide array of analytical challenges, offering enterprises precise, context-sensitive insights tailored to complex, real-world data scenarios.

6.2 Execution Graph

The Execution Graph underpins the operational efficiency of the Ana agent, providing a structured, cell-based Directed Acyclic Graph (DAG) architecture. Each interaction with Ana's chat interface is encapsulated in a "cell," which independently invokes specialized analytical tools and functions provided. These cells include the following:

- **Python Cell:** Executes Python scripts directly within Ana, enabling sophisticated data manipulations, transformations, and advanced analytical computations within a secure, sandboxed environment.
- **Object Explorer Cell:** Allows users to browse and interactively explore available database tables and their schemas directly, significantly improving the ease and precision of data selection processes.
- **Metric Explorer Cell:** Provides an interactive way to discover and utilize predefined metrics, enhancing consistency, accuracy, and speed when performing analyses that involve established business KPIs and metrics.
- **Search Cell:** Facilitates the rapid and efficient search of data elements, including tables, dashboards, or specific metrics, directly from within the notebook interface, reducing friction and accelerating analytical workflow.
- **Web Search Cell:** Enables the integration of real-time web searches directly into analysis workflows, expanding the scope and richness of analytical insights by incorporating up-to-date external information.
- **Streamlit Cell:** Supports the creation and embedding of interactive Streamlit applications directly within the interface, allowing dynamic user interaction and facilitating rich, customized visualization and reporting experiences.

This DAG-based cell structure promotes clarity, transparency, and reproducibility, explicitly showing the sequence and dependencies of analytical steps. It enables users to track each analytical stage, ensuring robust validation, auditing, and adjustment capabilities. Moreover, the modular nature of the cells ensures each analytical function is self-contained and easily adaptable, facilitating iterative refinement without disrupting overall analytical integrity.

By integrating these distinct cell functionalities into a coherent execution graph, we significantly enhance the analytical agility and reliability of Ana. Enterprises gain the flexibility to address complex, iterative analytical tasks swiftly, fostering informed decision-making and strategic data utilization in dynamic business contexts.

7 Evaluation

7.1 Conglomerate-Benchmark-V1-Lite

CONGLOMERATE-BENCHMARK-V1-LITE (3) is an algorithmically generated benchmark designed to test AI and conversational data analysis tools. It features an ontology with 109 interconnected tables representing a diversified conglomerate business with over 10 subdivisions, including healthcare, medical devices, chemical manufacturing, logistics, CRM, and database tools.

The benchmark includes 18 question-answer pairs covering a broad range of capabilities such as text-to-SQL, Python scripting, data prediction, plotting, and exporting functionalities. The primary challenge for tools evaluated by this benchmark is navigating complex schema, executing multi-table joins, and effectively managing searches across extensive dimensions and metrics.

Five sample question answer pairs are provided in Table 1.

Question	Answer
How many employees did we hire in 2021?	16
Average duration for CRM user activities for teams in North America on mobile by team type	IT: 75 sec Procurement: 74 sec Operations: 73 sec Finance: 72 sec
Do we make more money from snacks or gambling?	Gambling Revenue: $19.6M Snack Sales Revenue: $8.0M
In which months of 2024 did our web channel outperform all other channels in furniture sales by order quantity, and in what percentage of those months did we successfully launch 3 or more ERP consulting projects that ended up completed?	33%: Jan, Mar, Apr, Sept
For each property class, what's the total net income per square foot for mixed use spaces	Class A: $121.7/sq ft Class B: $46.74/sq ft Class C: $64.0/sq ft

Table 1: CONGLOMERATE-BENCHMARK-V1-LITE: Sample Question Answer Pairs

7.2 Evaluation

We evaluate TEXTQL-ANA-ENTERPRISE, TEXTQL-ANA-SMALL, on CONGLOMERATE-BENCHMARK-V1-LITE, along with competitors solutions such as DEEPNOTE-AI, AMAZON-Q, DATABRICKS-GENIE, HEX-AI, and SNOWFLAKE-CORTEX-ANALYST.

Our evaluation methodology is tailored to assess each tool's ability to provide accurate and usable answers from their primary interfaces. We considered the tools' native capabilities and operational contexts, scoring responses as correct if they produced precise answers without the need for repeated interactions. In partiuclar,

- AMAZON-Q was awarded partial credit where data retrieval led to correct conclusions despite incomplete query resolution.
- HEX-AI and DEEPNOTE-AI were evaluated on the accuracy and practicality of the outputs produced by their notebook interfaces.
- SNOWFLAKE-CORTEX-ANALYST faced limitations in dataset handling, failing to complete the benchmark queries due to the restrictive size and complexity thresholds.

Interactive tools were allowed follow-up interactions for more detailed context, but TEXTQL-ANA-ENTERPRISE and TEXTQL-ANA-SMALL were scored based solely on their initial responses.

The performance of the tools is summarized in Figure 3, which visually represents the percentage scores.

References

[1] A. Arrow, "Apache arrow," 2025. Version X.Y.Z, accessed March 19, 2025.

[2] e2b, "e2b - Autonomous agents that control your apps," 2025. Accessed: 19 March 2025.

[3] TextQLLabs, "Conglomerate-benchmark-v1-lite." `https://github.com/TextQLLabs/conglomerate-benchmark`, n.d. Accessed: March 19, 2025.

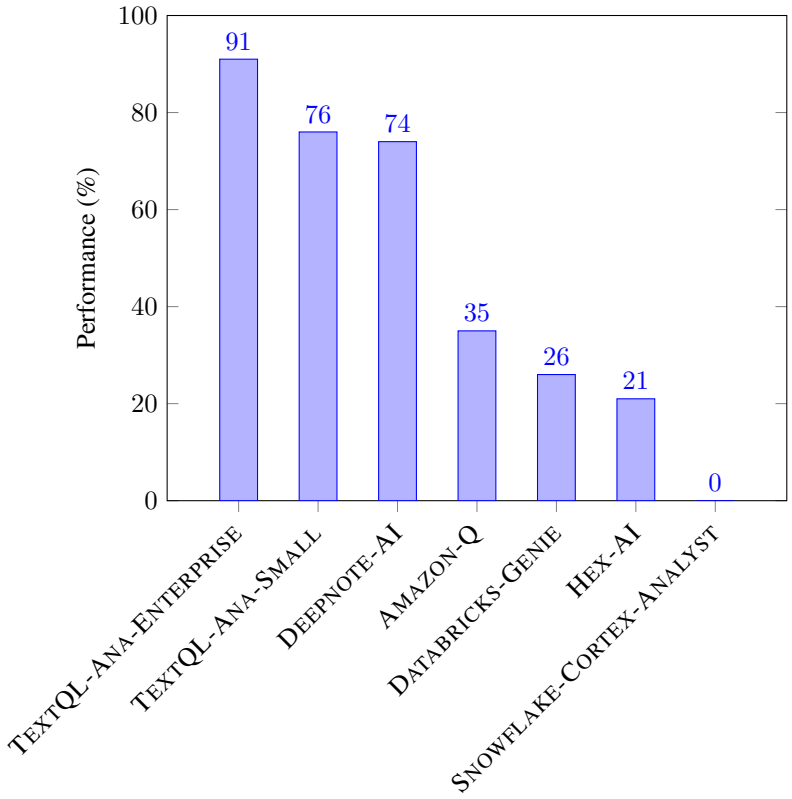

Figure 3: CONGLOMERATE-BENCHMARK-V1-LITE: Benchmark results.