The Semantic Layer: Architectural Strengths of Linguistic Schemas over Graph Ontologies in Text-to-SQL Systems

Executive Summary

The enterprise data ecosystem is at an inflection point. The long-promised democratisation of dataenabling any user, irrespective of technical expertise, to query and analyse complex datasets, depends fundamentally on the effectiveness of Text-to-SQL systems. Historically, this translation between human intent and database execution has relied on constructs such as Entity-Relationship (ER) diagrams, visual ontologies, and static graph representations. These abstractions were explicitly designed for human cognition, leveraging visual perception, spatial reasoning, and diagrammatic intuition.

Large Language Models (LLMs) have fundamentally altered this landscape. LLMs are not visual reasoners; they are probabilistic linguistic systems optimised to process, compress, and reason over language. As a result, schema representations engineered for human comprehension often become suboptimal often counter productive when applied to language-first models. Compounding this mismatch, traditional ontology creation has historically demanded extensive manual effort, making it slow, brittle, and difficult to maintain at enterprise scale. In contrast, modern LLM-driven systems can now generate and evolve semantic layers automatically, dramatically reducing time-to-insight while improving analytical accuracy.

This whitepaper argues for a necessary and decisive shift away from graph-centric schema paradigms toward a linguistically grounded Semantic Layer. Drawing on empirical insights from state-of-the-art research and the techniques employed by top performers on benchmarks such as Spider and BIRD, we demonstrate that Text-to-SQL accuracy improves markedly when database structures are expressed as rich natural-language descriptions, logical narratives, and linearised pseudo-schemas representations aligned with how LLMs reason.

We examine the structural limitations of ER diagrams within Transformer-based attention architectures, outline the architectural advantages of Terno AI's Semantic Metastore and SQLShield components, and emphasise the importance of agentic workflows that replicate the iterative reasoning patterns of experienced data scientists. This paper serves as a practical and conceptual guide for enterprise architects, data scientists, and technology leaders seeking to build

high-accuracy, secure, and context-aware natural language interfaces for data.

---

1. The Cognitive Mismatch: Why Traditional Data Modelling Fails LLMs

The foundational error in many contemporary Text-to-SQL implementations is the assumption that tools designed for human understanding are transferable to artificial intelligence. This anthropomorphic bias has led to the persistence of Entity-Relationship (ER) diagrams and visual knowledge graphs as the primary means of supplying context to LLMs. To understand why the "Terno Way", a semantic text-first approach, is more effective in this context, we must first deconstruct the cognitive mismatch between human engineers and large language models.

1.1 The Visual-Spatial vs. Sequential-Linguistic Divide

Human database administrators (DBAs) and data engineers rely heavily on visual abstractions. An ER diagram uses spatial positioning, connecting lines (crow's foot notation), and distinct shapes (rectangles for entities, diamonds for relationships) to convey the structure of a database.[1] When a human looks at an ER diagram, they engage in parallel processing; they can instantly perceive clusters of related tables, trace foreign key paths visually, and ignore irrelevant sections of the schema. The "interface" is the image, and the "processor" is the visual cortex.

LLMs, conversely, operate on a fundamentally different substrate. They process information sequentially as a stream of tokens. They do not "see" the relationships in an ER diagram; they "read" the serialized representation of that diagram. When a graph structure is converted into a format that an LLM can ingest, typically an adjacency matrix, a JSON object, or a list of nodes and edges, the spatial intuition is stripped away, leaving behind a dense, high-entropy string of identifiers.[2]

Research into the efficacy of schema representations has shown that "linearized schemas", where tables and columns are described in natural language sequences, consistently outperform complex graph encodings when used with pre-trained language models.[2] The LLM's pre-training on vast corpora of text (books, articles, code) has optimized its attention heads to track semantic relationships across linguistic sequences, not to reconstruct 2D graph topologies from serialized text. Therefore, forcing an LLM to parse a graph-based ontology is akin to describing a painting to a computer by listing the coordinates of every brushstroke; the "meaning" is lost in the translation. The semantic approach, which describes the painting's subject and mood (e.g., "A sunset over a harbor"), aligns with the model's native capability.[4]

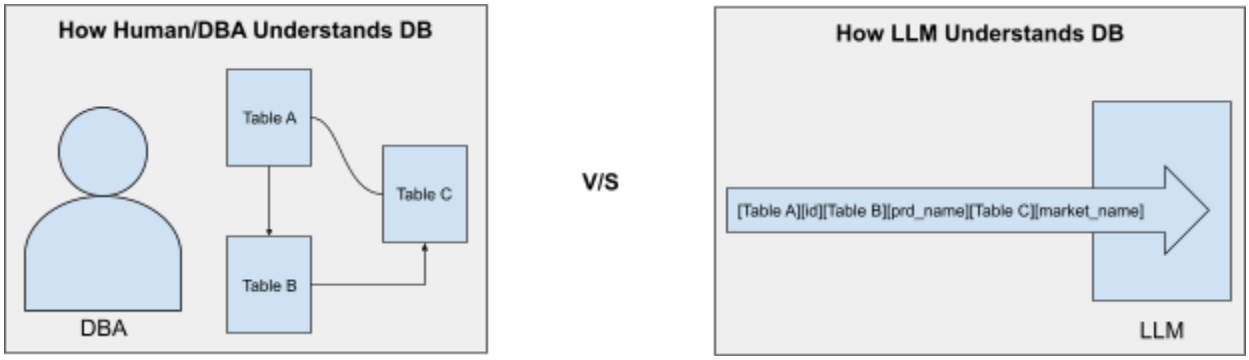

Figure 1: Visual-Spatial vs. Sequential-Linguistic

1.2 The Failure of Rigid Ontologies in Natural Language Reasoning

Traditional ontologies are constructed on rigid hierarchies and strict taxonomies. They function well in deterministic systems where terms have a one-to-one mapping. However, natural language is inherently ambiguous, polysemous, and context-dependent. A user might ask for "revenue," "sales," "turnover," or "gross income." In a strict graph ontology, unless these edges are explicitly defined, the traversal fails.

LLMs, however, excel at soft probabilistic matching. They understand that "turnover" in the context of a retail database likely refers to "sales," whereas in an HR database, it refers to "employee attrition." This inference is not driven by hard-coded graph edges but by the semantic proximity of the words in the model's high-dimensional embedding space.[5]

The "Terno Way" leverages this by utilizing a **Semantic Layer** that prioritizes "verbose descriptions" over rigid structural definitions.[1] By describing a column not just as sales_amt but as "The total monetary value of transactions completed within the fiscal period, excluding returns," the system provides the LLM with the "semantic anchors" necessary to resolve ambiguity. This approach mirrors the findings in dimensional modelling, where descriptive attributes are recognized as the primary target for business intelligence queries.[7] The graph approach requires the user to know the specific vocabulary of the ontology; the semantic approach meets the user in their own linguistic territory.

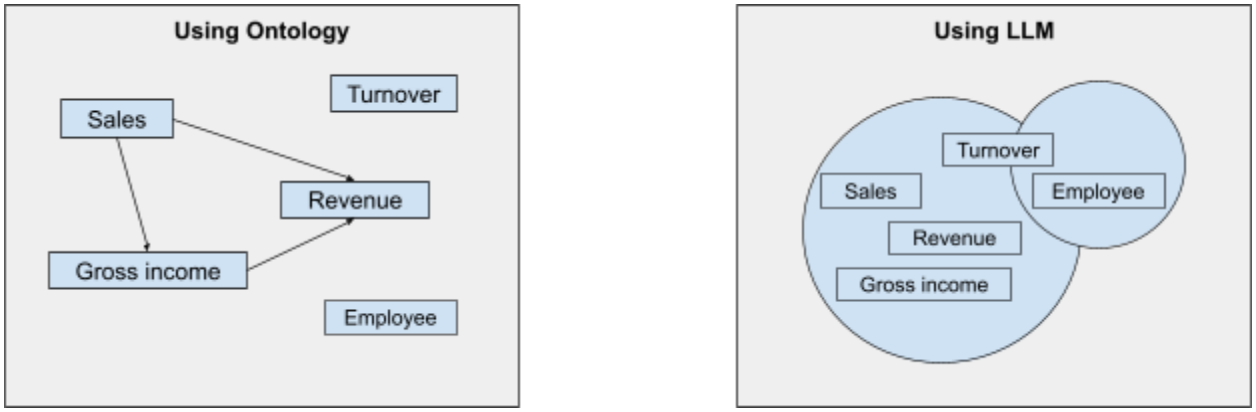

Figure 2: Rigid Ontology vs. Semantic Matching

1.3 The "Tokenization of Structure" Problem

Another critical inefficiency of traditional mechanisms is the "token tax." Representing a complex database schema as a full knowledge graph or ontology often requires a massive number of tokens to define every node, edge, and constraint. Given the finite context windows of LLMs, this "noise" can crowd out the relevant signal.

Papers analyzing the impact of schema representation on performance have noted that "concise representation... through sampled graph language corpus" often beats raw graph dumps.[4] However, even more effective is the **Pseudo-Schema** approach (discussed in depth in Section 3), which abstracts the physical schema into a simplified, semantic view.[8] Terno's architecture minimizes the cognitive load on the LLM by effectively "compressing" the structural complexity into semantic density. Instead of spending tokens defining the cardinality of a relationship (which the LLM might struggle to strictly enforce via text generation), the system provides a logical narrative: "Customers place Orders." This is concise, semantically rich, and directly translatable to SQL JOIN logic by the LLM.[8]

1.4 Second-Order Insight: The "Hallucination" of Connectivity

A subtle but profound risk of using graph representations with LLMs is the induction of "connectivity hallucinations." When an LLM is presented with a list of graph edges (e.g., Table A -> Table B, Table B -> Table C), it may infer a transitive relationship (Table A -> Table C) that does not semantically exist or is logically invalid for the specific query context. Because the model is trying to complete the pattern, it might generate a SQL query that joins A to C directly, bypassing the necessary intermediate logic of B.

In contrast, a semantic description that explains the *nature* of the relationship (e.g., "Table B acts as a junction table recording the date of the transaction between A and C") forces the model to

attend to the *process* rather than just the *connectivity*. This distinction is vital for accuracy. The graph provides the *path*; the semantic layer provides the *reason* for the path. For "great accuracy," the reasoning is indispensable.

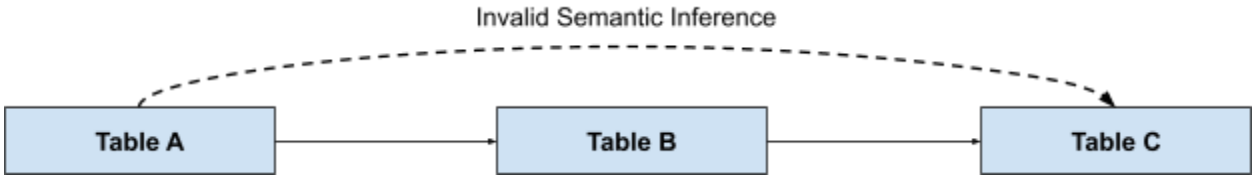

Figure 3: Connectivity Hallucinations

---

2. The Mechanics of Terno: Architecting the Semantic Layer

Having established the theoretical limitations of traditional graph-based approaches, we turn to the specific architecture that overcomes them. Terno AI represents an implementation of the "Semantic First" philosophy. Its architecture is not merely a wrapper around an LLM but a comprehensive ecosystem designed to manage context, enforce security, and simulate the reasoning capabilities of a human data scientist.

2.1 The Metastore: The Brain of the System

The heart of Terno's accuracy is the **Metastore**.[6] To the uninitiated, this might sound like a standard database catalog (such as the information_schema in SQL). However, the distinction is profound.

- **Traditional Metadata:** Stores technical details: column_name, data_type, and is_nullable.
- **Terno Metastore:** Stores **Business Knowledge**.

The Metastore is designed to capture the "why" and "what" of the data, not just the "how." It retains knowledge provided by experts or learned automatically from the data itself.[6]

2.1.1 Mechanisms of Knowledge Retention

The Metastore functions as a dynamic knowledge graph (in the conceptual sense, not the rigid structural sense) that maps physical data assets to business concepts. Rather than encoding relationships purely through static schema constraints, it maintains a semantically enriched representation of the data environment that can be continuously referenced by downstream reasoning systems.

1. **Semantic Aliasing:** It maps cryptic or system-generated physical identifiers (e.g., t_sub_01) to logical business entities (e.g., Subscription_History). This abstraction layer decouples business semantics from physical schema volatility, enabling large language models to reason over concepts rather than implementation artefacts.
2. **Logic Encapsulation:** It stores the logic for computed metrics. For example, "Gross Margin" is not a column; it is a calculation (Revenue - COGS) / Revenue. The Metastore stores this formula. When a user asks for "Gross Margin," the system retrieves this definition, preventing the LLM from hallucinating an incorrect formula.6
3. **Automated Insight Generation:** The Metastore proactively analyzes the data to understand distributions, frequent values, and seasonal trends. This "statistical awareness" is fed to the LLM, allowing it to generate queries that are data-aware (e.g., knowing that the status column contains 'Active' and 'Inactive' rather than '1' and '0'). This materially improves query correctness and reduces trial-and-error execution.
4. **Foreign Key and Relationship Inference:** Beyond explicit constraints, the Metastore infers relational links by combining automated insights with schema-level metadata embedded in column descriptions (e.g., prd_id joins to dim_product.id). By analyzing value overlap and cardinality patterns, it establishes semantically meaningful join paths, enabling accurate relationship reasoning even when foreign keys are missing, inconsistent, or undocumented.
5. **Schema Curation, Renaming, and Visibility Control:** The Metastore enables explicit control over table and column visibility to reduce schema noise common in long-lived databases. Ambiguous, duplicate, deprecated, or system-generated artefacts are hidden or aliased with clear semantic names. This curation minimizes cognitive and token-level overhead for the LLM while constraining query generation to valid structures, improving SQL accuracy and reducing inference costs.

2.2 SQLShield and the Pseudo-Schema: Bridging Accuracy and Security

One of the most innovative components of Terno's architecture is **SQLShield**. While primarily marketed as a security feature to prevent SQL injection and enforce access control, its role in **accuracy** is equally critical.[6]

2.2.1 The Pseudo-Schema Concept

Standard Text-to-SQL systems feed the raw database schema to the LLM. If the schema is

messy, the LLM's performance degrades. SQLShield intervenes by generating a **Pseudo-Schema**.[8]

- **Definition:** A Pseudo-Schema is a virtual, idealized version of the database schema. It consists of highly descriptive table and column names (pub_name) that are optimized for natural language understanding.
- **The Translation Process:**
  1. **Generation:** The system generates a clean Pseudo-Schema based on the Metastore's business definitions.
  2. **Prompting:** The LLM prompts against this Pseudo-Schema. Because the names are intuitive (e.g., Customer_Purchases instead of tbl_cp_2024), the LLM generates the SQL with high confidence and accuracy.
  3. **Transpilation:** SQLShield intercepts the generated SQL (which is valid only against the Pseudo-Schema) and translates it into the complex, optimized SQL required by the physical database.[9]

Figure 4: Working of SQLShied

2.2.2 Accuracy through Abstraction

This mechanism directly addresses the prompt's requirement for "better names, descriptions and foreign key mapping." By decoupling the schema the LLM sees from the schema the database uses, Terno allows for "Virtual Refactoring." We can "rename" tables for the sake of the AI without breaking the underlying application. Research validates this: papers like **Gen-SQL** explicitly show that bridging natural language questions and database schemas with pseudo-schemas significantly enhances efficiency and accuracy.[8] The Pseudo-Schema acts as a "Cognitive Buffer," smoothing out the irregularities of the physical data model.

2.3 The Agentic Workflow: The "AI Data Scientist"

The prompt asks for "how they will do it." The answer lies in **Agentic Workflows**. Terno does not treat Text-to-SQL as a single-shot translation task. It treats it as a data science project.[11]

2.3.1 Decomposition and Reasoning

Complex queries (e.g., "Show me the month-over-month growth of churned users compared to

the industry average") cannot be solved in a single inference pass. Terno uses an agentic architecture to:

1. **Decompose:** Break the query into sub-tasks (Calculate Churn -> Calculate Growth -> Fetch Industry Average -> Compare).[13]
2. **Execute & Verify:** Each step is executed. If a step fails (e.g., a divide-by-zero error), the agent analyzes the error message and self-corrects.[15]
3. **Memory & artefacts:** Intermediate results are stored in the **Artifact Store**.[6] This "Memory Layer" is crucial. It allows the system to hold the result of "Step 1" while calculating "Step 2," enabling multi-step reasoning that a stateless graph traversal could never achieve.

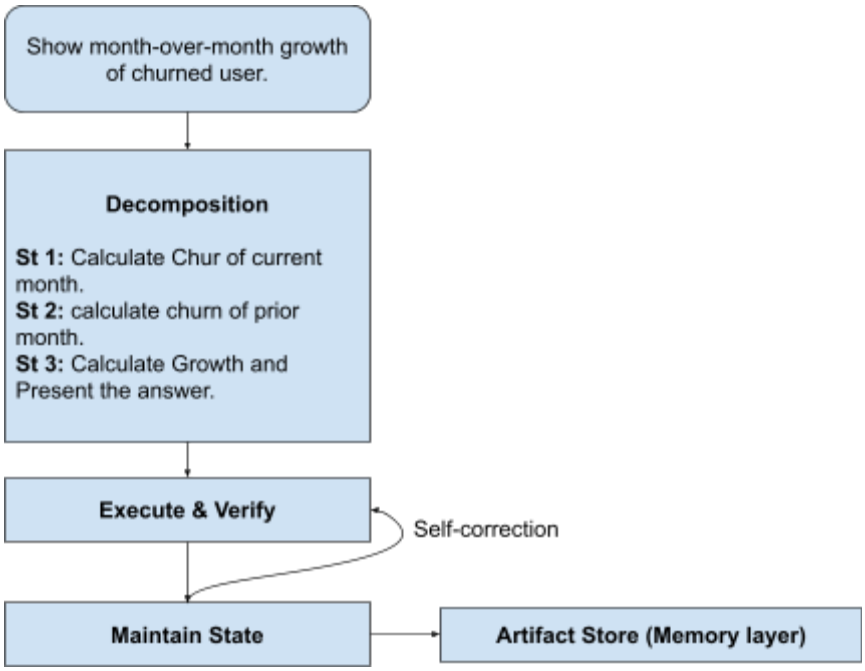

Figure 5: How Agentic-Workflow works

2.4 Terno Architecture vs. Traditional Method

Component	Traditional Approach (Graph/ERD)	Terno Approach (Semantic/Agentic)	Impact on Accuracy
Schema Representation	Nodes, Edges, Cardinality Constraints	Verbose Descriptions, Business Logic,	**High:** LLMs understand descriptions better

		Pseudo-Schemas	than topology.[2]
Schema Linking	Exact Keyword Matching (Graph Traversal)	Semantic Similarity Search (Metastore)	**High:** Resolves synonyms and ambiguity (e.g., Revenue = Sales).[6]
Complex Logic	Hard-coded Ontology Rules	Agentic Decomposition & Self-Correction	**Very High:** Handles multi-step reasoning and "dirty" data.[14]
Context Management	Stateless Query Generation	Artifact Store & Memory Layer	**High:** Enables follow-up questions and complex workflows.[6]
Security/Abstraction	Direct Schema Exposure	SQLShield (Pseudo-Schema Translation)	**High:** Decouples AI from physical schema complexity.[9]

Table 1: The Terno Architecture vs. Traditional Method

---

3. Evidence from the Frontlines: Analyzing the Spider and BIRD Benchmarks

To validate the advantages of the Terno approach, we must examine the empirical evidence provided by the premier Text-to-SQL benchmarks: **Spider** (Standard) and **BIRD** (Enterprise/Real-World). The evolution of top-performing models on these leaderboards mirrors the industry's shift from structural to semantic methodologies.

3.1 Spider Benchmark: The Victory of Description over Structure

The Spider benchmark evaluates cross-domain Text-to-SQL performance. An analysis of the leaderboard reveals that the "Toppers" utilize mechanisms that align perfectly with the semantic/agentic approach.

3.1.1 DAIL-SQL: The Power of Formatting

DAIL-SQL, a leading method on Spider, does not rely on complex graph neural networks. Instead, it focuses on "formatting schema as code/text" and efficient example selection.[16]

- **Mechanism:** It optimizes the prompt by selecting the most relevant few-shot examples based on semantic similarity and presenting the schema in a simplified, code-like structure (linearized schema).
- **Insight:** The success of DAIL-SQL proves that the *quality of the text description* passed to the LLM is more important than the underlying graph structure. It emphasizes **Schema Linking,** identifying the mapping between question words and schema wordsas a linguistic task, not a topological one.[18]

3.1.2 DIN-SQL: Decomposition as the Key

DIN-SQL (Decomposed In-Context Learning) represents the shift towards agentic reasoning.[14]

- **Mechanism:** It breaks the problem into four sub-modules: Schema Linking, Classification, Decomposition, and Self-Correction.
- **Insight:** This mirrors Terno's agentic workflow. By decomposing a complex "hard" question into a sequence of simpler steps, DIN-SQL achieves state-of-the-art performance. It explicitly rejects the idea that a single "graph traversal" can solve complex intent. It builds a "Logical Graph" through the *process* of decomposition, rather than relying on a static ER diagram.[14]

3.2 BIRD Benchmark: The Reality of Dirty Data

The BIRD (Big Instruction to Real Database) benchmark is widely considered the true test of enterprise readiness because it involves "dirty" data and requires external knowledge.[20]

3.2.1 The Necessity of External Knowledge

On BIRD, models that only look at the schema (even a graph schema) fail. Why? Because the database might contain values like C_Type = 1, where 1 means "Corporate." A graph cannot tell you this.

- **Top Performers:** The top models on BIRD (like **Arctic-Text2SQL** and **IBM Granite**) explicitly use "external knowledge" files, descriptions of values and business rules.[22]
- **Terno Alignment:** This validates Terno’s **Metastore** strategy. The Metastore is the repository for this "external knowledge." It allows the system to say, "When the user asks for 'Corporate', look for C_Type = 1." Without this semantic injection, accuracy on real-world data is impossible.[6]

3.2.2 Gen-SQL and the Pseudo-Schema Validation

The **Gen-SQL** paper [8] provides direct academic validation for Terno's **SQLShield** approach.

- **Finding:** Gen-SQL demonstrates that bridging the natural language question and the database schema with a "Pseudo-Schema" significantly enhances efficiency.
- **Reasoning:** The Pseudo-Schema reduces the search space and aligns the vocabulary of the problem (the question) with the vocabulary of the solution (the schema). This "Semantic Alignment" is the primary driver of accuracy in large-scale systems.[24]

Benchmark	Top Technique	Corresponding Terno Component
Spider	**DAIL-SQL:** Textual Schema Formatting. **86.6% execution accuracy** [30]	**Metastore:** Linearised, descriptive schema presentation.
Spider	**DIN-SQL:** Task Decomposition raising execution accuracy to **85.3% (prev ~79.9%)** [31]	**Agentic Workflow:** Step-by-step reasoning and self-correction.
BIRD	**External Knowledge Injection** report **~77–78% execution accuracy** [32]	**Metastore:** Storage of business logic and value mappings.
General	**Pseudo-Schema Generation (Gen-SQL)** [24]	**SQLShield:** Dynamic generation of English-friendly schemas.

Table 2: Benchmark Techniques vs. Terno Components

---

4. The Three Pillars of Accuracy: A Technical Synthesis

To "write a whitepaper outlining how to achieve great accuracy," we must distill the techniques used by Terno and the benchmark toppers into three actionable pillars.

4.1 Pillar 1: Semantic Enrichment ("Better Names and Descriptions")

The prompt correctly identifies that "coming up with better names and descriptions" is key.

- **The Mechanism:** We replace the physical schema S_phys with a logical schema S_log.
- **Implementation:** S_log contains Table: Orders (Description: Records of customer purchases, including date and total amount).
- **Impact:** This maximizes **Schema Linking** accuracy. Papers show that "verbose descriptions" allow the LLM to cluster embeddings more effectively, reducing variance and increasing the probability of selecting the correct table.[5]
- **Terno Integration:** Terno's Metastore automates this enrichment, analyzing the data to generate these descriptions if they are missing.[6]

4.2 Pillar 2: The Logical Graph ("Building Logical Relationships")

While we reject the *visual* ER diagram, we embrace the *logical* graph constructed via text.

- **The Mechanism:** Explicitly stating relationships in the prompt: "The Orders table is linked to the Users table via user_id. This relationship represents 'Who placed the order'."
- **Implementation:** This is not a graph data structure; it is a narrative.
- **Impact:** This prevents the "Connectivity Hallucinations" discussed in Section 1.4. It gives the LLM the "Join Logic" in natural language, which it can easily translate to SQL JOIN syntax.
- **Evidence: MAC-SQL** uses a "selector agent" to prune the schema and a "refiner agent" to verify these logical links.[28]

4.3 Pillar 3: The Memory Layer ("Overview and Context")

The prompt asks for "an overview of the whole DB along with the memory layer."

- **The Mechanism:** The **Artifact Store** and **Global Context**.
- **Implementation:**
  - **Global Context:** A high-level summary of the database domain (e.g., "Retail E-Commerce") is always present in the system prompt.
  - **Memory Layer:** The system maintains a "Conversation State." If the user filters a result set, the system doesn't just rewrite the SQL; it understands the *data* in the previous result. Terno's Artifact Store saves these intermediate states, allowing for "Data-Aware" follow-up queries.[6]
  - **Impact:** This solves the "Lost in Translation" problem of multi-turn conversations, a key differentiator between a "Chatbot" and an "AI Data Scientist."

---

5. Automatic Generation of Semantic Layer

One of the primary barriers to adopting semantic-first architectures at enterprise scale is the

perceived cost and effort of manual curation. Traditional semantic layers and ontologies require months of workshops, schema reviews, and domain expert involvement. This paper asserts that such manual processes are no longer necessary. Modern LLMs, when combined with controlled prompting, statistical inspection, and verification loops, can **automatically construct and continuously refine a high-fidelity Semantic Layer** directly from raw databases.

The Terno architecture treats semantic generation as a **first-class system capability**, not a one-time setup task. The Semantic Layer is generated, validated, and evolved through a structured, multi-stage pipeline.

5.1 Stage 1: Automated Table Relevance Detection and Pruning

Enterprise databases contain numerous tables that are irrelevant for analytical querying, including empty tables, intermediate ETL artefacts, copies, aggregates, system-generated logs, and deprecated structures. Exposing these tables to an LLM introduces noise and degrades accuracy.

Objective:
Identify and hide tables that do not contain meaningful or query-relevant data.

Mechanism (LLM Instructions):

- *"Hide all tables that don't have the data."*
- *"Identify tables that appear to be system-generated, staging, temporary, sorted, aggregated, or duplicates of other tables."*
- *"List tables that should be excluded from natural language querying and provide a brief reason."*

Outcome:
Only semantically meaningful tables are retained in the Semantic Layer. All hidden tables remain physically intact but are excluded from downstream reasoning and prompt context.

5.2 Stage 2: Automatic Generation of Table and Column Descriptions and Canonical Names

Raw database schemas are rarely designed for natural language understanding. Table and column names are often abbreviated, cryptic, or system-oriented. This stage converts physical identifiers into business-aligned semantic representations.

Objective:
Generate clear descriptions and canonical, human-readable names for tables and columns.

Mechanism (LLM Instructions):

- *"Go through 100 rows of each table and come up with a clear business description of*

what this table represents."

- *"Suggest a better, business-logical name for each table."*
- *"For each column, generate a concise semantic description based on the observed data."*
- *"Recommend improved column names that reflect their actual meaning in business terms."*

Outcome:
Each table and column is associated with:

- A natural-language description
- A canonical semantic name
- Business context suitable for direct LLM reasoning

These representations form the primary interface exposed to the LLM during query generation.

5.3 Stage 3: Logical Relationship Inference

In many enterprise databases, foreign key constraints are missing, incomplete, or unreliable. Instead of relying solely on physical constraints, Terno infers relationships at the semantic level.

Objective:
Identify and describe the logical relationships between tables.

Mechanism (LLM Instructions):

- *"Figure out the logical relationships between tables."*
- *"Identify which columns can be used to join tables and explain the relationship in business terms."*
- *"Describe how entities in one table relate to entities in another."*

Outcome:
Relationships are stored as **natural language join narratives**, not rigid graph edges. These descriptions guide SQL JOIN construction during query generation and reduce hallucinated or invalid joins.

5.4 Stage 4: Ambiguity Detection and Schema Curation

Ambiguous table or column names (e.g., status, type, code) and overlapping entities significantly increase the likelihood of incorrect queries.

Objective:
Detect ambiguity and curate the exposed schema.

Mechanism (LLM Instructions):

- *"Go through all the tables and list tables whose names or descriptions are ambiguous."*

- *"Identify columns whose meaning is unclear or overloaded."*
- *"For each ambiguous table or column, recommend whether it should be renamed, hidden, or have its description updated."*

Outcome:
The Semantic Layer is curated to ensure:

- Each exposed entity has a single, unambiguous meaning
- Redundant or misleading structures are hidden
- Renamed entities better match the user's language and intent

5.5 Stage 5: Continuous Validation and Self-Healing

Semantic layers degrade as schemas evolve. Terno treats semantics as **living knowledge**.

Mechanisms:

- Monitor query failures and correction loops.
- Detect schema drift and data distribution shifts.
- Re-run semantic generation selectively when anomalies appear.

Feedback Loop:

- Failed SQL → error analysis → semantic update → retry
- User corrections → Metastore updates → future prevention

This ensures the Semantic Layer improves over time without manual intervention.

5.6 Summary: Why Automation Matters

Automatic Semantic Layer generation:

- Eliminates months of manual ontology engineering
- Aligns schema representation with LLM reasoning
- Enables scalable External Knowledge Injection
- Maintains enterprise safety and reversibility

By transforming raw schemas into **living semantic narratives**, Terno operationalises what benchmark winners implicitly rely on.

---

6. Conclusion: The Future is Semantic

The evolution of Text-to-SQL is a microcosm of the broader evolution of Artificial Intelligence. We are moving from systems that require us to speak their language (code, rigid schemas,

graphs) to systems that understand ours (natural language, nuance, context).

The "Terno Way" demonstrates that achieving great accuracy is not about building a better graph. It is about building a better **bridge**. By leveraging a Semantic Layer (Metastore), abstracting complexity via Pseudo-Schemas (SQLShield), and employing Agentic Workflows that mimic human reasoning, we can achieve accuracy rates that were previously thought impossible.

Crucially, this semantic architecture is no longer a manual, brittle artefact. Terno shows that the Semantic Layer itself can be **automatically generated and continuously refined**. Through LLM-driven reasoning, raw enterprise schemas are transformed into business-aligned narratives: irrelevant tables are hidden, meaningful entities are described and renamed, logical relationships are inferred, ambiguity is removed, and semantic understanding improves with every interaction. What once required months of ontology engineering and domain workshops can now be achieved programmatically, safely, and incrementally.

Traditional mechanisms, such as ER diagrams, Ontologies, and Entity-Relation graphs, are artefacts of a human-centric past. They fail because they optimize for the eye, not the token. LLMs understand language better than graphs. Therefore, the architecture of the future must be built on words, descriptions, and narratives.

For the enterprise seeking to unlock the value of its data, the directive is therefore unambiguous. **Do not invest in tools that merely visualise your schema that freeze meaning in time. Invest instead in systems that can automatically understand, narrate, and evolve your data.**

Detailed Analysis of Supporting Research & Methodologies

Terno AI’s Architecture and the "Metastore" Advantage

Terno AI’s approach centers on a **semantic layer** that enables "text searches based on meaning and similarity".[6] This is fundamentally different from exact-match keyword systems. The **Metastore** [6] is the repository for this semantic intelligence. It stores "business knowledge" and "automated insights" (e.g., popular products, seasonal trends).[6] This allows the system to inject *data awareness* into the prompt. For example, knowing that "sales peak in December" might help the AI disambiguate a query about "holiday performance."

SQLShield: The Pseudo-Schema Innovation

The **SQLShield** component [9] is crucial for both security and accuracy. It uses a **Pseudo-Schema** mechanism. This involves "creating a pseudo-schema that keeps track of original table and column names while exposing different 'public' names to the LLM".[9] This "Schema

Augmentation" [13] allows the system to present a clean, English-friendly schema to the AI, decoupling the messy physical reality from the generation process. The "translation" step [9] ensures that the generated SQL is executable. This aligns with findings in **Gen-SQL** [8], which show that bridging NL and DB with pseudo-schemas improves efficiency.

Benchmark Validation: DAIL-SQL, DIN-SQL, and MAC-SQL

- **DAIL-SQL** [16] dominates the Spider leaderboard by focusing on "formatting schema as code/text." It validates the hypothesis that *how* the schema is described textually matters more than its graph topology. It uses "Schema Linking" [18] to identify relevant columns, a process heavily reliant on the "verbose descriptions" [1] found in semantic layers.
- **DIN-SQL** [14] introduces "Decomposed In-Context Learning." It breaks queries into sub-tasks. This mirrors Terno's **Agentic Workflow**.[12] It proves that "reasoning" (decomposition) beats "structure" (graph traversal) for complex queries.
- **MAC-SQL** [27] utilizes a multi-agent framework with "selector," "decomposer," and "refiner" agents. This supports the argument for a "Memory Layer" and "Logical Relationships" built through agent collaboration rather than static edges.

The "Memory Layer" and artefacts

Terno's **Artifact Store** [6] provides the "Memory Layer" requested in the prompt. It saves "intermediate artefacts, such as datasets, machine learning models, code, graphs, and charts." This allows the system to be **stateful**, a critical requirement for multi-turn accuracy that static graph models lack.

Report Authored By:

Engineering & Research Team, Terno AI
Date: Jan 12, 2026

(This report is based on a comprehensive analysis of research data, including Terno AI documentation [6], SQLShield specifications [9], and benchmark papers for Spider and BIRD.[14])

Works cited

1. The Data Warehouse Toolkit - NET, accessed December 29, 2025, https://thinkdataanalysis.blob.core.windows.net/files/UC_Presentation/The_Data_Warehouse_Toolkit.pdf
2. multi-hop question answering over knowledge graphs using large language models - arXiv, accessed December 29, 2025, https://arxiv.org/pdf/2404.19234
3. Compositional Generalization and Natural Language Variation: Can a Semantic Parsing

Approach Handle Both? | Request PDF - ResearchGate, accessed December 29, 2025, https://www.researchgate.net/publication/353488206_Compositional_Generalization_and_Natural_Language_Variation_Can_a_Semantic_Parsing_Approach_Handle_Both
4. Each graph is a new language: Graph Learning with LLMs - ACL Anthology, accessed December 29, 2025, https://aclanthology.org/2025.findings-acl.902.pdf
5. TabMeta: Table Metadata Generation with LLM-Curated Dataset and LLM-Judges - OpenReview, accessed December 29, 2025, https://openreview.net/pdf/356d2c7ac67361ccbd5550f5f212fc6837e55f04.pdf
6. Terno AI: Terno – Your AI Data Scientist, accessed December 29, 2025, https://terno.ai/
7. The data warehouse ETL toolkit - Find People, accessed December 29, 2025, https://people.wou.edu/~chenz/ORACLE/Data%20Warehouse%20ETL%20Toolkit1.pdf
8. Gen-SQL: Efficient Text-to-SQL By Bridging Natural Language Question And Database Schema With Pseudo-Schema - ACL Anthology, accessed December 29, 2025, https://aclanthology.org/2025.coling-main.256/
9. sqlshield · PyPI, accessed December 29, 2025, https://pypi.org/project/sqlshield/
10. A Synthetic Data Generation Framework for In-Domain Text-to-SQL Translation - arXiv, accessed December 29, 2025, https://arxiv.org/html/2509.25672v1
11. careers - Terno AI, accessed December 29, 2025, https://terno.ai/careers/
12. All Events - CloudxLab, accessed December 29, 2025, https://cloudxlab.com/events/all-events/
13. Next-Generation Database Interfaces: A Survey of LLM-Based Text-to-SQL - IEEE Computer Society, accessed December 29, 2025, https://www.computer.org/csdl/journal/tk/2025/12/11160657/29XcBYaVNDO
14. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL - arXiv, accessed December 29, 2025, https://arxiv.org/html/2406.08426v1
15. Seeking advice for developing a text-to-sql application : r/LLMDevs - Reddit, accessed December 29, 2025, https://www.reddit.com/r/LLMDevs/comments/196ldg3/seeking_advice_for_developing_a_texttosql/
16. Daily Papers - Hugging Face, accessed December 29, 2025, https://huggingface.co/papers?q=text-to-SQL%20models
17. LLMs Meet SQL: Revolutionizing Data Querying with Natural Language Processing, accessed December 29, 2025, https://levelup.gitconnected.com/llms-meet-sql-revolutionizing-data-querying-with-natural-language-processing-52487337f043.
18. PET-SQL: A Prompt-Enhanced Two-Round Refinement of Text-to-SQL with Cross-consistency - arXiv, accessed December 29, 2025, https://arxiv.org/html/2403.09732v4
19. Can LLMs Narrate Tabular Data? An Evaluation Framework for Natural Language Representations of Text-to-SQL System Outputs - ACL Anthology, accessed December 29, 2025, https://aclanthology.org/2025.emnlp-industry.60.pdf.
20. BIRD-SQL benchmark, accessed December 29, 2025, https://bird-bench.github.io/
21. SEED: Enhancing Text-to-SQL Performance and Practical Usability Through Automatic Evidence Generation - arXiv, accessed December 29, 2025, https://arxiv.org/html/2506.07423v1

22. Smaller Models, Smarter SQL: Arctic-Text2SQL-R1 Tops BIRD and Wins Broadly, accessed December 29, 2025, https://www.snowflake.com/en/engineering-blog/arctic-text2sql-r1-sql-generation-benchmark/
23. IBM text-to-SQL generator tops leaderboard - IBM Research, accessed December 29, 2025, https://research.ibm.com/blog/granite-LLM-text-to-SQL
24. Gen-SQL: Efficient Text-to-SQL By Bridging Natural ... - ACL Anthology, accessed December 29, 2025, https://aclanthology.org/2025.coling-main.256.pdf
25. Synthetic SQL Column Descriptions and Their Impact on Text-to-SQL Performance - arXiv, accessed December 29, 2025, https://arxiv.org/html/2408.04691v3
26. Semantic Layers: The Missing Link Between AI and Business Insight | by Axel Schwanke, accessed December 29, 2025, https://medium.com/@axel.schwanke/semantic-layers-the-missing-link-between-ai-and-business-insight-3c733f119be6
27. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL - ACL Anthology, accessed December 29, 2025, https://aclanthology.org/2025.coling-main.36.pdf
28. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL - arXiv, accessed December 29, 2025, https://arxiv.org/html/2312.11242v2?ref=blog.premai.io
29. Challenges and opportunities presented by generative AI in official statistics - Bank for International Settlements, accessed December 29, 2025, https://www.bis.org/ifc/publ/ifcb62_25.pdf
30. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation - arXiv, accessed August 29, 2023, https://arxiv.org/pdf/2308.15363.
31. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction - arXiv, accessed April 21, 2023, https://arxiv.org/pdf/2304.11015
32. Enhancing Text-to-SQL Capabilities of Large Language Models via Domain Database Knowledge Injection - rXiv, accessed September 24, 2024, https://arxiv.org/pdf/2409.15907