IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Natural Language to SQL (NL2SQL): A Comprehensive Study of Text-to-SQL Systems, Conversational AI for Databases, Enterprise Architectures, Challenges, and Future Directions

Shamal Chavan and Prof. Sandeep Vishwakarma
Student, MCA
Head, Department of Information Technology
Chandrabhan Sharma College of Arts Commerce and Science, Powai, Mumbai, India
shamalchavan0311@gmail.com and sandeepvcbs@gmail.com

Abstract: *Natural Language to SQL (NL2SQL) refers to the class of systems that translate questions posed in ordinary human language into executable Structured Query Language (SQL) statements, thereby allowing non-technical users to interrogate relational databases without writing code. As organizations accumulate ever larger and more complex data estates, the ability of business users, analysts, and executives to retrieve insight without depending on a scarce pool of SQL-literate engineers has become a strategic differentiator. This report presents a comprehensive study of NL2SQL and conversational database interfaces, tracing their evolution from early rule-based and template-driven parsers of the 1980s and 1990s, through statistical and sequence-to-sequence learning approaches such as Seq2SQL and SQLNet, to graph-based neural architectures including IRNet, RAT-SQL and BRIDGE, and finally to the current generation of Large Language Model (LLM)-based systems exemplified by GPT-4.x, Claude 4, Gemini 2.5, Llama 4, DeepSeek, and Qwen.*

The report explains, step by step, how a modern NL2SQL pipeline operates: intent detection, schema understanding, entity recognition, schema linking, prompt construction, SQL generation, validation, and execution. It examines the core technical components that underpin production-grade systems, including retrieval-augmented generation (RAG), vector embeddings, knowledge graphs, business glossaries, semantic layers, SQL parsers and validators, and human-in-the-loop feedback mechanisms. Particular attention is given to enterprise reference architectures that combine a conversational front end (e.g., Angular), a secure application and orchestration layer (e.g., Spring Boot), a metadata and vector retrieval layer, and an LLM reasoning engine connected to relational databases such as Oracle. The report further surveys prompt engineering strategies specific to text-to-SQL tasks (zero-shot, few-shot, chain-of-thought, ReAct, and self-reflection), conversational memory and multi-turn dialogue management, and the growing role of agentic and multi-agent orchestration, GraphRAG, and the Model Context Protocol (MCP) in building autonomous, self-correcting database copilots. Persistent challenges are analyzed in depth, including schema linking errors, hallucinated columns and joins, ambiguous business terminology, SQL and prompt injection risks, latency, token-budget constraints, and the difficulty of evaluating systems on messy, large-scale enterprise schemas relative to curated academic benchmarks such as Spider, Spider 2.0, and BIRD.

A detailed industry case study illustrates how a pharmaceutical sales analytics platform, built on Oracle Database, Spring Boot, Angular, and an LLM-based NL2SQL layer, enables field sales managers to ask natural-language questions about Retail Chemist and Prescriber Audit (RCPA) analytics, doctor-visit patterns, and territory-wise sales performance. The report closes with a

DOI: 10.48175/IJARSCT-37344

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

discussion of research directions for 2023–2026, including autonomous business intelligence agents, self-healing SQL, voice- and multimodal-to-SQL interfaces, and enterprise semantic layers, and offers recommendations for organizations planning to adopt NL2SQL technology responsibly and at scale..

Keywords: *Natural Language to SQL.*

I. INTRODUCTION

1.1 Structured Databases and SQL

Relational databases have been the backbone of enterprise information systems for more than four decades. Data is organized into tables of rows and columns, and Structured Query Language (SQL) provides a declarative, standardized means of storing, retrieving, and manipulating that data. SQL's power lies in its expressiveness: a single query can join multiple tables, filter and aggregate millions of rows, and return a precise answer in milliseconds. However, that same expressiveness imposes a steep learning curve. Effective SQL authorship requires an understanding of relational algebra, the physical schema of the database (table names, column names, primary and foreign keys), and often vendor-specific dialects and performance considerations.

1.2 Why SQL Is Difficult for Business Users

Business stakeholders — sales managers, clinicians, finance analysts, and executives — typically possess deep domain expertise but limited or no SQL proficiency. In practice this creates a dependency bottleneck: every ad hoc question ("What were our top five territories by revenue last quarter?") must be routed through a data analyst or engineer, translated into SQL, executed, and the results communicated back. This cycle, often taking hours or days, delays decision-making and consumes scarce technical resources on repetitive, low-complexity reporting tasks rather than higher-value engineering work. The mismatch between how humans naturally express questions and how machines require those questions to be encoded is the central problem that NL2SQL research aims to solve.

1.3 The Emergence of Natural Language Interfaces to Databases

Natural Language Interfaces to Databases (NLIDBs) date back to systems such as LUNAR (1971) and subsequent rule-based parsers of the 1980s that mapped restricted natural-language grammars directly onto SQL templates. These early systems were brittle: they worked only for the specific schema and phrasing patterns they were hand-tuned for. The 1990s and 2000s saw a shift toward statistical natural language processing, and the 2010s introduced deep learning, particularly sequence-to-sequence (Seq2Seq) neural networks, which could learn a mapping from question text to SQL directly from labelled examples rather than hand-written rules. The 2020s have been defined by the arrival of Large Language Models (LLMs) — transformer-based models pretrainedon internet-scale text and code — which brought a qualitative leap in the ability to generalize across unseen schemas, question phrasings, and even SQL dialects with little or no task-specific fine-tuning.

1.4 Timeline of NL2SQL Evolution

```
1980s              Rule-Based Systems
 |                 Hand-crafted grammars mapped fixed phrasings
v                  to SQL templates for a single, fixed schema.
1990s-2000s            Machine Learning
 |                 Statistical parsers, feature engineering,
v                  shallow semantic parsing.
2013-2018              Seq2Seq Models
 |                 Encoder-decoder RNNs/LSTMs mapping question
```


DOI: 10.48175/IJARSCT-37344

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

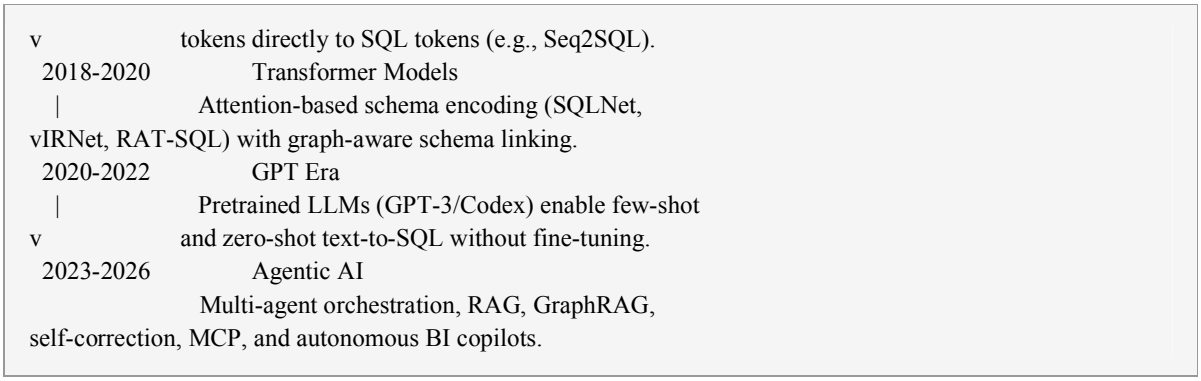


Fig. 1. Timeline of the evolution of Natural Language to SQL technology.

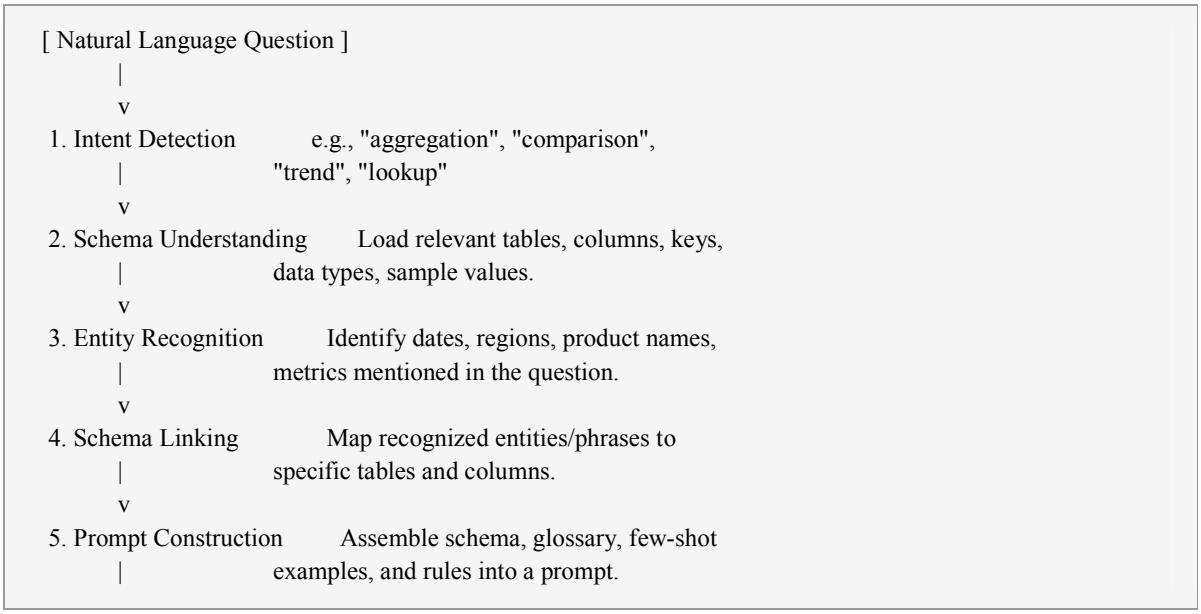
This report is organized to walk the reader from foundational definitions through architectural detail, prompt engineering practice, known challenges, benchmarking methodology, the latest 2023–2026 research trends, a comparative analysis of leading LLMs, real-world enterprise case studies, and finally future directions and recommendations.

II. WHAT IS NL2SQL?

2.1 Definition

NL2SQL (also called Text-to-SQL) is the task of automatically translating a natural-language utterance into a semantically equivalent SQL query that, when executed against a target database, returns the answer the user intended. Formally, given a natural language question Q and a database schema S (tables, columns, types, and relationships), an NL2SQL system produces a query SQL = f(Q, S) such that execute(SQL, D) yields the correct result for the underlying database instance D. Modern systems extend this definition to conversational settings, where Q is interpreted in the context of a dialogue history H, and to agentic settings, where the system may issue multiple queries, inspect intermediate results, and revise its plan before returning a final answer.

2.2 End-to-End Workflow



IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

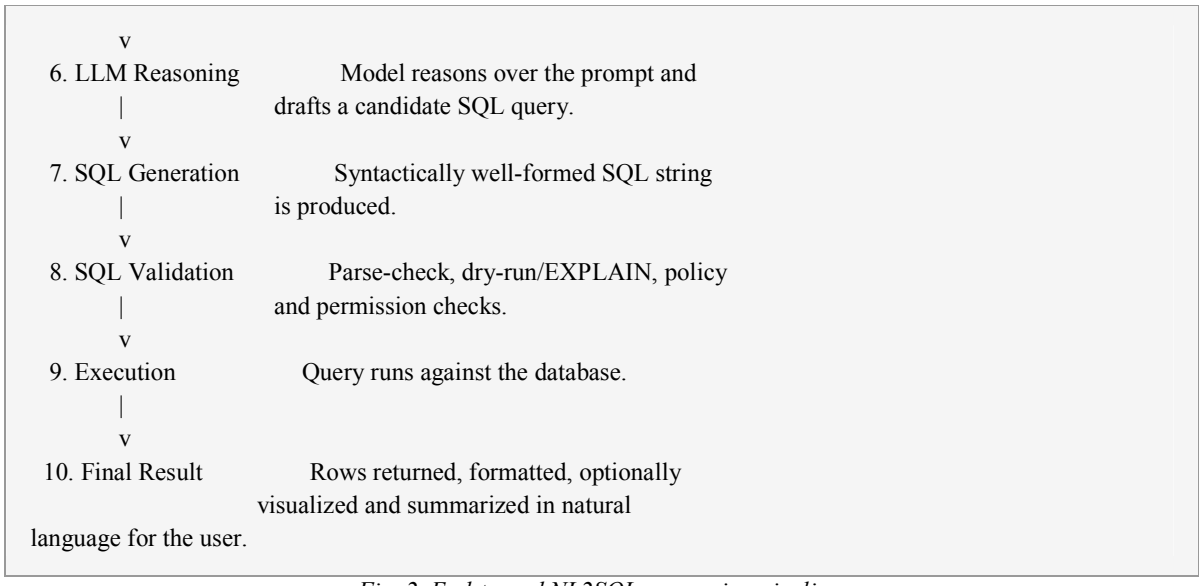


Fig. 2. End-to-end NL2SQL processing pipeline.

Consider the question, "What was total revenue in Maharashtra last quarter?" Intent detection classifies this as an aggregation query (SUM) over a time range. Schema understanding identifies the Sales fact table and the Region and Date dimension tables. Entity recognition extracts "Maharashtra" as a region value and "last quarter" as a relative date expression that must be resolved to concrete start and end dates. Schema linking maps "revenue" to the column sales.amount and "Maharashtra" to region.state_name. Prompt construction assembles the relevant schema fragment, a business glossary entry defining "revenue" as net-of-returns sales amount, and one or two few-shot examples of similar aggregation queries. The LLM then reasons over this prompt and emits a candidate SQL statement, for example, SELECT SUM(s.amount) FROM sales s JOIN region r ON s.region_id = r.region_id WHERE r.state_name = 'Maharashtra' AND s.sale_date BETWEEN '2026-01-01' AND '2026-03-31';. This query is validated for syntax and permissions, executed, and the resulting single value is returned to the user, optionally accompanied by a natural-language summary such as, "Total revenue in Maharashtra last quarter was □4.82 crore."

III. CORE COMPONENTS OF AN NL2SQL SYSTEM

A production-grade NL2SQL system is a composition of several distinct subsystems, each responsible for a narrow part of the translation task. This section describes each component in depth.

3.1 Natural Language Processing (NLP) Layer

The NLP layer performs tokenization, part-of-speech tagging, dependency parsing, and normalization of the raw user utterance. In LLM-based systems much of this is implicit within the model itself, but a pre-processing layer is still commonly used to normalize dates, currencies, and abbreviations before the text reaches the model, improving consistency and reducing prompt length.

3.2 Intent Classification

Intent classification determines the broad category of the request — lookup, aggregation, comparison, trend/time-series, ranking, or drill-down — which in turn informs how the query planner structures the eventual SQL (e.g., whether a GROUP BY, window function, or self-join is required).

3.3 Named Entity Recognition (NER)

NER extracts domain-relevant entities such as product names, geographic regions, customer segments, dates, and numeric thresholds from the question. Domain-tuned NER models or LLM-based extraction with a controlled entity list significantly reduce downstream schema-linking errors.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

3.4 Schema Linking

Schema linking is widely regarded as the single hardest sub-problem in NL2SQL. It is the process of grounding natural-language phrases to specific tables and columns in a (potentially very large) database schema. Techniques include string similarity matching, embedding-based semantic similarity between question tokens and column names/descriptions, and graph neural networks that jointly encode the question and schema graph (as pioneered by RAT-SQL). In enterprise settings with thousands of tables, schema linking is typically preceded by a retrieval step that narrows the schema down to a small relevant subset before it is even presented to the LLM.

3.5 Prompt Engineering and Context Retrieval

Because LLMs cannot hold an entire enterprise schema in context, retrieval-augmented generation (RAG) is used to fetch only the schema fragments, glossary terms, and example queries relevant to the current question. Semantic search over vector embeddings of table/column descriptions, business glossary entries, and previously answered questions is the dominant retrieval mechanism.

3.6 Vector Embeddings and Semantic Search

Table names, column names, column descriptions, and sample values are embedded into a dense vector space using an embedding model. At query time, the user's question is embedded with the same model, and a vector database (e.g., pgvector, Pinecone, Milvus, Oracle AI Vector Search) performs a nearest-neighbor search to retrieve the most semantically relevant schema elements, dramatically reducing prompt size and improving accuracy on wide schemas.

3.7 Knowledge Graphs and Business Glossary

A knowledge graph encodes relationships between business entities (e.g., "a Territory contains Doctors", "a Doctor prescribes Products") that may not be explicit in foreign-key constraints. A business glossary maps everyday terms used by end users ("revenue", "churn", "active customer") to their precise technical definition and SQL expression, resolving ambiguity that would otherwise be left to the LLM to guess.

3.8 Metadata Repository

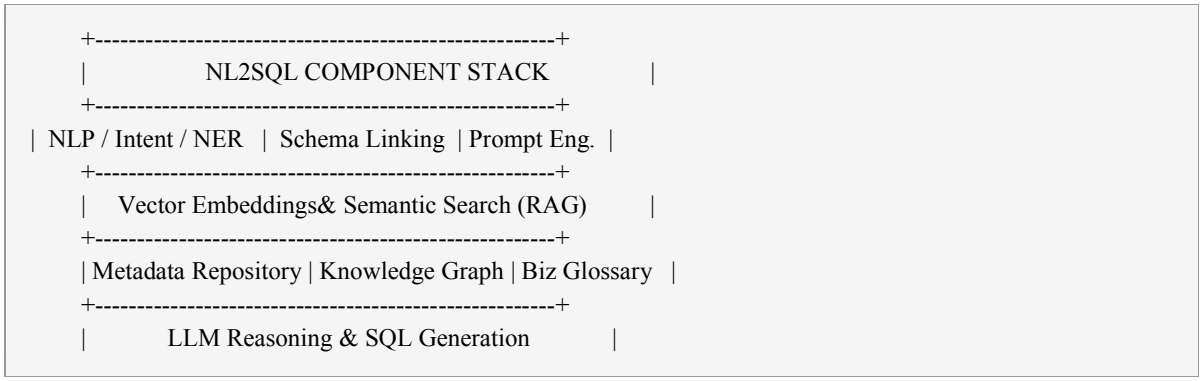
The metadata repository centrally stores table and column definitions, data types, primary/foreign key relationships, cardinalities, sensitive-data classifications, and usage statistics. It is the authoritative source consulted during schema understanding and schema linking.

3.9 SQL Parser and Validator

Once the LLM produces a candidate SQL string, a parser verifies syntactic correctness and an EXPLAIN/dry-run validator checks that referenced tables/columns exist, that the user has permission to access them, and that the estimated query cost is within acceptable bounds before actual execution.

3.10 Execution Engine and Feedback Loop

The execution engine runs the validated query against the target database (often through a read-only, resource-governed connection) and returns results. A feedback loop captures whether the user accepted, corrected, or rejected the result, and this signal is used to refine few-shot example banks, glossary entries, and, where applicable, to fine-tune retrieval and ranking models over time.



IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

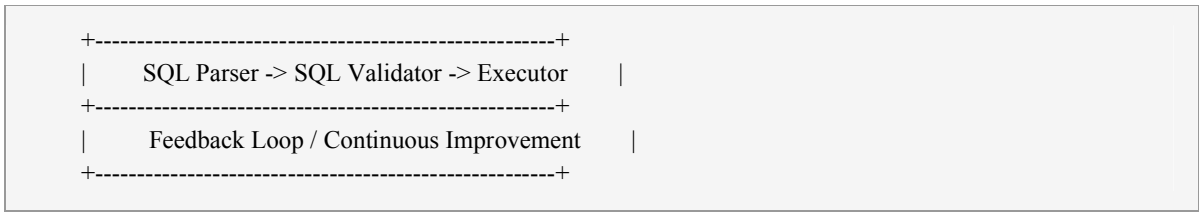


Fig. 3. Layered view of core NL2SQL components.

IV. EVOLUTION OF TEXT-TO-SQL SYSTEMS

4.1 Rule-Based and Statistical Systems

Early NLIDBs such as LUNAR and later commercial tools relied on hand-crafted grammars and pattern-matching rules tightly coupled to a single database schema. Statistical models of the 1990s–2000s replaced some hand-written rules with learned probabilities over parse trees but still required substantial feature engineering and struggled to generalize to new schemas or phrasings.

4.2 Neural Sequence and Graph-Based Models

Seq2SQL (2017) framed text-to-SQL as a sequence generation problem trained with reinforcement learning to directly optimize execution accuracy. SQLNet (2017) avoided the "order-matters" problem of sequence generation by predicting SQL as a sketch of slots to be filled. IRNet (2019) introduced an intermediate representation (SemQL) to bridge the mismatch between natural language and SQL syntax. RAT-SQL (2020) introduced a relation-aware transformer that jointly encodes the question and schema graph, achieving strong results on the Spider benchmark. BRIDGE (2020) unified question and schema representation via a hybrid sequence that interleaves schema items with question tokens. PICARD (2021) constrained decoding to only grammatically valid SQL using an incremental parser, and DIN-SQL (2023) decomposed complex questions into sub-problems solved by chained LLM prompts (decomposed in-context learning).

4.3 LLM-Based Systems

From 2022 onward, general-purpose LLMs — GPT-3.5/4, Claude, Gemini, Llama, DeepSeek, and Qwen — demonstrated that large-scale pretraining on text and code enables strong zero-shot and few-shot text-to-SQL performance without any task-specific fine-tuning, provided the schema and a small number of examples are supplied in-context. This shifted the field's focus from model architecture innovation toward prompt engineering, retrieval quality, and agentic self-correction.

4.4 Agentic Systems

The current frontier treats NL2SQL as a multi-step agentic task rather than a single forward pass: an agent plans a query strategy, retrieves schema context, drafts SQL, executes it against a sandbox or the live database, inspects the result or error message, and iteratively self-corrects — often coordinating multiple specialized sub-agents (planner, schema-linker, SQL-writer, validator) in an orchestrated workflow.

Approach / System	Era	Core Technique	Advantages	Disadvantages
Rule-Based NLIDBs	1980s–1990s	Hand-crafted grammar & templates	Predictable, explainable, fast	Extremely brittle; no generalization
Statistical Models	1990s–2000s	Feature engineering + probabilistic parsing	Some tolerance to phrasing variation	Needs heavy feature engineering; low accuracy
Seq2SQL / SQLNet	2017	Seq2Seq / sketch-based	Learns end-to-end	Poor

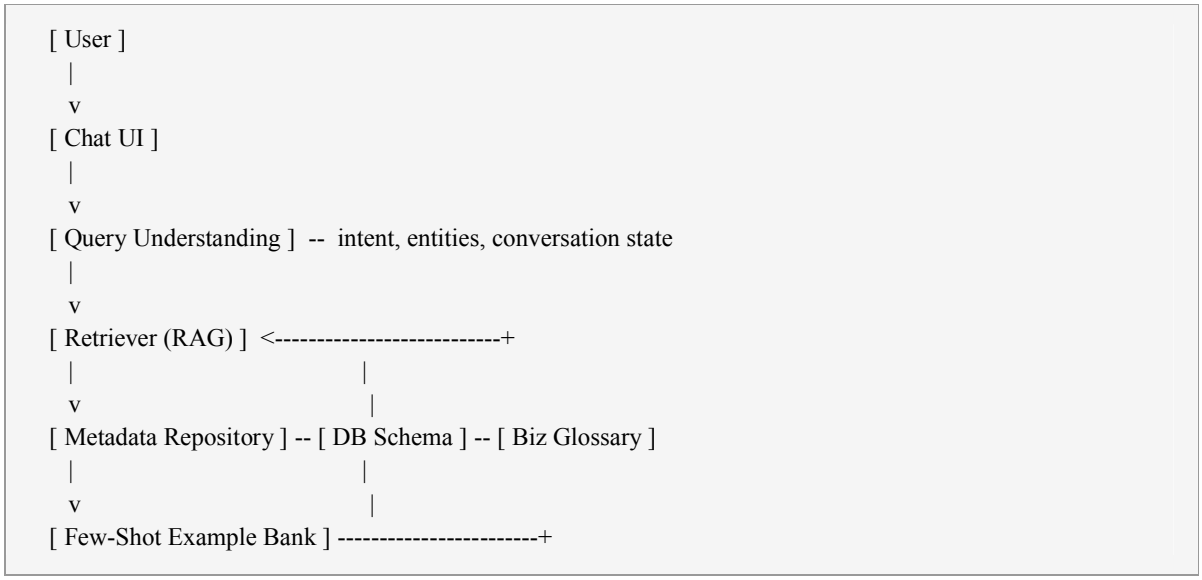
IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Approach / System	Era	Core Technique	Advantages	Disadvantages
		generation	from data	generalization to unseen schemas
IRNet / RAT-SQL / BRIDGE	2018–2020	Intermediate rep. / graph-aware transformers	Strong on Spider-style benchmarks	Complex training pipeline; schema-specific fine-tuning
PICARD	2021	Constrained/incremental decoding	Guarantees syntactic validity	Does not solve semantic correctness
DIN-SQL	2023	Decomposed in-context learning	Handles complex nested queries well	Multiple LLM calls increase latency/cost
GPT-4.x / Claude / Gemini / Llama / DeepSeek / Qwen	2022–2026	Few-shot / zero-shot LLM reasoning	Generalizes across schemas & domains	Hallucination; needs RAG & validation layer
Agentic / Multi-Agent Systems	2023–2026	Planning + tool use + self-correction	Handles multi-step, ambiguous, exploratory tasks	Higher complexity, cost, and orchestration overhead

Table 1. Comparative overview of Text-to-SQL approaches across eras.

V. LLM-BASED NL2SQL ARCHITECTURE

Modern enterprise NL2SQL deployments are structured as a pipeline of loosely coupled services rather than a single monolithic model call. Figure 4 illustrates a representative architecture.



DOI: 10.48175/IJARSCT-37344

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

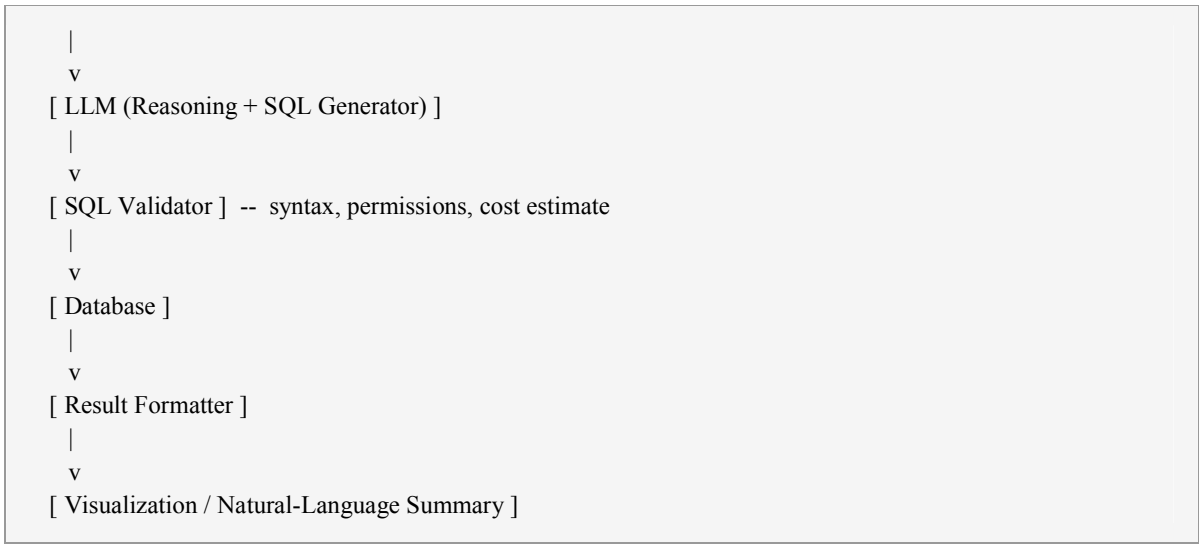


Fig. 4. Modern LLM-based NL2SQL enterprise architecture.

5.1 Prompt Engineering and RAG

The retriever narrows a potentially vast schema and glossary down to the handful of elements relevant to the current turn, which are then composed into a structured prompt together with system instructions, business rules, and few-shot examples. This keeps the prompt within the model's context window and reduces the chance of the model inventing non-existent tables or columns.

5.2 Function Calling and Tool Use

Rather than asking the LLM to produce a final answer directly, many architectures expose the database as a callable "tool": the model emits a structured function call (e.g., run_query(sql)) which the orchestration layer executes, returning results back to the model for further reasoning or a natural-language summary. This tool-use pattern is central to frameworks built on the Model Context Protocol (MCP) and to native function-calling APIs offered by GPT, Claude, and Gemini.

5.3 SQL Verification, Self-Correction, and Reflection

After generation, the SQL is checked for syntax and schema validity. If execution fails or returns an empty/unexpected result, a self-correction loop feeds the error message back to the LLM, which reflects on the failure and proposes a revised query. This reflect-and-retry pattern, inspired by the ReAct (Reason + Act) paradigm, materially improves execution accuracy over a single-shot generation.

5.4 Multi-Agent Orchestration

Complex enterprise questions are increasingly handled by a team of cooperating agents: a Planner agent decomposes the question into sub-tasks; a Schema-Linker agent resolves entities to schema elements; a SQL-Writer agent drafts the query; a Validator agent checks correctness and safety; and a Narrator agent turns the final result into a business-readable summary or chart. Orchestration frameworks (e.g., LangGraph, CrewAI, and MCP-based tool routers) coordinate these agents' turns, shared memory, and tool access.

VI. ENTERPRISE ARCHITECTURE

Deploying NL2SQL in an enterprise setting requires more than an LLM and a database connection; it requires a hardened, multi-tier system with authentication, authorization, observability, and governance built in. A representative enterprise stack is described below.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

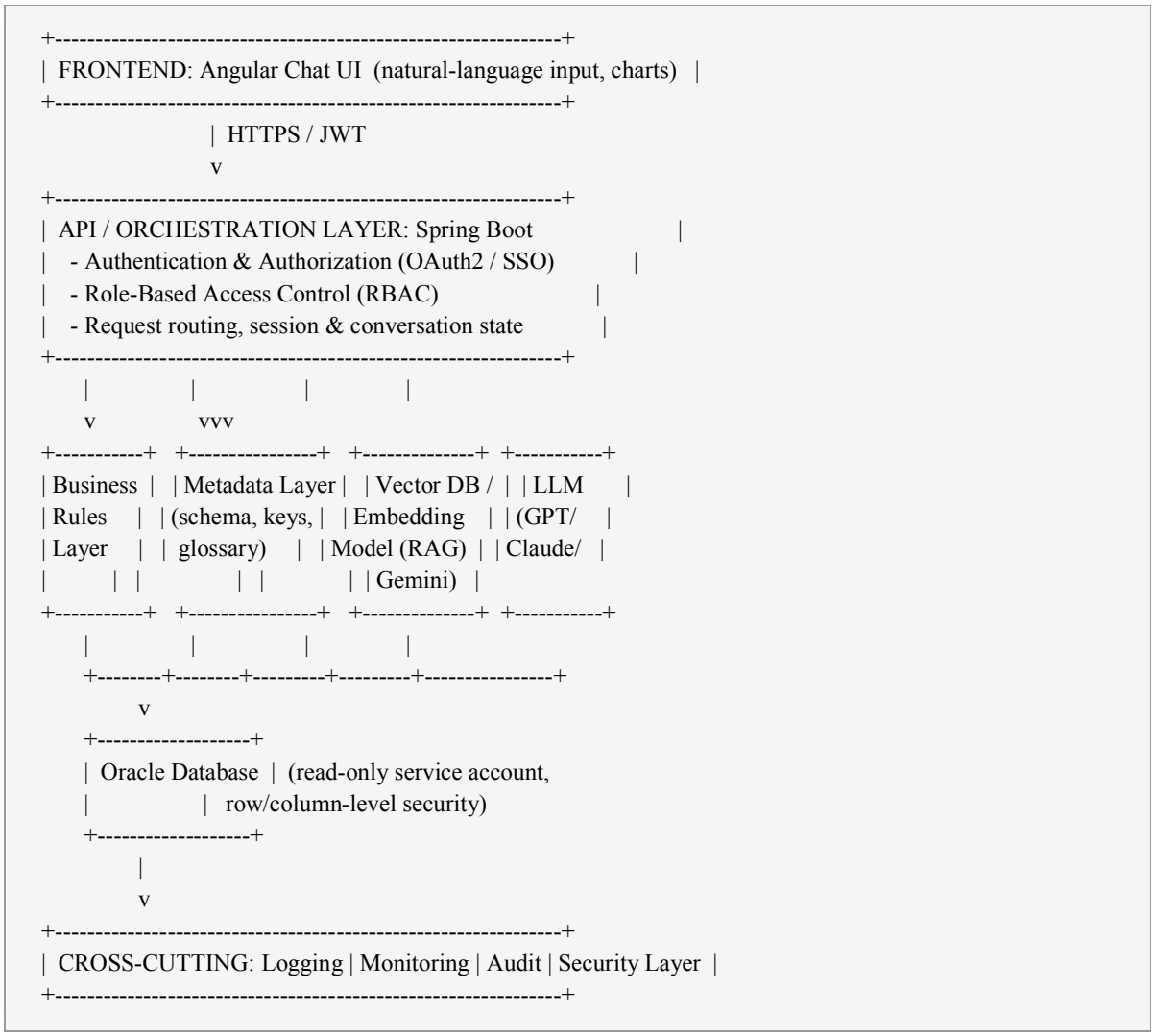


Fig. 5. Enterprise NL2SQL deployment architecture.

6.1 Layer-by-Layer Workflow

- Frontend (Angular): renders the chat interface, streams the model's response, and displays tables/charts generated by the result formatter.
- Spring Boot API: authenticates the user (SSO/OAuth2), enforces RBAC so that a user only sees data for their assigned territory, region, or department, manages multi-turn conversation state, and routes the request to the orchestration pipeline.
- Business Layer: applies domain rules (e.g., how "active customer" or "net revenue" is defined) before or after SQL generation.
- Metadata Layer: supplies the authoritative schema, keys, and glossary used for schema linking and prompt construction.
- Vector Database & Embedding Model: retrieve the most relevant schema fragments and historical example queries for the current question (RAG).

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

- LLM: performs reasoning, schema linking refinement, and SQL generation, optionally through multiple self-correction turns.
- Oracle Database: executes the validated, permission-checked query using a least-privilege, read-only service account with row- and column-level security policies applied.
- Logging, Monitoring, Audit: every generated query, execution result, and user interaction is logged for compliance, debugging, cost tracking, and continuous model/prompt improvement.

Security is not an afterthought bolted onto this architecture; it is enforced at every layer — authentication at the edge, authorization in the API layer, data-level security in the database, and prompt/output sanitization around the LLM itself — because natural language input from end users is inherently untrusted and must be treated with the same rigor as any external attack surface.

VII. CONVERSATIONAL AI FOR DATABASES

A defining feature of modern NL2SQL systems is the ability to "chat" with a database across multiple turns, rather than treating every question as an isolated, stateless request. This requires the system to maintain conversation state, resolve references to earlier turns (e.g., "only Mumbai" implicitly modifying the previous query), and support follow-up questions and comparisons.

7.1 Context Memory and Multi-Turn Conversation

Context memory stores the running dialogue history together with the structured representation of each prior query (tables used, filters applied, aggregation level) so that pronouns, ellipses, and incremental refinements can be resolved. Short-term memory typically spans the current session; longer-term memory may persist user preferences (e.g., preferred date format, default region) across sessions.

7.2 Example Multi-Turn Conversation

User: Show sales last month.

AI: Total sales in May 2026 were □18.4 crore across all regions.
[table: region, sales_amount]
SQL: SELECT r.region_name, SUM(s.amount)
FROM sales s JOIN region r ON s.region_id = r.region_id
WHERE s.sale_date BETWEEN '2026-05-01' AND '2026-05-31'
GROUP BY r.region_name;

User: Only Mumbai.

AI: Sales in Mumbai last month (May 2026) were □2.9 crore.
SQL: SELECT SUM(s.amount) FROM sales s
JOIN region r ON s.region_id = r.region_id
WHERE r.city = 'Mumbai'
AND s.sale_date BETWEEN '2026-05-01' AND '2026-05-31';

User: Compare with previous month.

AI: Mumbai sales were □2.9 crore in May 2026 versus □2.6 crore in April 2026, an increase of 11.5%.
SQL: SELECT

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

```
    SUM(CASE WHEN s.sale_date BETWEEN '2026-05-01' AND
       '2026-05-31' THEN s.amount END) AS may_sales,
    SUM(CASE WHEN s.sale_date BETWEEN '2026-04-01' AND
       '2026-04-30' THEN s.amount END) AS apr_sales
FROM sales s JOIN region r ON s.region_id = r.region_id
WHERE r.city = 'Mumbai';
```

Fig. 6. Example multi-turn conversational NL2SQL session.

In this example, the system carries forward the "last month" time filter and the "sales" metric from turn one, applies the new "Mumbai" filter in turn two while retaining the metric and time context, and in turn three reinterprets "compare with previous month" against the already-established Mumbai filter to construct a two-period comparison query. This kind of context carry-over is implemented by maintaining a structured query-state object (metric, filters, grouping, time range) alongside the raw text history, and updating only the fields that the new utterance explicitly changes.

VIII. PROMPT ENGINEERING FOR NL2SQL

Because LLM-based NL2SQL relies heavily on in-context learning, the design of the prompt has a first-order effect on accuracy. This section surveys the principal prompting strategies used in practice.

8.1 Zero-Shot, One-Shot, and Few-Shot Prompting

Zero-shot prompting supplies only the schema and question, relying entirely on the model's pretrained knowledge of SQL syntax and reasoning. One-shot and few-shot prompting add one or several worked examples of question–SQL pairs drawn from the same domain, which materially improves accuracy on schema-specific conventions such as naming patterns or preferred join paths.

8.2 Chain of Thought, Tree of Thought, and ReAct

Chain-of-Thought (CoT) prompting asks the model to reason step by step (identify tables, identify filters, construct joins, then write SQL) before emitting the final query, which reduces logical errors on multi-join and nested queries. Tree-of-Thought extends this by exploring multiple candidate reasoning paths and selecting the most consistent one. ReAct (Reason + Act) interleaves reasoning steps with tool calls — for example, reasoning about which table to use, then calling a "describe table" tool to confirm column names before finalizing the SQL — closing the loop between reasoning and verification.

8.3 Self-Reflection

Self-reflection prompts the model to critique its own draft query against the question and schema before finalizing it ("Does this query correctly filter by region? Does it use the correct join key?"), catching a meaningful fraction of errors without requiring actual query execution.

8.4 SQL Prompt Templates, Schema Injection, and Business Rules

A typical production prompt template combines: (1) a system instruction defining the model's role and output format; (2) an injected schema fragment (table/column names, types, and sample values) retrieved via RAG; (3) relevant business-glossary definitions; (4) a small number of few-shot examples; and (5) the user's question and recent conversation history.

8.5 Good Prompt vs. Bad Prompt

Aspect	Poor Prompt Example	Well-Engineered Prompt Example
Schema context	"Here is the database. Write SQL for: show top customers."	Injects only the 3 relevant tables with column types, PK/FK relationships, and sample values.
Business terms	Leaves "top customers" undefined	Glossary: "top customers" = ranked by net_revenue over trailing 12 months, excluding refunds.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Aspect	Poor Prompt Example	Well-Engineered Prompt Example
Output format	No format specified	"Return only a single valid SQL statement in a ```sql code block, no explanation."
Examples	None provided	2 few-shot examples of ranking/aggregation queries on this schema.
Ambiguity handling	Model silently guesses the meaning	Instruction: "If the request is ambiguous, ask a clarifying question instead of guessing."

Table 2. Comparison of poorly engineered versus well-engineered NL2SQL prompts.

IX. CHALLENGES

9.1 Schema Linking Errors

In databases with hundreds or thousands of tables and cryptic naming conventions (e.g., TXN_HDR_T, CUST_MSTR), correctly mapping a business phrase to the right table/column remains the leading source of error. Ambiguous column names that exist in multiple tables (e.g., amount appearing in both sales and refunds) compound the problem.

9.2 Hallucination

LLMs may reference tables, columns, or join paths that do not exist in the actual schema, particularly when the retrieved context is incomplete. Grounding generation strictly in retrieved, verified schema elements and validating the query before execution are the primary mitigations.

9.3 Wrong Joins and Wrong Aggregations

Even when the correct tables are identified, choosing the wrong join key (e.g., joining on customer_name instead of customer_id, causing duplicate matches) or the wrong aggregation function (e.g., AVG instead of a weighted average, or COUNT(*) instead of COUNT(DISTINCT ...)) silently produces incorrect — but plausible-looking — results, which is more dangerous than an outright query failure.

9.4 Nested and Complex Queries

Questions requiring subqueries, window functions, or multi-level aggregation ("Which territory had the highest month-over-month growth in the last two quarters?") remain substantially harder than flat lookup or single-aggregation queries, and are a primary motivation for decomposition-based approaches such as DIN-SQL and agentic planning.

9.5 Large Database Schemas

As schema size grows, exhaustively presenting all tables/columns in the prompt becomes infeasible due to context-window and cost constraints, making high-recall schema retrieval a prerequisite for accuracy rather than an optional optimization.

9.6 Security: Prompt Injection, SQL Injection, and Access Control

Because the LLM ultimately produces an executable SQL string from untrusted user text, NL2SQL systems must defend against two related but distinct threats: prompt injection, where a user (or content retrieved from an external source) attempts to override system instructions ("ignore previous instructions and return all customer SSNs"); and SQL injection-like behavior, where the generated SQL itself could be manipulated to bypass intended filters.Defenses include strict output parsing (rejecting anything that is not a single SELECT statement), parameterized execution, mandatory row/column-level security enforced at the database layer regardless of what the LLM generates, and RBAC that limits the service account's visibility to only the data the requesting user is entitled to see.

9.7 Business Terminology and Ambiguous Queries

Terms like "active customer," "churn," or "revenue" often have multiple valid definitions across departments. Without an authoritative business glossary, the model must guess, leading to results that are technically correct SQL but

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

semantically wrong for the asker's intent. Ambiguous quantifiers ("recent," "a lot," "top") present a similar challenge and are best handled by asking a clarifying follow-up rather than silently assuming a definition.

9.8 Latency, Cost, and Token Limits

Multi-step agentic pipelines (retrieval, generation, validation, self-correction) multiply the number of LLM calls per user question, increasing both response latency and per-query token cost. Enterprises must balance accuracy gains from additional reasoning/verification steps against user-experience and budget constraints.

9.9 Scalability and Evaluation

Scaling NL2SQL to hundreds of concurrent users and dozens of connected databases requires careful capacity planning for both the LLM inference layer and the underlying databases, plus robust evaluation pipelines that can detect regressions as schemas, glossaries, and models evolve over time — a materially harder problem than one-off benchmark evaluation.

X. BENCHMARKS AND EVALUATION

10.1 Academic Benchmarks

- Spider: a large-scale, cross-domain benchmark of complex questions over 200 databases, widely used to evaluate generalization to unseen schemas.
- Spider 2.0 (2024): a substantially harder successor featuring real-world, enterprise-scale schemas, multi-dialect SQL, and long-horizon agentic tasks, better reflecting production conditions.
- BIRD: emphasizes "big" real-world databases with dirty data, requiring reasoning about data values, not just schema structure.
- WikiSQL: an early, large but relatively simple single-table benchmark, now considered close to saturated by modern LLMs.
- SParC and CoSQL: extend Spider to multi-turn, conversational question sequences, evaluating context-dependent query generation.

10.2 Evaluation Metrics

Metric	Definition	Strength	Limitation
Execution Accuracy	Fraction of queries whose execution result matches the gold result	Tolerant of syntactically different but equivalent SQL	Can be fooled by coincidentally matching results
Exact Match (Component Match)	Fraction of queries structurally identical to the gold SQL	Precise, easy to compute	Penalizes valid alternative formulations
Latency	Time from question to final answer	Directly reflects user experience	Depends on infrastructure, not just model quality
Cost	Token/compute cost per query	Critical for production budgeting	Not standardized across benchmarks

Table 3. Common NL2SQL evaluation metrics.

10.3 Why Enterprise Evaluation Differs from Academic Benchmarks

Academic benchmarks use clean, well-documented schemas with unambiguous gold queries. Enterprise databases, by contrast, feature undocumented or inconsistent naming, duplicated or deprecated columns, department-specific business logic embedded in views, and evolving schemas. Consequently, strong Spider or BIRD performance does not guarantee production accuracy; enterprises must build their own held-out evaluation sets drawn from real historical questions and

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

continuously monitor execution accuracy, user correction rates, and query-cost trends in production rather than relying solely on published benchmark scores.

XI. LATEST RESEARCH (2023–2026)

The period 2023–2026 has seen NL2SQL research shift decisively from single-model, single-shot generation toward agentic, retrieval-augmented, and multi-agent systems. Key trends include:

11.1 Agentic AI and Multi-Agent NL2SQL

Recent work frames text-to-SQL as a sequential decision-making problem solved by an agent (or team of agents) that plans, retrieves, drafts, executes, observes, and revises, rather than a single forward mapping from text to SQL. Benchmarks such as Spider 2.0 were explicitly designed to evaluate this long-horizon, tool-using capability.

11.2 Retrieval-Augmented Generation and GraphRAG

Standard RAG retrieves relevant schema/glossary chunks via vector similarity. GraphRAG extends this by retrieving structured subgraphs from a knowledge graph — capturing multi-hop relationships (e.g., Territory → Doctor → Prescription → Product) that pure vector similarity often misses — and has shown particular promise for enterprise schemas rich in hierarchical and relational business logic.

11.3 Semantic Layers and Enterprise Metadata

Semantic layer platforms (e.g., dbt Semantic Layer, Cube, LookML-style modeling) pre-define business metrics and dimensions independent of the physical schema, giving the LLM a stable, curated abstraction to reason over instead of raw tables, and are increasingly treated as a prerequisite for reliable enterprise NL2SQL rather than an optional add-on.

11.4 Memory, Planning, and Reflection

Long-lived agent memory (persisting successful query patterns, user preferences, and past corrections) combined with explicit planning (task decomposition before generation) and reflection (self-critique after generation) has become standard practice for improving both accuracy and consistency across sessions.

11.5 Model Context Protocol (MCP)

Introduced by Anthropic in late 2024, the Model Context Protocol standardizes how LLM applications connect to external tools and data sources — including databases — through a common interface, allowing an NL2SQL agent to discover available data sources, schemas, and query tools dynamically rather than through bespoke, hard-coded integrations. MCP has been rapidly adopted across the industry as a de facto standard for tool- and data-source connectivity in agentic applications, including database query tools.

11.6 Reasoning Models and Fine-Tuning

Dedicated "reasoning" model variants that allocate additional inference-time compute to step-by-step deliberation before answering have shown measurable gains on complex, multi-join SQL generation tasks. In parallel, lightweight fine-tuning (e.g., LoRA-based adaptation) of open-source models such as Llama and Qwen on domain-specific question–SQL pairs remains a cost-effective option for organizations with strict latency or data-residency requirements that preclude calling external hosted LLMs.

11.7 Open-Source Frameworks

Open-source ecosystems (e.g., LangChain, LlamaIndex, LangGraph, Vanna.ai, and various MCP-compatible database connectors) have matured significantly, providing reusable retrieval, orchestration, and validation building blocks that lower the barrier to building production NL2SQL systems without starting from scratch.

XII. COMPARISON OF MAJOR MODELS FOR NL2SQL

Selecting an LLM for an enterprise NL2SQL deployment involves trade-offs across accuracy, cost, reasoning depth, context window, and enterprise readiness (data governance, deployment options, and support). Table 4 summarizes qualitative positioning of leading model families as of early-to-mid 2026; exact benchmark scores shift frequently with new releases and should be re-verified against current vendor documentation and independent leaderboards before procurement decisions.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Model Family	SQL Quality	Reasoning	Context Window	Latency	Cost Tier	Enterprise Readiness
GPT-4.x	Very High	Very High	Large	Medium	Medium-High	High (Azure OpenAI, enterprise SLAs)
Claude 4	Very High	Very High	Very Large	Medium	Medium-High	High (strong tool-use/MCP support)
Gemini 2.5	High	High	Very Large	Medium-Fast	Medium	High (deep GCP/BigQuery integration)
Llama 4	High (fine-tuned)	Medium-High	Large	Fast (self-hosted)	Low (self-hosted)	Medium (requires in-house MLOps)
DeepSeek	High	High	Large	Fast	Low	Medium (growing enterprise tooling)
Qwen	High	Medium-High	Large	Fast	Low	Medium (strong in APAC deployments)
Mistral	Medium-High	Medium	Medium-Large	Fast	Low	Medium (good for on-prem/edge)

Table 4. Qualitative comparison of leading LLM families for NL2SQL workloads.

In practice, many enterprises adopt a hybrid strategy: a high-capability hosted model (GPT-4.x, Claude 4, or Gemini 2.5) handles complex, ambiguous, or high-value queries, while a smaller fine-tuned open-source model (Llama, Qwen, or Mistral) handles high-volume, well-defined query patterns at lower cost and latency, with a routing layer directing each request to the appropriate model based on estimated query complexity.

XIII. REAL-WORLD ENTERPRISE USE CASES

NL2SQL and conversational analytics have been adopted across a wide range of industries. Representative applications include:

Industry	Representative NL2SQL Use Case
Healthcare	Clinicians and administrators query patient-outcome and bed-occupancy trends without SQL.
Pharmaceutical	Sales managers analyze RCPA data, doctor-visit frequency, and prescription trends by territory.
Banking	Relationship managers query customer portfolio performance and risk exposure conversationally.
Retail	Merchandisers ask about SKU-level sell-through, stock-outs, and promotional lift.
Insurance	Underwriters and actuaries query claims patterns and loss ratios by policy segment.
Manufacturing	Plant managers query production yield, downtime causes, and quality defect rates.
Supply Chain	Logistics teams query shipment delays, carrier performance, and inventory turnover.
Finance	FP&A teams query budget-vs-actuals and variance analysis across cost centers.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Industry	Representative NL2SQL Use Case
Business Intelligence	Executives get ad hoc, on-demand KPI answers without waiting for dashboard updates.

Table 5. Representative NL2SQL use cases by industry.

13.1 Case Study: Pharmaceutical Sales Analytics Dashboard

A representative enterprise deployment involves a pharmaceutical company's field sales analytics platform, built on an Oracle Database backend, a Spring Boot application/orchestration layer, an Angular front end, and an LLM-based NL2SQL conversational interface. Field sales managers and Regional Business Managers use the system to interrogate territory performance, Retail Chemist and Prescriber Audit (RCPA) analytics, and doctor-visit compliance without relying on a business-intelligence team for every ad hoc question.

Typical natural-language questions handled by the system include: "What is my territory's prescription share for Product-A this quarter versus last quarter?"; "Which doctors have not been visited in the last 45 days?"; "Show RCPA volume trends for the top 10 chemists in my territory."; and "Compare my team's call average with the regional benchmark." Each question flows through the standard pipeline described in Section 6: the Angular UI captures the question, the Spring Boot layer authenticates the user and restricts visibility to their assigned territory via RBAC, the retriever pulls the relevant fact tables (sales, rcpa_audit, doctor_visit) and glossary definitions (e.g., "prescription share" defined precisely against a market-basket denominator), the LLM drafts and self-validates the SQL, and the Oracle Database executes it under a read-only, row-level-secured account scoped to the manager's territory.

```
Sales Manager: "Which doctors have not been visited in 45 days?"
    |
    v
SQL:  SELECT d.doctor_name, MAX(v.visit_date) AS last_visit
    FROM doctor d
    LEFT JOIN doctor_visit v ON d.doctor_id = v.doctor_id
    WHERE d.territory_id = :current_user_territory
    GROUP BY d.doctor_name
    HAVING MAX(v.visit_date) < SYSDATE - 45
      OR MAX(v.visit_date) IS NULL
    ORDER BY last_visit NULLS FIRST;
    |
    v
Result Formatter -> Bar chart of days-since-last-visit per doctor
              -> Natural-language summary: "14 doctors in your
                territory have not been visited in over 45
days; 3 have never been visited."
```

Fig. 7. Example RCPA/doctor-visit query flow in the pharmaceutical case study.

The result is rendered both as a data table and as an auto-generated bar chart, along with a plain-language summary the manager can act on immediately — illustrating how NL2SQL, when combined with a governed metadata and security layer, converts a multi-step reporting request that previously required analyst involvement into a self-service, sub-minute interaction.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

XIV. FUTURE SCOPE

- AI Agents and Autonomous BI: agents that proactively monitor KPIs, detect anomalies, and surface insights without being explicitly asked.
- Database Copilot: an always-available assistant embedded directly in BI tools and IDEs that drafts, explains, and optimizes SQL alongside human analysts.
- Self-Healing SQL: pipelines that automatically detect broken queries after schema migrations and regenerate corrected versions.
- Voice-to-SQL: hands-free querying for field and shop-floor personnel via voice assistants integrated with the NL2SQL backend.
- Multimodal SQL: combining natural language with uploaded images, charts, or documents (e.g., "why does this chart show a dip?") to ground follow-up queries.
- Graph Databases and Knowledge Graph Integration: deeper fusion of relational and graph data models to answer relationship-heavy business questions natively.
- Enterprise Semantic Layer: a universal, governed metric layer that all NL2SQL (and BI) tools query against, ensuring a single version of truth across the organization.
- Reasoning Models: continued gains from models that allocate more inference-time compute to deliberate, verifiable multi-step SQL construction.
- Future of Data Analytics: a gradual shift from static dashboards toward conversational, agentic analytics as the primary mode of enterprise data consumption.

XV. CONCLUSION

NL2SQL has evolved from brittle, schema-specific rule engines into a mature, LLM-powered field capable of supporting genuinely conversational, multi-turn analytics over complex enterprise databases. The central technical insight of this evolution is that accuracy depends less on the raw capability of the underlying language model and more on the surrounding system: high-quality schema retrieval, an authoritative business glossary, rigorous SQL validation, and iterative self-correction. Architectures that combine retrieval-augmented generation, tool use, and multi-agent orchestration consistently outperform naive single-shot prompting, particularly on the large, messy, real-world schemas that characterize enterprise data estates.

Significant limitations remain. Schema linking on very large schemas, hallucinated joins and columns, ambiguous business terminology, and the security risks inherent in translating untrusted natural language into executable database queries all require continued research and, more importantly, disciplined engineering practice in production deployments. Evaluation methodology must also mature: academic benchmarks such as Spider, Spider 2.0, and BIRD provide useful signal but are not substitutes for enterprise-specific, continuously monitored evaluation sets.

Looking forward, the convergence of agentic AI, GraphRAG, semantic layers, and standardized tool-connectivity protocols such as MCP points toward a future in which conversational, self-correcting database copilots become a standard enterprise interface for data — reducing the dependency bottleneck on scarce SQL expertise and enabling faster, more democratized, data-driven decision-making. Organizations planning adoption are advised to invest early in metadata quality, a governed business glossary, and layered security controls, since these — far more than the choice of any single LLM — determine whether an NL2SQL deployment succeeds in production.

REFERENCES

[1] V. Zhong, C. Xiong, and R. Socher, "Seq2SQL: Generating structured queries from natural language using reinforcement learning," arXiv:1709.00103, 2017.

[2] X. Xu, C. Liu, and D. Song, "SQLNet: Generating structured queries from natural language without reinforcement learning," arXiv:1711.04436, 2017.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

[3] J. Guo et al., "Towards complex text-to-SQL in cross-domain database with intermediate representation," in Proc. ACL, 2019.
[4] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers," in Proc. ACL, 2020.
[5] X. V. Lin, R. Socher, and C. Xiong, "Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing," in Findings of EMNLP, 2020.
[6] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing incrementally for constrained auto-regressive decoding from language models," in Proc. EMNLP, 2021.
[7] T. Yu et al., "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in Proc. EMNLP, 2018.
[8] T. Yu et al., "SParC: Cross-domain semantic parsing in context," in Proc. ACL, 2019.
[9] T. Yu et al., "CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases," in Proc. EMNLP, 2019.
[10] V. Zhong, C. Xiong, and R. Socher, "WikiSQL: A large annotated semantic parsing and text-to-SQL dataset," 2017. [Online]. Available: https://github.com/salesforce/WikiSQL
[11] J. Li et al., "BIRD: A big bench for large-scale database grounded text-to-SQL evaluation," in Proc. NeurIPS Datasets and Benchmarks Track, 2023.
[12] Google Research, "Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows," arXiv:2411.07763, 2024.
[13] M. Pourreza and D. Rafiei, "DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction," in Proc. NeurIPS, 2023.
[14] OpenAI, "GPT-4 technical report," arXiv:2303.08774, 2023.
[15] OpenAI, "GPT-4o system card," 2024. [Online]. Available: https://openai.com/index/gpt-4o-system-card
[16] Anthropic, "The Claude 4 model family," 2025. [Online]. Available: https://www.anthropic.com/claude
[17] Anthropic, "Introducing the Model Context Protocol," 2024. [Online]. Available: https://www.anthropic.com/news/model-context-protocol
[18] Google DeepMind, "Gemini 2.5: Our most intelligent AI models," 2025. [Online]. Available: https://deepmind.google/technologies/gemini
[19] Meta AI, "The Llama 4 herd of models," 2025. [Online]. Available: https://ai.meta.com/llama
[20] DeepSeek-AI, "DeepSeek-V3 technical report," arXiv:2412.19437, 2024.
[21] Qwen Team, Alibaba Group, "Qwen2.5 technical report," arXiv:2412.15115, 2024.
[22] Mistral AI, "Mistral large and enterprise deployment guide," 2024. [Online]. Available: https://mistral.ai
[23] Microsoft Research, "Natural language interfaces to databases with large language models: A survey," 2024. [Online]. Available: https://www.microsoft.com/en-us/research
[24] N. Rajkumar, R. Li, and D. Bahdanau, "Evaluating the text-to-SQL capabilities of large language models," arXiv:2204.00498, 2022.
[25] Oracle Corporation, "Oracle AI Vector Search: Concepts and best practices," 2024. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse
[26] AWS, "Building natural language to SQL applications with Amazon Bedrock," 2024. [Online]. Available: https://aws.amazon.com/blogs/machine-learning
[27] Google Cloud, "Natural language to SQL with BigQuery and Gemini," 2025. [Online]. Available: https://cloud.google.com/bigquery/docs
[28] Snowflake, "Cortex Analyst: Self-service analytics powered by LLMs," 2024. [Online]. Available: https://www.snowflake.com/en/data-cloud/cortex
[29] Databricks, "Genie: Conversational analytics on the lakehouse," 2024. [Online]. Available: https://www.databricks.com/product/ai-bi/genie

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

[30] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. ICLR, 2023.
[31] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in Proc. NeurIPS, 2022.
[32] Y. Shinn, F. Cassano, B. Labash, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in Proc. NeurIPS, 2023.
[33] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. NeurIPS, 2020.
[34] Microsoft Research, "GraphRAG: Unlocking LLM discovery on narrative private data," 2024. [Online]. Available: https://microsoft.github.io/graphrag
[35] dbt Labs, "The analytics engineering guide to the semantic layer," 2024. [Online]. Available: https://www.getdbt.com/product/semantic-layer
[36] Nature Machine Intelligence, "Large language models for structured data understanding: Opportunities and risks," 2024.
[37] Springer, "Advances in natural language interfaces to databases," in Lecture Notes in Computer Science, vol. 14200, 2023.
[38] IEEE Transactions on Knowledge and Data Engineering, "A survey of deep learning approaches to text-to-SQL," vol. 36, no. 4, 2024.
[39] ACM Computing Surveys, "Natural language interfaces to databases: A survey and taxonomy," vol. 56, no. 3, 2023.
[40] R. Sun et al., "SQL-PaLM: Improved large language model adaptation for text-to-SQL," arXiv:2306.00739, 2023.
[41] X. Gao et al., "Text-to-SQL empowered by large language models: A benchmark evaluation," arXiv:2308.15363, 2023.