ChatGPT

Open-Source Text-to-SQL Projects with Schema-Augmented Retrieval

QueryWeaver (FalkorDB/GraphRAG Approach)

Link: FalkorDB/QueryWeaver on GitHub – an open-source Text-to-SQL tool that uses a **graph database as a semantic layer** between the LLM and the SQL database [1]. QueryWeaver represents the database schema (tables, columns, foreign keys) as a graph and **embeds schema elements** for similarity search. This lets it retrieve only the relevant schema subgraph for a given query, reducing prompt size while preserving relationships. Key technologies include OpenAI's GPT-4.1 as the default LLM and text-embedding-ada-002 for embeddings [2]. It uses the FalkorDB graph engine (a high-performance property graph) to store schema nodes and supports vector similarity search on those nodes.

- **Description:** QueryWeaver converts natural-language questions into SQL by leveraging "graph-powered" schema understanding [3]. The schema's tables/columns are nodes in a graph DB; relationships (like foreign keys) are edges. At query time, the tool finds which parts of the schema graph are most relevant (using embeddings for semantic similarity) and feeds only those to the LLM, improving precision and avoiding irrelevant context.
- **Key Technologies:** Uses OpenAI GPT-4 (or Azure OpenAI) for SQL generation and OpenAI's ada-002 embeddings for semantic search [2]. Schema graph storage and traversal is handled by FalkorDB (the project's own graph database) rather than a traditional vector store. This combination enables hybrid retrieval: vector similarity to identify candidate tables/columns, plus graph traversal to expand to related schema elements.
- **Maturity/Status:** QueryWeaver is under active development (v0.0.12 as of Oct 2025) with ~200+ GitHub stars. It provides a full stack (FastAPI backend and a React/TypeScript UI) including REST APIs for uploading schema graphs and querying them [4] [5]. It's relatively young but already usable via Docker, and supports multiple SQL dialects. It is maintained by the FalkorDB team as part of their "GraphRAG" ecosystem.
- **Limitations/Observations:** Since it uses its own graph DB, it doesn't natively run on Neo4j (though a converter exists to import Neo4j data to FalkorDB) [6]. Being new, documentation is still evolving. Also, it currently relies on OpenAI for embeddings and completions (no plug-and-play support for local LLMs out-of-the-box, though one could integrate them). Overall, QueryWeaver demonstrates the power of a graph+vector hybrid approach: using a knowledge graph of the schema to constrain and enrich the LLM's input, which can **reduce hallucinations and improve accuracy** in complex schemas [1].

Datrics Text2SQL (Vector Semantic Layer)

Link: datrics-ai/text2sql on GitHub – an open-source Text-to-SQL engine focused on **Retrieval-Augmented Generation (RAG)** for accuracy [7]. It doesn't use a graph database, but it does build a "semantic layer" of the schema by indexing documentation and examples in a **vector database**. The idea is to retrieve the most relevant schema info via embeddings, thereby trimming the context given to the LLM.

- **Description:** Datrics Text2SQL emphasizes using database documentation (schema descriptions, sample queries) and any provided examples to enrich the LLM's understanding [7]. Upon setup, it ingests the DB schema and generates a knowledge base of semantic embeddings (essentially an indexed FAQ of tables/columns and their meanings). At query time, the system finds which tables or columns are semantically related to the user's question by vector similarity, and includes only those in the prompt. This helps when the schema is large or when exact column names aren't explicitly mentioned in the question.
- **Key Technologies:** It uses **ChromaDB** as the vector store for embeddings and defaults to OpenAI's text-embedding models [8]. The LLM for SQL generation can be OpenAI GPT-4/GPT-3.5 or others configured via the LiteLLM router. The tool also provides a Streamlit UI for querying. Notably, it does not incorporate an explicit graph representation of the schema – instead it relies on the vector index and example-based re-ranking ("Smart Example Matching") to identify relevant schema elements [9].
- **Maturity/Status:** This project is relatively young (single-digit forks and <50 stars at last count [10]). It was released with an accompanying whitepaper describing its approach. It's usable with a local Dockerized database (it even ships a sample Postgres DB) and is intended to work out-of-the-box without model fine-tuning. The developers mention planned support for more LLMs and vector stores beyond OpenAI and Chroma [8].
- **Limitations/Observations:** Because it relies purely on vector similarity over text, Datrics Text2SQL might struggle if the schema terms in the user query are very different from the documentation wording (although the embedding should handle synonyms to some extent). It doesn't explicitly model relationships between tables – e.g. it may pull relevant tables by name similarity but might miss a join path unless that was documented or seen in an example. There's no Neo4j or graph integration here; however, one could envision extending its "semantic layer" to a graph DB to capture join relations. As is, it's a straightforward RAG-based solution: **vector search on schema/docs + LLM**, which the authors claim improves robustness on unseen tables [9].

Vanna – Text-to-SQL via RAG Library

Link: vanna-ai/vanna on GitHub – "Chat with your SQL database" is the tagline, and Vanna is a popular MIT-licensed library (20k+ stars) for Text-to-SQL that uses **retrieval-augmented generation**, pulling in schema info at runtime [11]. It is a lightweight, developer-centric component rather than a full end-user application. Vanna's approach is to use vector similarity to find relevant schema snippets (table and column definitions, or even past queries) and include those in the LLM prompt. This avoids sending the entire schema for each query.

- **Description:** Vanna essentially provides an API where you give it a natural language question and it returns an SQL query (and optionally the result). Under the hood, it ensures the LLM has the needed schema context by retrieving it first. For example, if the question asks about "customers in Europe", Vanna's pipeline will fetch the definition of the `customers` table (and related tables like `orders` if needed) from its indexed schema store, and supply those to the LLM as context [11]. It can also ingest database documentation to help the model disambiguate terms. This RAG approach helps Vanna achieve more accurate mapping of NL to SQL, as noted in an enterprise comparison: "Vanna improves accuracy through RAG, pulling schema snippets and documentation at runtime" [11].

- **Key Technologies:** Vanna is Python-based and built to integrate with LangChain-style workflows. It uses vector embeddings (e.g. OpenAI or SentenceTransformers) and can work with popular vector stores (FAISS, Chroma, etc.) for similarity search. The LLM backend is configurable – developers often use OpenAI GPT-3.5/4, but it could be any that LangChain supports. Vanna focuses on **schema retrieval and prompt engineering** rather than fine-tuning. It also supports multiple SQL dialects by adjusting the few-shot examples it provides.
- **Maturity/Status:** Very active – it started in mid-2023 and quickly gained traction. Many users integrate Vanna into custom apps or data notebooks. It's considered stable for its core use (there are community discussions and frequent updates). However, it's a "bring your own UI" library; out of the box it's just an API and CLI. It's great for developers embedding Text2SQL into tools, but not a turnkey BI solution (which is by design [12] [13]).
- **Limitations/Observations:** Vanna's reliance on prompt-based retrieval means it doesn't enforce a formal semantic layer or global view of enterprise metrics. As the Wren AI team pointed out, the same term might be interpreted differently across queries if not consistently defined [14]. Maintaining the quality of the retrieved snippets (and curating good few-shot examples) is up to the developer. Also, it currently doesn't use a graph database – the relationships are inferred from foreign keys present in the schema text, not via graph traversal. That said, its RAG approach could theoretically be extended by plugging in Neo4j: one could store the schema graph in Neo4j and use Neo4j's vector index for similarity search on node properties [15], then traverse to get related tables. This isn't in the library by default, but is a logical extension to combine Vanna's strengths with a graph-backed schema store.

Wren AI – Semantic Layer & Vector Retrieval

Link: Canner/WrenAI on GitHub – an open-source **"GenBI" (Generative Business Intelligence) platform** that includes Text-to-SQL, charts, and insights. Wren AI is notable for combining an **explicit semantic model layer** with retrieval techniques. Instead of just vector-matching schema text, Wren lets you define business concepts (metrics, dimensions, relationships) in a metadata layer (MDL), which acts like a knowledge graph over your data warehouse schema [16] [17]. During querying, Wren uses a vector database to fetch relevant context (similar to others), but the context can include these semantic definitions and not just raw schema text [18]. This reduces LLM confusion and ensures consistency for enterprise use-cases.

- **Description:** Wren AI's focus is enterprise readiness – it aims to prevent the LLM from hallucinating or misinterpreting schema by providing governed context. When a user asks a question, Wren's pipeline will do a few things: (1) use the **"Wren Engine"** to retrieve relevant schema metadata (tables involved, join paths, any predefined metric logic) from its internal store, (2) use the **"Wren AI Service"** to embed and retrieve any relevant documentation or past queries via a vector DB [18], and then (3) feed the combined context to the LLM to get an SQL query. The answer is presented with explanations, and can include automatically generated charts or text summaries (hence "GenBI"). Crucially, Wren enriches the prompt with semantic info – e.g., if "revenue" is defined in the semantic layer as a specific calculation, the LLM is given that definition to avoid ambiguity [19] [20].
- **Key Technologies:** Wren is a full-stack platform. It supports many LLMs (OpenAI, Azure, Anthropic, local models via Ollama, etc.) through a modular configuration [21]. For vector search, Wren uses an integrated **Qdrant** (by default) or other vector DB to store embeddings of schema texts and example queries. The **graph aspect** is in the semantic modeling: Wren's UI lets you define relationships between tables and the meaning of fields (similar to Looker's LookML or a star-schema semantic layer). This effectively builds a knowledge graph of the schema (though it's stored internally, not in Neo4j format). The system then uses those relations during retrieval – it knows, for instance, which tables are linked and can pull the correct join keys.

- **Maturity/Status:** WrenAI is fairly mature for an open project – it has ~12k stars and is backed by Canner's team (with a commercial cloud offering, but the core is AGPL open-source). It's actively maintained; as of early 2025 it supports multiple SQL backends (DuckDB, Postgres, BigQuery, etc.) [22] . Users can self-host it and get a web UI for constructing the semantic layer and running queries. Because it's a larger system, deploying Wren is a bit more involved (several services: UI, service, engine as noted in its architecture [23]), but the docs provide quick-start Docker setups.
- **Limitations/Observations:** WrenAI's strength is in enterprises with well-defined metrics – it ensures that, say, "customer churn rate" is computed consistently every time. The flip side is that it requires upfront modeling: someone must input those semantic definitions. This is both a feature and an overhead (whereas a simpler RAG tool like Vanna will work immediately on any schema by reading the raw schema text). Wren's retrieval of context is hybrid – it uses the structured semantic layer plus unstructured vector search over descriptions. It does not currently integrate Neo4j; the schema/semantic info is stored in its own metadata store and the vector DB handles similarity search. However, this design is aligned with the idea of graph-based retrieval. In fact, Wren's approach of **"semantics to data schema"** (i.e., mapping business terms to the schema) is similar in spirit to using a knowledge graph to augment the LLM [24] . The project demonstrates that carefully curated schema knowledge + embeddings can significantly boost Text-to-SQL accuracy by **grounding the LLM in real schema context** (less guesswork, more exact matches).

DB-GPT (Agent Framework with Knowledge Graph module)

Link: eosphoros-ai/DB-GPT on GitHub – an ambitious framework that goes beyond Text-to-SQL, featuring an **agentic workflow** and even a knowledge graph component. DB-GPT can be seen as a toolkit to build AI data applications (it supports RAG, fine-tuning, multi-modal agents, etc.), with Text-to-SQL as one of its showcased capabilities [25] [26] . For our purposes, the relevant part is how it handles large schemas: it includes a **schema retrieval and knowledge graph step** to aid SQL generation. Specifically, DB-GPT can ingest database schema information into a graph database and use that for retrieval and reasoning – however, the current open-source implementation only officially supports the TuGraph graph DB for this, not Neo4j [27] .

- **Description:** In a typical DB-GPT workflow for Text-to-SQL, multiple specialized agents are involved. One agent might perform a vector search on a corpus of schema text or past queries; another agent (the "knowledge graph" agent) can query the graph database that holds facts about the schema or the data itself. For example, if a user question is complex and involves understanding relationships, the system could translate part of the question into a graph query to find how tables are linked (this is analogous to a Text2Cypher step for schema exploration). The knowledge graph in DB-GPT is used to store either the schema structure or a semantic abstraction of the data. In practice, a user report noted that this feature currently works with **TuGraph** (a graph DB from Ant Group) and wasn't readily compatible with Neo4j [28] . The retrieved schema info and any relevant nodes from the graph are then provided to the main LLM agent which formulates the SQL.
- **Key Technologies:** DB-GPT is designed to be model-agnostic, supporting many open LLMs (Llama2, Falcon, ChatGLM, etc.) and OpenAI APIs. It leverages **vector databases** for retrieval (users have plugged in Milvus, Chroma, etc. for different tasks) as well as graph databases (TuGraph for now). It has a custom agent orchestration language (AWEL) to script how these components interact [29] . For Text-to-SQL specifically, there is a sub-project called DB-GPT-Hub that provides fine-tuning scripts and workflows aiming to improve accuracy on benchmarks (they report ~82.5% on Spider after fine-tuning) [30] .
- **Maturity/Status:** DB-GPT is an evolving project with a large community (11k+ stars). It's more of a research platform than a plug-and-play solution – setting it up can be complex, and using all

features (like the knowledge graph) might require customization. The multi-agent approach is powerful but can be fragile. Documentation has been noted as a weak point [31], though it's improving. As of late 2024, versions ~0.7 exist, indicating active iteration.
- **Limitations/Observations:** The idea of combining vector search with a graph-based schema store is present, but in practice not as streamlined in DB-GPT due to the TuGraph dependency. It shows a path forward: e.g., one could load a Neo4j database with schema info and adapt DB-GPT's agent to use Neo4j's Cypher queries for schema retrieval. Out-of-the-box, however, this is not provided. Users also observed that while DB-GPT's agents can generate SQL, handling follow-up questions in a conversation or producing visualizations was less consistent [27]. In summary, DB-GPT is a cutting-edge toolkit that **partially implements** the described approach (vector RAG + graph schema reasoning). It validates that enriching LLMs with a structured schema graph can help on complex queries, but it may require further development or integration (e.g., with Neo4j's GraphRAG tools) to fully realize a robust Neo4j+LLM Text-to-SQL system.

Closing Notes: Partial Solutions and Extensibility

The projects above cover various angles of the vector+schema retrieval approach. In cases where an exact match to "Neo4j as schema store" was not found, we highlighted close matches and how they could be extended. For instance, **GraphRAG libraries** (like Neo4j's official GraphRAG Python package) provide the building blocks to use Neo4j with LLMs for retrieval [32]. One could imagine using Neo4j to ingest a relational schema as a graph of nodes (tables/columns) and relationships, then use Neo4j's built-in **vector indexing** for similarity search on node embeddings [15]. This would achieve the same goal of schema-based context filtering. While no single turnkey solution was found that exactly uses Neo4j in this manner for Text-to-SQL, the ingredients are all present in open source:

- **LangChain/LlamaIndex integrations:** Both frameworks allow combining vector searches with structured data lookups. A developer can use LlamaIndex to do a first-pass embedding search for relevant tables, then use a graph (or SQL metadata query) to pull connected columns. There are examples of "schema-aware retrievers" in research (e.g., retrieval based on schema content similarity) [33]. These could be paired with Neo4j.
- **Research prototypes:** Recent papers like **CRED-SQL** propose clustering and retrieving schema pieces for large databases [34], and **Enhancing Text2SQL with Schema Filtering** (Neo4j's team for Text2Cypher) explores using schema knowledge to narrow down query interpretation [35]. While code for these might not be released, their methods align with the open-source projects above.

In summary, the landscape is quickly evolving. Tools like QueryWeaver and WrenAI show that **hybrid retrieval (vectors + graph relationships) can significantly improve Text-to-SQL** by focusing the LLM on the right context. And even where Neo4j isn't explicitly used, the concept of a graph-based schema layer is proving its value. It's likely only a matter of time until more projects directly combine Neo4j with Text-to-SQL – possibly by extending the ones above or through new contributions – to fully leverage graph traversal alongside semantic search for precise SQL generation.

Sources: The information above was synthesized from project documentation and developer discussions, including official READMEs and blogs for QueryWeaver [1] [3], Datrics Text2SQL [8], Vanna [11], Wren AI [18], and DB-GPT [27], as well as comparative analyses [14]. These references illustrate how each system implements schema retrieval and highlight their use of embeddings and (in some cases) graph structures to augment LLM-based SQL generation.

---

[1] Building an open-source text2sql (with a graph semantic layer) : r/SQL
https://www.reddit.com/r/SQL/comments/1n7et1u/building_an_opensource_text2sql_with_a_graph/

[2] [3] [4] [5] GitHub - FalkorDB/QueryWeaver: An open-source Text2SQL tool that transforms natural language into SQL using graph-powered schema understanding. Ask your database questions in plain English, QueryWeaver handles the weaving.
https://github.com/FalkorDB/QueryWeaver

[6] FalkorDB · GitHub
https://github.com/FalkorDB

[7] [8] [9] [10] GitHub - datrics-ai/text2sql: Text2SQL Engine with advanced RAG
https://github.com/datrics-ai/text2sql

[11] [12] [13] [14] [19] [20] Wren AI vs. Vanna: The Enterprise Guide to Choosing a Text-to-SQL Solution
https://www.getwren.ai/post/wren-ai-vs-vanna-the-enterprise-guide-to-choosing-a-text-to-sql-solution

[15] FalkorDB vs Neo4j: Choosing the Right Graph Database for AI
https://www.falkordb.com/blog/falkordb-vs-neo4j-for-ai-applications/

[16] [17] [21] GitHub - Canner/WrenAI: ⚡ GenBI (Generative BI) queries any database in natural language, generates accurate SQL (Text-to-SQL), charts (Text-to-Chart), and AI-powered insights in seconds.
https://github.com/Canner/WrenAI

[18] [22] [23] [24] WrenAI: Make your database RAG-ready. : r/SQL
https://www.reddit.com/r/SQL/comments/1cksm7u/wrenai_make_your_database_ragready/

[25] [26] [29] GitHub - eosphoros-ai/DB-GPT: AI Native Data App Development framework with AWEL(Agentic Workflow Expression Language) and Agents
https://github.com/eosphoros-ai/DB-GPT

[27] [28] [30] [31] Top AI SQL Tools: Features, Reviews, and Key Insights
https://scopicsoftware.com/blog/ai-sql-tools/

[32] GitHub - neo4j/neo4j-graphrag-python: Neo4j GraphRAG for Python
https://github.com/neo4j/neo4j-graphrag-python

[33] How to Build a Production-Grade Text2SQL Engine | HackerNoon
https://hackernoon.com/how-to-build-a-production-grade-text2sql-engine

[34] CRED-SQL: Enhancing Real-world Large Scale Database Text-to ...
https://arxiv.org/html/2508.12769v1

[35] Enhancing Text2Cypher with Schema Filtering - arXiv
https://arxiv.org/html/2505.05118v1