Multi-Agent Natural Language to SQL System - Full Design (Agents A to D)

Methods

Overview

This system converts **natural language questions** into **SQL queries** over one of many databases (e.g., Spider/real-world datasets) using a **multi-agent architecture**:

- **Agent A** – *Schema Intelligence Agent*: Selects top-k most relevant schemas/databases
- **Agent B** – *Query Understanding Agent*: Extracts entities, intents, and constraints
- **Agent C** – *SQL Generation Agent*: Converts the user question and schema into SQL
- **Agent D** – *Execution & Response Agent*: Runs SQL, fetches and formats result

1. Agent A – **Schema Intelligence Agent (Database Selector)**

Task: Given a natural language question and a list of database names, select the most relevant one(s).

Tool/Model	Type/Purpose	Pros	Cons
OpenAI Embeddings (e.g., text-embedding-3-small) + FAISS	Semantic search	Very accurate semantic matching Easy to integrate Fast filtering on large corpus	Requires embedding generation beforehand Cloud-based (API usage costs)
GPT-4, Claude	LLM Reranker	Rerank top-k matches using deep reasoning	API-based

Each database schema is converted into a concise, human-readable description and embedded using SBERT, with the embeddings stored for retrieval. At runtime, the user query is embedded using the same model, and cosine similarity is computed against the stored schema embeddings to identify the most relevant matches. Optionally, the top-k results can be reranked using an LLM for enhanced semantic alignment.

2. Agent B – **Query Understanding Agent (Fetch Table & Column)**

Task: Given the selected database schema and the user query, identify which tables/columns are relevant.

Tool/Model	Type/Purpose	Pros	Cons
LLM with Schema-aware prompt (GPT-4, Claude)	Generative reasoning	Handles complex schema relations	Costly with long schema Token limits

		Can explain reasoning	
SQLNet / TypeSQL	Text-to-SQL baselines	Trained to understand tables Schema-aware	Weak generalization Needs fine-tuning for new schemas

Agent B (Query Understanding Agent) takes the top-k databases from Agent A and deeply analyses the user's question to determine which tables and columns within those databases are relevant. It uses an LLM or fine-tuned semantic parser to extract entities, actions, and constraints from the question, then maps these elements to schema items using semantic similarity or embedding lookups. The output is a reduced, "focused schema" containing only the relevant tables, columns, and join conditions. This step ensures that when Agent C generates the SQL, it works with a minimal, context-specific schema—improving accuracy and lowering token costs.

3. Agent C – **SQL Generation Agent**

Task: Transform Agent B's structured parse + the selected schema from Agent A into a valid, efficient SQL query.

Tool/Model	Type/Purpose	Pros	Cons
GPT-4 / Claude 3 / Gemini Pro	LLM	Can generate SQL & human-readable explanation in one go Handles nested/complex queries	Expensive May produce incorrect SQL unless prompted carefully
SQLCoder (open-source)	Text-to-SQL	Optimized for SQL generation Good base model for fine-tuning	No explanations Requires setup

Agent C (SQL Generation Agent) takes the focused schema from Agent B along with the original user question and generates the precise SQL query needed to retrieve the answer. It typically uses a Text-to-SQL model either a fine-tuned transformer like T5-SQL / SQL-PaLM or an API-based LLM like GPT-4 to translate natural language into SQL syntax while respecting join conditions, filters, and constraints. The agent ensures the query adheres to the database's dialect and structure. By operating only on the pruned schema, Agent C reduces hallucinations, avoids irrelevant joins, and outputs a ready-to-execute SQL query tailored to the user's request.

4. Agent D – **Execution & Response Agent**

Task: Safely execute the SQL against the chosen DB, validate results, and format the final answer (plus optional natural language explanation).

Tool/Model	Type	Pros	Cons
SQLAlchemy (Python ORM)	SQL execution	Clean execution Schema mapping support Python-native	Requires handling query errors

Agent D (Answer Execution & Explanation Agent) takes the SQL query produced by Agent C, executes it against the actual database, and formats the results for the user. Beyond just returning raw data, Agent D can use models like PandasAI, LangChain's SQLDatabaseChain, or an LLM such as GPT-4 to generate a natural language explanation of the answer, summarize patterns, or highlight key points. The focus is on making the response clear, accurate, and interpretable showing not only *what* the answer is, but also *why* it matches the query.

Overall Methods Summary:

Task 1: Schema Intelligence and Database Selection (Agent A)

Natural language queries must first be mapped to the most relevant database(s) from a large collection. This step is crucial to reduce search space, lower processing costs, and ensure contextual relevance.
We will develop a schema intelligence module that:

- Converts each database schema into concise, human-readable descriptions
- Generates vector embeddings using SBERT or OpenAI text-embedding-3-small
- Stores embeddings in FAISS for fast semantic search
- Embeds the incoming user query and retrieves top-k most similar schemas via cosine similarity
- Optionally re-ranks top-k results using an LLM (e.g., GPT-4, Claude) for deeper semantic alignment

This module ensures downstream agents operate only on the most relevant databases, reducing token usage for LLM-based processing.

Task 2: Query Understanding and Schema Pruning (Agent B)

- Even after database selection, the schema may contain dozens of tables and hundreds of columns, many irrelevant to the current query.
  The Query Understanding Agent will:
- Take the top-k databases from Task 1
- Parse the user question to identify entities, actions, and constraints
- Map these elements to tables and columns using either:
- LLMs with schema-aware prompts (e.g., GPT-4, Claude)
- Fine-tuned Text-to-SQL parsers such as SQLNet or TypeSQL

- Use embedding similarity or classifier lookup to match query terms to schema elements
- Output a focused schema containing only relevant tables, columns, and join paths

This step reduces complexity for SQL generation, improves accuracy, and lowers processing costs by limiting schema size.

Task 3: SQL Generation (Agent C)

With a pruned schema and a fully parsed understanding of the query, this agent generates the executable SQL.
The SQL Generation Agent will:

- Take the reduced schema and query from Task 2
- Generate valid SQL using either:
    - API-based LLMs (e.g., GPT-4, Claude 3, Gemini Pro) for flexible, complex query handling
    - Open-source models (e.g., SQLCoder, T5-SQL, SQL-PaLM) for fine-tuned offline execution
- Ensure correct join conditions, filters, aggregations, and database-specific syntax
- Produce optional human-readable explanations of the SQL for transparency

By operating only on the focused schema, the agent avoids irrelevant joins and reduces the risk of hallucinations in query generation.

Task 4: Execution, Validation, and Response Generation (Agent D)

This agent executes the generated SQL, validates outputs, and formats results for the end-user.
The Execution & Response Agent will:

- Use SQLAlchemy or a similar ORM to execute the query against the chosen database
- Handle query errors and validate results for completeness and correctness
- Format outputs as tables, charts, or summaries depending on query intent
- Optionally use LLMs (e.g., GPT-4, LangChain SQLDatabaseChain, PandasAI) to provide natural-language explanations or highlight insights

This stage ensures that users receive clear, accurate, and interpretable results along with context for how the answer was derived.