

Building a Unified Language Interface for Hybrid Data Access

Lessons Learned

Matthew Schwartz

Contents

The Challenge	1
My Approach: A Hybrid System	1
Key Technical Insights	2
What Worked Well	2
Challenges and Limitations	3
Evolution of the Approach	3
Looking Forward: Emerging Techniques	3
Conclusion	3
Why move to neuro-symbolic routing?	4
References	4

Over the past few months, I have been working on an exciting project to create a unified natural-language interface for accessing both structured and unstructured data. In modern organizations, valuable information lives in multiple formats – from relational databases and data warehouses to document repositories and websites. End users shouldn’t need to know SQL for databases, use a separate vector-search tool for documents, or switch between multiple interfaces just to find answers. My goal was to eliminate this friction by using language models as a single entry point to diverse data sources.

The Challenge

Today, data is siloed across formats: a sales figure might reside in a SQL database, while a policy answer might be buried in PDFs. Traditionally, one must write database queries for structured data and perform keyword or vector searches for unstructured text. This fragmentation is time consuming, data professionals spend over half their time just on finding, cleaning, and organizing data before any analysis [14]. A unified interface that lets users simply ask questions in natural language and fetch relevant answers (regardless of source) can democratize data exploration. The core challenge is grounding free-form questions in the right data context. The system must interpret the user’s intent and determine *where* to get the answer: Does the question require an SQL query over structured data? A passage lookup in documents? Or both? Achieving this reliably, without user intervention, is a hard problem in natural-language understanding and context interpretation.

My Approach: A Hybrid System

To address these challenges, I built a hybrid pipeline that combines multiple data-access techniques behind a single natural-language interface. Given any user question, the system routes it to one or more of the following components:

1. **Retrieval-Augmented Generation (RAG)** for unstructured data - I leverage a vector database of documents and an LLM-based generator. The user’s query is converted into an embedding and used to retrieve relevant text passages (e.g., from PDFs, wikis, webpages). These passages are

then provided as additional context to the language model, which generates an answer in natural language. This RAG approach grounds the LLM's output on external knowledge, ensuring up-to-date and factual answers from documents rather than the model's parametric memory [1]. It's essentially an "open-book" mode for the LLM, greatly reducing hallucinations and allowing users to see source citations for transparency.

2. **Pattern-based SQL queries** for common structured questions – I created a library of predefined SQL query templates to handle frequent question types reliably. For example, a question like *"What were the total sales by region last month?"* matches a template that maps to a SUM aggregation grouped by region. These templates are essentially hand-crafted patterns that cover a wide range of typical analytic queries (counts, sums, top-N, etc.). If the user's question fits one of the known patterns (determined via simple heuristics or classification), the system fills in the template with the relevant table and column names and executes it. This pattern-first approach ensures correctness and speed for routine queries – the SQL is guaranteed to be well-formed and tested, with no risk of model hallucination. It also reduces load on the LLM by handling easy cases with rule-based logic.
3. **Dynamic SQL generation** for novel or complex queries – If a question doesn't match any known template, I fall back to an LLM-driven solution: have the language model generate an SQL query on the fly. The LLM is provided with context about the target database schema (table and column names, relationships) and the user's question, and it produces a SQL query that could answer the question. I then execute this SQL against the database and return the results. This component gives the system flexibility; it can handle arbitrary analytical questions, even those I didn't anticipate. However, it introduces the challenge of ensuring the LLM's SQL is correct and *safe to run*. I built safeguards such as syntax validators, schema checks, and timeouts on execution. In practice, providing the database schema as prompt context significantly improves the quality of LLM-generated SQL. The model is far less likely to produce invalid SQL when it knows what tables and fields exist, as demonstrated by Microsoft's RAT-SQL approach [2].

How it works end-to-end: A user asks a question in plain English. The system's router component first classifies the query or uses heuristics to decide which backend(s) to invoke. For example, a question containing the word "document" or requiring a descriptive answer might trigger the RAG pipeline; a question with words like "how many" or "average" might indicate a structured-data query. In ambiguous cases, the router can even run both: use RAG to pull possible textual answers and simultaneously try an SQL generation, then decide which result to present. The user ultimately sees a single, consolidated answer – which might include a direct answer with supporting text evidence, a table/chart result from the database, or a combination along with citations or indicators of sources.

Key Technical Insights

What Worked Well

1. **Schema-informed SQL generation** – Providing database schema context to the LLM dramatically improved the quality and accuracy of generated SQL. In early trials, the model would sometimes ask for non-existent tables or fields. Once I began injecting a description of the relevant database including table names, column names, and primary-key relationships those errors dropped significantly. Similar improvements are reported in the literature, including RAT-SQL [2] and the more recent DBCopilot [3]. With schema-aware generation and post-validation (checking each query against the schema), the success rate of LLM-generated SQL improved substantially.
2. **Hybrid pattern-first, generation-second approach** – I achieved the best balance of reliability and flexibility by using pattern-based solutions as a first resort and falling back to LLM generation only for the outliers. This strategy embodies neuro-symbolic integration concepts [4]. In practice, it reduced execution errors without sacrificing the ability to answer novel questions.
3. **Automatic visualization selection** – For questions that call for data summaries or comparisons, presenting the result as a chart can greatly enhance understanding. I implemented a module that automatically suggests an appropriate visualization based on the query intent and the shape of the data, inspired by VizML [5] and DataTone [11]. Automatically generating the visualization (I output Plotly or

Vega-Lite specs) closed the loop in user experience – the user asks a question in words and gets an insightful visual when appropriate.

Challenges and Limitations

1. Hallucination in SQL generation – Even with recent advances, LLMs can still emit SQL that is syntactically or semantically wrong. I began with basic prompt instructions and simple post-generation filters, then introduced stronger safeguards—schema validation, constrained decoding with PICARD [6], and sandboxed execution—to catch errors. Yet fully eliminating hallucinations remains an open research challenge.

2. Semantic routing complexity – Deciding whether a question should be answered from structured or unstructured data (or both) proved more complex than anticipated. Hybrid dense-plus-sparse retrieval research [7] informed my current approach, which sometimes invokes both backends and merges the results.

3. Result-formatting inconsistency – Because the pipeline delivers answers as text, tables, numbers, or charts, I needed a consistent presentation. I use the LLM to wrap each result with a plain-language explanation so users never receive a bare number or orphaned graphic. This approach takes cues from Nielsen Norman Group’s UX guidelines [8] and related research on multimodal clarity.

Evolution of the Approach

The system evolved through several iterations:

1. **Initial pattern-matching (rule-based)** – Fast and deterministic but limited in coverage.
2. **Semantic routing with embeddings** – Introduced dual-mode querying and greater flexibility.
3. **Dynamic SQL generation with LLMs** – Greatly expanded coverage and capability.
4. **Robust error handling and fallback** – Added resilience via validation and intelligent retries.
5. **Automatic visualization** - Added logic that selects suitable chart types and wraps every result in plain-language captions, creating a consistent, story first presentation layer.

Looking Forward: Emerging Techniques

The field of natural-language interfaces for data is advancing quickly. Here are several techniques I believe could further improve systems like mine:

1. **Reasoning-optimized LLMs** – Leveraging models that natively expose chain-of-thought or other structured-reasoning traces improves logical consistency and reduces errors on complex queries without heavy few-shot prompt engineering [9].
2. **Tool-augmented LLMs** – Giving the model structured access to tools (e.g., QueryDB, plotting libraries) will blur the line between “generation” and “execution” and further reduce hallucinations [10].
3. **Multi-modal and user-in-the-loop interfaces** – Combining language input with GUI widgets and follow-up questions (à la DataTone) can resolve ambiguity and empower non-technical users [11].

As of 2025, cutting-edge approaches like UniK-QA [12] and GPT-4 with advanced prompting have pushed execution accuracy on complex text-to-SQL benchmarks (Spider) to ~85 percent [13]. Commercial BI platforms are already embedding these techniques in their “Analytics Copilot” features.

Conclusion

Building a unified natural-language interface for hybrid data access is already unlocking value, but the journey is far from over. My next milestone is to upgrade the routing layer from today’s embedding plus keyword heuristics to a **true neuro-symbolic router** combining declarative rules with neural confidence scores.

Why move to neuro-symbolic routing?

- **Higher routing precision & fewer edge-case failures** – symbolic rules capture domain knowledge that embeddings alone miss, while neural scoring handles fuzzy language.
- **Explainability & audit trails** – every dispatch decision is traceable through a rule engine log, simplifying security reviews and compliance reporting.
- **Policy-aware safeguards** – rules can enforce data-governance constraints (e.g., block PII queries from hitting less-restricted RAG paths) *before* any LLM call.
- **Continuous learning** – logs of mis-routed queries feed back to automatically refine both the rule set and neural thresholds, reducing manual tuning.
- **Faster onboarding of new data silos** – adding a backend becomes a matter of writing a few rules rather than curating dozens of example embeddings.

The state of the art is evolving rapidly, yet open problems remain (fully eliminating hallucinations, seamless modality integration, richer interactivity). My experience demonstrates that it is already possible to deliver real value by combining symbolic reliability with generative power. As models become more capable and tool-aware, I expect natural language to become the **universal interface** for data access.

References

- [1] IBM Research (2023). *What is Retrieval-Augmented Generation?* IBM Research Blog.
<https://research.ibm.com/blog/retrieval-augmented-generation-RAG>
- [2] Wang, B. Y. et al. (2020). *RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers*. ACL 2020.
<https://arxiv.org/abs/1911.04942>
- [3] Li, X. et al. (2023). *DBCopilot: Scaling Natural-Language Querying to Massive Databases*. arXiv 2312.03463.
<https://arxiv.org/abs/2312.03463>
- [4] Marcus, G. (2020). *The Next Decade in AI: Four Steps Toward Robust Artificial Intelligence*. arXiv 2002.06177.
<https://arxiv.org/abs/2002.06177>
- [5] Hu, K. Z. et al. (2019). *VizML: A Machine-Learning Approach to Visualization Recommendation*. CHI 2019.
<https://arxiv.org/abs/1808.04819>
- [6] Scholak, T. et al. (2021). *PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding*. EMNLP 2021.
<https://arxiv.org/abs/2109.05093>
- [7] Pinecone (2022). *Keyword-Aware Semantic Search with Hybrid Indexes*. Engineering Blog.
<https://www.pinecone.io/blog/hybrid-search/>
- [8] Nielsen Norman Group (n.d.). *Choosing Chart Types: Consider Context*.
<https://www.nngroup.com/articles/choosing-chart-types/>
- [9] Wei, J. et al. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv 2201.11903.
<https://arxiv.org/abs/2201.11903>
- [10] Schick, T. et al. (2023). *Toolformer: Language Models Can Teach Themselves to Use Tools*. arXiv 2302.04761.
<https://arxiv.org/abs/2302.04761>
- [11] Liu, Z. et al. (2014). *DataTone: Managing Ambiguity in Natural-Language Interfaces for Data Visualization*. UIST 2014.
<https://dl.acm.org/doi/10.1145/2807442.2807478>

- [12] Oguz, B. et al. (2022). *UniK-QA: Unified Representations of Structured and Unstructured Knowledge for Open-Domain QA*. NAACL 2022.
<https://arxiv.org/abs/2012.14610>
- [13] Pourreza, H. (2023). *Text-to-SQL Benchmarks and the Current State of the Art*. Medium (Dataherald).
<https://medium.com/dataherald/text-to-sql-benchmarks-and-the-current-state-of-the-art-63dd3b3943fe>
- [14] Anaconda (2020). *State of Data Science 2020* (White Paper).
<https://www.anaconda.com/resources/whitepaper/state-of-data-science-2020>