Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Self-Optimizing Enterprise RAG+SQL Knowledge Agents: Unified Retrieval, Schema-Aware Query Translation, Event Graph Reasoning, and Adaptive Model Routing

Harshil Lodhiya

lodhiyaharshil@gmail.com

SlicedHealth, Inc. https://orcid.org/0009-0005-5798-9176

Research Article

Keywords: enterprise AI, retrieval-augmented generation, text-to-SQL, knowledge graphs, schema-aware query translation, adaptive model routing, feedback control, LLM middleware

Posted Date: August 25th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10798738/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Self-Optimizing Enterprise RAG+SQL Knowledge Agents: Unified Retrieval, Schema-Aware Query Translation, Event Graph Reasoning, and Adaptive Model Routing

Harshil Lodhiya
Sliced Health
hlodhiya@slicedhealth.com

August 2026

Abstract

Enterprise knowledge is fragmented across relational databases, PDFs, incident reports, tickets, logs, spreadsheets, policy documents, and operational runbooks. Retrieval-augmented generation (RAG) systems help with unstructured text, while text-to-SQL systems help with structured databases, but production workflows frequently require both forms of reasoning in the same answer. A finance analyst may ask for contract terms and revenue aggregates; a compliance officer may ask for a policy exception and the exact affected accounts; a support lead may ask for the causal sequence behind an incident and the customer records touched by it. This paper introduces the Self-Optimizing Enterprise RAG+SQL Knowledge Agent, a middleware architecture that unifies schema-aware SQL translation, document retrieval, event-schema knowledge graphs, execution validation, and adaptive model routing. The proposed agent decomposes enterprise questions into typed evidence plans, retrieves from hybrid vector and graph indexes, prunes relational schemas through schema relationship graphs, executes SQL in bounded read-only sandboxes, composes multi-source evidence, and continuously updates routing and retrieval policies from observed quality, latency, cost, and validation feedback. The design extends prior work on schema-aware query translation, EventForge hybrid RAG+KG storage, LLM-Advisor dynamic model routing, and feedback-control allocation. We provide a detailed architecture, objective function, routing policy, validation protocol, benchmark design, implementation blueprint, and risk analysis. The central claim is that enterprise question answering should be treated as a self-optimizing control problem over heterogeneous knowledge tools, not as a single prompt over retrieved text.

Keywords: enterprise AI; retrieval-augmented generation; text-to-SQL; knowledge graphs; schema-aware query translation; adaptive model routing; feedback control; LLM middleware.

1 Introduction

Modern enterprises do not store knowledge in one form. Business facts live in relational tables. Procedures live in PDFs and internal wikis. Incidents live in logs, tickets, chat transcripts, and postmortems. Regulatory obligations live in policy documents. Customer state is spread across operational databases and CRM systems. A realistic enterprise question often crosses several of these surfaces:

- "Which customers affected by last week's outage also have premium support language in their contracts?"

- "What policy clause governs this exception, and how many active accounts currently violate it?"
- "Summarize the chain of events behind this failed payment batch and list the database rows that need remediation."
- "Which vendors mentioned in our security documents also appear in current production access tables?"

These questions cannot be answered reliably by text-only RAG or by text-to-SQL alone. Text-only RAG can retrieve a policy or a postmortem, but it cannot safely aggregate current relational state. Text-to-SQL can answer structured questions, but it cannot infer causal narratives from incident reports or interpret policy language across documents. A unified agent must reason over both unstructured and structured evidence, decide which tools to use, validate intermediate outputs, and optimize its own routing behavior over time.

Prior work provides important pieces. Schema-aware query translation introduced Inference-from-RDBMS, which converts database catalogs into schema relationship graphs, prunes irrelevant schema context, and validates generated SQL before execution [4]. EventForge reframed event schema induction as typed DSPy programs and stored induced event graphs together with dense retrieval artifacts in PostgreSQL/pgvector [6]. LLM-Advisor introduced dynamic model selection across heterogeneous LLM pools under cost, latency, and quality constraints [10]. Feedback-control routing treated online LLM allocation as a closed-loop control problem with persistent excitation and bounded allocation updates [12]. This paper combines these directions into one enterprise knowledge-agent architecture.

The Self-Optimizing Enterprise RAG+SQL Knowledge Agent, abbreviated SERQ-Agent, is a middleware layer between users and enterprise knowledge systems. The agent accepts natural-language questions, decomposes them into typed evidence tasks, retrieves document passages, traverses event graphs, prunes relational schemas, generates and validates SQL, fuses evidence, and learns from feedback. It is self-optimizing in a specific operational sense: the agent continuously updates its retrieval weights, model-routing allocations, validator thresholds, and tool-selection policies from observed performance signals.

Contributions. This paper makes six contributions:

1. It formulates enterprise question answering as a multi-tool control problem over text retrieval, graph traversal, SQL execution, and synthesis.
2. It proposes SERQ-Agent, a detailed architecture for unified RAG+SQL+KG enterprise reasoning.
3. It defines typed evidence plans that make query decomposition auditable and executable.
4. It introduces a self-optimization loop that updates routing, retrieval, and validation policies from quality, cost, latency, and execution signals.
5. It provides an implementation blueprint using PostgreSQL, pgvector, schema relationship graphs, event graphs, read-only SQL sandboxes, and adaptive LLM routing.
6. It specifies an evaluation protocol for enterprise-grade benchmarks spanning answer correctness, SQL execution accuracy, citation faithfulness, graph reasoning, cost, latency, and safety.

2 Background and Motivation

2.1 Limits of Text-Only RAG

Retrieval-augmented generation grounds language-model answers in external documents [1]. This improves factuality for knowledge-intensive tasks, but standard chunk retrieval is weak for enterprise workflows that require joins, aggregation, row-level filters, and current operational state. Chunking a database export destroys schema constraints. Retrieving a table as text hides primary keys, foreign keys, data types, and business semantics. Even strong dense retrieval can miss a relevant table because the table name is abbreviated or because the question references a business concept rather than a column name.

2.2 Limits of Text-to-SQL Alone

Text-to-SQL systems translate natural-language questions into SQL over a known database schema. Benchmarks such as Spider accelerated progress on cross-domain SQL generation [2], and recent LLM-based methods improve complex query construction through decomposition and self-correction [3]. In production, however, text-to-SQL systems face schema scale, access control, query safety, and business-context ambiguity. Production databases can contain hundreds or thousands of tables. Injecting the full schema into a prompt is expensive and confusing. More importantly, many enterprise questions require information outside the database: policy text, incident timelines, customer contracts, or analyst notes.

2.3 Why Knowledge Graphs Matter

Knowledge graphs provide explicit relationships among entities, events, documents, and database records. Event graphs are especially useful when enterprise answers require causal or temporal reasoning. EventForge demonstrated that LLM-based event schema induction can be organized as typed modules and stored alongside dense retrieval artifacts in PostgreSQL/pgvector [6]. This suggests a practical path for enterprise agents: combine dense retrieval for semantic recall, graph traversal for multi-hop structure, and SQL execution for exact current facts.

2.4 Why Self-Optimization Matters

Enterprise workloads are non-stationary. Schemas change. Documents are revised. Model providers update endpoints. User populations shift. Latency and cost constraints vary by workflow. A static RAG pipeline can silently degrade when retrieval distributions change or when a model becomes weaker on a task class. LLM-Advisor and feedback-control routing show that model selection can be treated as an adaptive allocation problem rather than a fixed configuration [10, 12]. SERQ-Agent extends that idea from LLM choice to the whole knowledge pipeline.

3 System Overview

SERQ-Agent is organized around five planes: ingestion, representation, planning, execution, and optimization. Table 1 summarizes the roles.

At query time, the agent receives a tuple

$$Q = (q, u, C, A),$$

Table 1: SERQ-Agent system planes.

Plane	Primary responsibility	Representative components
Ingestion	Convert enterprise sources into normalized artifacts	PDF extraction, HTML/wiki parsing, table metadata extraction, ACL capture, document versioning
Representation	Store structured, unstructured, and graph evidence	Vector chunks, schema relationship graph, event graph, entity index, provenance records
Planning	Convert user questions into typed evidence plans	Intent classification, tool selection, decomposition, risk assessment, constraint capture
Execution	Run retrieval, graph, SQL, and synthesis steps safely	Hybrid retriever, SQL generator, read-only sandbox, validators, answer composer
Optimization	Update policies from feedback and telemetry	Routing controller, retrieval-weight optimizer, validation threshold tuner, cost-latency monitor

where q is the natural-language request, u is the user or service identity, C is a set of operational constraints such as latency and cost budgets, and A is the applicable access-control context. The output is not merely a text answer. It is an evidence package:

$$Y = (a, E_D, E_G, E_S, V, P),$$

where a is the final answer, E_D is document evidence, E_G is graph evidence, E_S is SQL evidence, V contains validation records, and P contains provenance and policy decisions.

4 Data Representation

4.1 Document Store

Documents are chunked into semantically coherent passages and stored with embeddings, metadata, document identifiers, section headers, version timestamps, and access-control labels. Unlike simple RAG systems, SERQ-Agent treats chunk provenance as a first-class object. Each chunk stores

$$(doc_id, chunk_id, version, section, text, embedding, acl, timestamp, source_uri).$$

The retrieval layer supports both flat dense retrieval and hierarchical retrieval. Hierarchical retrieval can use RAPTOR-style recursive summarization to retrieve from multiple abstraction levels [5].

4.2 Schema Relationship Graph

For each relational database, SERQ-Agent builds a schema relationship graph

$$G_S = (V_S, E_S).$$

Vertices represent databases, schemas, tables, columns, views, and business metrics. Edges represent primary-key relationships, foreign-key relationships, view dependencies, semantic aliases,

ownership, and access constraints. This follows the Inference-from-RDBMS design in which the schema graph supports pruning before SQL generation [4].

Given question q, each table T_j receives a relevance score:

$$R(T_j, q) = \alpha \cdot \cos(e_q, e_{T_j}) + \beta \cdot \mathrm{PR}(T_j) + \gamma \cdot \mathrm{Alias}(T_j, q) - \eta \cdot \mathrm{ACLBlock}(T_j, u).$$

The agent passes only the induced subgraph above a threshold into SQL generation.

4.3 Event Knowledge Graph

SERQ-Agent stores event graphs derived from documents, tickets, incident reports, policies, and logs:

$$G_E = (V_E, E_E).$$

Event nodes include natural-language descriptions, event type, actors, affected entities, timestamps, confidence, source spans, and embedding vectors. Edges encode temporal, causal, hierarchical, evidential, and entity-linked relationships. EventForge provides the base pattern: typed event induction modules, duplicate control through embedding similarity, graph construction, and unified PostgreSQL/pgvector storage [6].

4.4 Entity and Record Linkage

The bridge between documents and SQL is an entity-linking layer. Document mentions such as customer names, vendor names, contract IDs, incident IDs, invoice numbers, and product codes are linked to canonical entity identifiers. A link record has the form

$$\ell = (mention, entity_id, source, confidence, resolver, acl).$$

Entity links allow the agent to transform a document-grounded statement into a SQL filter or join candidate. For example, a retrieved contract clause mentioning "Premium Support Plan" can be linked to a product code used in account tables.

5 Typed Evidence Plans

The core planning abstraction is a typed evidence plan. A plan is a directed acyclic graph of evidence operations:

$$\Pi = (O, D),$$

where each operation $o \in O$ has a type, input dependencies, expected output schema, validators, and allowed tools. Table 2 lists core operation types.

A sample plan for "Which customers affected by incident I-104 also have premium support language in their contracts?" is:

```
1. GraphTraverse(incident_id = "I-104", relation = affected_customer)
2. DocRetrieve(query = "premium support contract language", filter = customer_ids from
    step 1)
3. SchemaPrune(query = "active customer account support tier", entities = customer_ids)
4. SQLGenerate(schema = step 3, filters = customer_ids, fields = account status and
    support tier)
5. SQLValidate(sql = step 4, mode = read_only, max_rows = 5000)
6. EvidenceFuse(graph = step 1, docs = step 2, sql = step 5)
7. AnswerCompose(evidence = step 6, format = table plus summary)
```

Table 2: Typed operations in a SERQ-Agent evidence plan.

Operation type	Purpose	Example output
DocRetrieve	Retrieve document passages	Policy sections, contract clauses, incident paragraphs
GraphTraverse	Traverse event/entity graph	Causal chain, affected entities, temporal path
SchemaPrune	Select relevant schema subgraph	Tables, columns, joins, business aliases
SQLGenerate	Produce candidate SQL	Parameterized read-only query
SQLValidate	Check SQL safety and execution	Syntax validity, row count, result schema
EvidenceFuse	Combine document, graph, and SQL evidence	Grounded intermediate conclusion
AnswerCompose	Produce final answer and citations	Natural-language answer with evidence table

Typed plans give the system a concrete object to inspect, validate, cache, and optimize. They also reduce hallucination by separating planning, evidence collection, and final answer synthesis.

6 Query Planning and Tool Selection

SERQ-Agent classifies each question along five axes: evidence modality, answer operation, risk level, freshness need, and validation need. Evidence modality distinguishes document, SQL, graph, or mixed questions. Answer operation distinguishes lookup, aggregation, comparison, causal explanation, compliance check, forecast, and remediation. Risk level determines whether the agent can answer directly, must include stronger evidence, or must escalate.

The planner selects an evidence plan by minimizing expected loss:

$$\Pi^* = \arg\min_{\Pi \in \mathcal{P}(q)} \mathbb{E}\left[L_{ans}(\Pi) + \lambda_c C(\Pi) + \lambda_l Lat(\Pi) + \lambda_r Risk(\Pi)\right],$$

where L_{ans} is expected answer error, C is cost, Lat is latency, and $Risk$ captures security or compliance exposure.

For ambiguous questions, the planner can ask a clarification question. However, enterprise agents should not overuse clarification. If a safe, bounded answer can be produced with explicit assumptions, the agent should proceed and surface those assumptions.

7 SQL Generation and Validation

The SQL subsystem extends schema-aware query translation [4]. It has four stages:

1. Extract a schema subgraph from G_S using semantic and graph relevance.
2. Generate one or more candidate SQL queries from the user question, plan context, and schema subgraph.
3. Validate candidates with static checks, permission checks, and dry-run execution.

4. Execute the selected query in a read-only sandbox with row, time, and cost bounds.

The validator rejects queries with disallowed operations:

$$\{\texttt{INSERT}, \texttt{UPDATE}, \texttt{DELETE}, \texttt{DROP}, \texttt{ALTER}, \texttt{TRUNCATE}, \texttt{CREATE}\}.$$

It also checks that every referenced table and column appears in the pruned schema graph and is permitted by the user's access-control context. If execution fails, the error is fed back to the SQL generator, but the number of repair attempts is bounded.

7.1 SQL Evidence Contract

Every SQL result returned to the answer composer must satisfy an evidence contract:

$$E_{SQL} = (query, parameters, result_schema, row_sample, \\ aggregate, execution_time, validator_status, acl_status).$$

This prevents the final LLM from inventing database evidence. It can summarize only the records, aggregates, or samples that are explicitly present in the SQL evidence object.

8 Hybrid Retrieval and Graph Reasoning

The document and graph subsystem retrieves evidence through a weighted combination of dense similarity, lexical matching, graph proximity, entity linkage, and freshness:

$$Score(c, q) = w_d Dense(c, q) + w_b BM25(c, q) + w_g GraphProx(c, q) \\ + w_e EntityMatch(c, q) + w_t Fresh(c).$$

The weights are not fixed. SERQ-Agent updates them from feedback and validation outcomes. For example, if dense retrieval repeatedly returns semantically similar but policy-irrelevant chunks, lexical and entity weights may increase for compliance tasks. If graph traversal improves incident reconstruction, graph proximity may increase for operational questions.

Graph reasoning is used for multi-hop questions. Given a set of seed entities S and relation types $\mathcal{R}$, the graph traverser returns a bounded subgraph:

$$G_q = \text{Traverse}(G_E, S, \mathcal{R}, h, b),$$

where h is maximum hop count and b is a branching limit. Each returned path includes source spans, edge confidence, and timestamps.

9 Self-Optimization Loop

SERQ-Agent optimizes three policy families:

1. **Model routing policy** π_M: selects LLMs or local models for planning, SQL generation, extraction, validation, and synthesis.
2. **Retrieval policy** π_R: sets weights for dense, lexical, graph, entity, and freshness signals.
3. **Validation policy** π_V: sets thresholds for SQL repair, citation sufficiency, answer abstention, and escalation.

Each completed query produces telemetry:

$$\tau = (q, \Pi, E, y, cost, latency, validator_results, user_feedback, downstream_corrections).$$

The optimizer maps telemetry to policy updates:

$$(\pi_M, \pi_R, \pi_V)_{t+1} = Update((\pi_M, \pi_R, \pi_V)_t, \tau_t).$$

9.1 Adaptive Model Routing

For each model M_i and task class k, the router maintains a rolling performance estimate:

$$p_{i,k}(t) = \text{EMA}(reward_{i,k}).$$

Rewards combine correctness, validator pass rate, citation faithfulness, cost, and latency:

$$r = \omega_q qacc + \omega_v vpass + \omega_f faith - \omega_c cost - \omega_l latency.$$

The routing score is:

$$S(M_i, k) = \omega_p p_{i,k} - \omega_c \hat{c}_i - \omega_l \hat{\ell}_i + \omega_s safety_i.$$

The selected model is sampled from a floored softmax:

$$a_i = (1 - K a_{\min}) \frac{\exp(S(M_i, k)/T)}{\sum_j \exp(S(M_j, k)/T)} + a_{\min}.$$

The floor $a_{\min}$ preserves persistent exploration, following the motivation of feedback-control routing [12]. Without this floor, a model that performed poorly during one transient regime may never be sampled again after conditions change.

9.2 Retrieval Weight Optimization

Retrieval weights are updated by task class. If an answer is corrected by a user or fails a validator, the system records which retrieved chunks and graph paths supported the failed claim. Weights that promoted false evidence are penalized; weights that promoted validated evidence are reinforced:

$$w_{t+1} = \text{Proj}_\Delta(w_t + \rho \nabla \hat{R}_t),$$

where Proj_Δ projects weights onto the simplex and $\hat{R}_t$ is a retrieval reward proxy.

9.3 Validation Threshold Tuning

Validation policies must be risk-aware. For low-risk internal search, the agent may answer with moderate evidence confidence. For compliance, finance, or security contexts, it should require stronger evidence and may abstain. The threshold for answer acceptance is:

$$\theta(q) = \theta_0 + \theta_{risk} Risk(q) + \theta_{sens} Sensitivity(q) - \theta_{urg} Urgency(q).$$

If confidence falls below $\theta(q)$, the agent either asks for clarification, returns partial evidence, or escalates to a human reviewer.

10 Safety and Governance

Enterprise RAG+SQL agents carry different risks than consumer chat systems. They can expose sensitive records, produce incorrect aggregates, or make unsupported policy claims. SERQ-Agent addresses these risks through access control, read-only execution, provenance, and abstention.

10.1 Access Control

Access control is enforced before retrieval and before SQL execution. A user cannot retrieve document chunks, graph paths, or database rows outside their permission context. The answer composer receives only filtered evidence. This is important because post-generation redaction is not enough: a model may leak information in paraphrase if it sees unauthorized evidence.

10.2 Read-Only Execution

All generated SQL is executed in read-only transactions. Queries are bounded by timeouts, row limits, memory limits, and warehouse cost limits. Sensitive columns can be masked or aggregated depending on policy.

10.3 Provenance

Every answer includes provenance at the evidence-object level. For documents, provenance includes document ID, version, section, and chunk ID. For SQL, provenance includes query text, parameters, validator status, and execution timestamp. For graphs, provenance includes nodes, edges, source spans, and confidence scores.

10.4 Abstention

SERQ-Agent is allowed to abstain. Abstention is not a failure when evidence is insufficient, access is blocked, SQL validation fails, or retrieved sources conflict. The system should state the reason and return the closest safe evidence.

11 Implementation Blueprint

A practical implementation can be built with PostgreSQL for metadata, pgvector for dense embeddings, relational tables for graph nodes and edges, section-aware document extraction, event induction modules, entity resolution, information-schema extraction, business glossary mapping, typed JSON evidence plans, tool workers, telemetry storage, and governance services. The minimal data model includes:

- `documents(doc_id, source_uri, version, title, acl, created_at)`
- `chunks(chunk_id, doc_id, section, text, embedding, acl)`
- `kg_events(event_id, description, type, timestamp, confidence, embedding)`
- `kg_edges(src, dst, relation, confidence, source_chunk)`
- `entities(entity_id, canonical_name, type, aliases)`
- `entity_links(entity_id, source_type, source_id, mention, confidence)`

Table 3: Evaluation protocol for enterprise RAG+SQL knowledge agents.

Task family	Required capability	Metrics
Document-grounded QA	Retrieve and cite correct policy or contract text	Exact support, citation precision, citation recall
SQL-grounded QA	Generate and execute correct SQL	Execution accuracy, result accuracy, unsafe-query rejection
Mixed RAG+SQL QA	Combine documents with database state	End-to-end answer accuracy, evidence consistency
Incident reconstruction	Recover causal or temporal chain	Path F1, edge accuracy, source-span support
Compliance checks	Apply policy text to structured records	Rule accuracy, false-positive rate, false-negative rate
Optimization	Improve under repeated workload	Cost, latency, quality reward, regret versus oracle
Safety	Avoid unauthorized or unsupported answers	ACL violation rate, abstention precision, audit completeness

- `schema_nodes(node_id, db, schema, table, column, type, embedding, acl)`
- `schema_edges(src, dst, relation, confidence)`
- `query_runs(run_id, user_id, plan, telemetry, created_at)`

12 Evaluation Protocol

SERQ-Agent should be evaluated on mixed-modality enterprise tasks rather than only document QA or SQL benchmarks. Table 3 lists task families and metrics.

12.1 Baselines and Ablations

Strong baselines should include text-only RAG over documents and serialized tables, text-to-SQL with full schema injection, naive schema RAG without graph-aware pruning, hybrid RAG plus SQL without event graph traversal, hybrid RAG plus SQL with static model routing, SERQ-Agent with self-optimization disabled, and full SERQ-Agent. Important ablations remove schema relationship graph pruning, SQL execution validation, event graph traversal, entity linking, adaptive model routing, retrieval-weight updates, and persistent exploration.

12.2 Expected Hypotheses

Based on prior component results, SERQ-Agent should be expected to improve SQL execution accuracy relative to full-schema prompting and naive schema RAG, consistent with schema-aware pruning [4]; improve multi-hop incident and process questions relative to text-only RAG by using event graphs [6]; reduce cost and latency relative to static frontier-model routing by using dynamic model selection [10]; and recover better from workload drift than static routing by preserving feedback and persistent exploration [12].

13 Failure Modes

SERQ-Agent introduces new failure modes that should be studied explicitly. The planner may decompose a question into unnecessary operations, increasing cost and latency. Entity linking may map a document mention to the wrong database entity. Documents and SQL results may disagree because documents are stale or database records are incomplete. SQL may be syntactically safe but semantically wrong. The self-optimization loop may overfit to short-term user feedback or route too much traffic to a cheap model. These risks motivate plan complexity penalties, high-confidence entity links, conflict reporting, semantic SQL checks, offline regression tests, and exploration floors.

14 Discussion

The main design shift in SERQ-Agent is to treat enterprise knowledge work as evidence orchestration. The answer is not produced by one LLM call. It is produced by a sequence of typed operations that each generate evidence objects with validation records. This design makes answers auditable, supports optimization, and supports governance. A reviewer can inspect which document chunks, graph paths, SQL queries, and model calls contributed to a response. Because each operation has telemetry, the system can learn whether retrieval, schema pruning, graph traversal, or model routing caused a failure. Access control, abstention, and validation are enforced before answer synthesis rather than bolted on after generation.

The architecture is deliberately modular. An organization can adopt schema-aware SQL translation first, then add document RAG, then event graphs, then self-optimization. The strongest version combines all modules, but partial deployments still provide value.

15 Conclusion

Enterprise AI agents need to answer questions that cross documents, databases, events, and operational constraints. Text-only RAG and text-to-SQL solve important subproblems, but neither is sufficient for mixed enterprise reasoning. This paper proposed SERQ-Agent, a self-optimizing RAG+SQL knowledge-agent architecture that combines schema-aware SQL generation, hybrid document retrieval, event graph reasoning, adaptive model routing, execution validation, and telemetry-driven optimization. The result is a practical blueprint for enterprise knowledge systems that are grounded, auditable, cost-aware, and able to improve under changing workloads.

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[2] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018.

[3] Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems*, volume 36, pages 32415–32427, 2023.

[4] Harshil Lodhiya. Schema-aware query translation and tabular reasoning for enterprise databases. *International Journal of Research in Engineering, Science and Management*, 9 (7):87–89, Jul 2026. doi: 10.65138/ijresm.v9i7.3489. URL `https://journal.ijresm.com/index.php/ijresm/article/view/3489`.

[5] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059*, 2024.

[6] Harshil Lodhiya. Eventforge: Optimizable event schema induction with hybrid rag+kg storage. *World Journal of Advanced Research and Reviews*, 31(2):584–591, 2026. doi: 10.30574/wjarr.2026.31.2.2105.

[7] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[8] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[9] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

[10] Harshil Lodhiya. Llm-advisor: Dynamic model selection and query routing in heterogeneous multi-llm architectures. *International Journal of Research in Engineering, Science and Management*, 9(7):84–85, Jul 2026. doi: 10.65138/ijresm.v9i7.3488. URL `https://journal.ijresm.com/index.php/ijresm/article/view/3488`.

[11] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[12] Harshil Lodhiya. Feedback control for adaptive language model routing under non-stationary workloads. *Journal of Computer Science and Technology Studies*, 8(8):238–243, 2026. doi: 10.32996/jcsts.2026.8.8.17.

[13] Aurelien Garivier and Eric Moulines. On upper-confidence bound policies for switching bandit problems. *Proceedings of the 22nd International Conference on Algorithmic Learning Theory*, pages 174–188, 2011.

[14] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[15] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[16] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. *Proceedings of the 7th Python in Science Conference*, pages 11–15, 2008.