LangGraph-Orchestrated LLM Agents for Scalable Movie Knowledge Graphs and Question Answering

Alex Kaplunovich
University of Maryland, Baltimore, USA

akaplun1@umbc.edu

Abstract: Recent advances in large language models (LLMs) and agent-based orchestration are transforming automated knowledge graph (KG) creation as well as robust question answering in complex domains. We present a modular, multi-agent system that extracts, integrates, and reasons over diverse NoSQL movie data, powered by state-of-the-art LLMs such as GPT-4.1. Our architecture converts unstructured plots, cast/crew metadata, and numeric attributes into high-fidelity KGs - enabling both natural language and programmatic queries. To maximize reliability and flexibility, the system unifies multiple retrieval strategies - keyword search, vector similarity, knowledge graph querying, and summarization - each deployed as an autonomous pipeline. Parallel orchestration via LangGraph supports adaptive engine selection, concurrent execution, and robust answer verification with LLM ensemble "jury" scoring. Critically, the framework features comprehensive observability, allowing detailed monitoring and analysis of agent decisions, pipeline performance, and query outcomes. By treating each retrieval method and LLM as a specialized agent, our approach delivers scalable, explainable, and highly accurate results (up to 97%), significantly surpassing monolithic solutions. This agentic, observable architecture paves the way for next-generation autonomous analytics, integration, and decision support across data-rich domains.

Keywords: LLM agents, Multi-Agent orchestration, Knowledge graph construction, Question answering, RAG systems, LangGraph

1. Introduction

Recent breakthroughs in large language models (LLMs) have significantly enhanced automated question answering (QA) and information retrieval, enabling sophisticated reasoning across complex, heterogeneous datasets (Liu, 2022; West, 2024). The emergence of agent-based orchestration frameworks such as LangGraph now makes concurrent, graph-structured workflows feasible, offering greater flexibility and scalability compared to traditional linear pipelines (Han et al., 2023; Tran et al., 2025). Leveraging these advancements, we introduce a modular, multi-agent LLM system designed to fully automate knowledge graph (KG) construction and robust QA over semi-structured NoSQL movie databases.

Movie data exemplifies both the challenges and opportunities of contemporary QA technologies, comprising structured metadata (cast, crew, box office figures) alongside unstructured narrative elements (plot summaries, character dynamics). Traditional rule-based extraction methods struggle with such complexity, often failing to capture implicit relationships embedded within textual narratives. Advanced models - including GPT-4.1, Claude 3.7, and Llama 4 - provide powerful new capabilities for semantic parsing and relational inference but continue to face significant operational hurdles, such as hallucinations, schema drift, and context-window limitations (Zhu et al., 2024).

Our framework directly addresses these challenges through concurrent multi-agent orchestration, dynamically combining diverse retrieval strategies - including keyword, vector similarity, summarization, and knowledge graph approaches - guided by intelligent ensemble verification. Leveraging iterative zero-shot prompting (Carta et al., 2023) and retrieval-augmented generation (Wang et al., 2024, Karpukhin et al., 2020), the system automatically builds both property-based and triplet-based KGs without manual schema engineering. By embedding semantic concepts during KG creation, our architecture supports real-time, adaptive querying across evolving datasets, enabling complex searches (identifying collaborations, aggregating box-office metrics, or exploring thematic connections) without the overhead of runtime inference.

Furthermore, comprehensive observability integrated via unified logging and modular design ensures transparency, auditability, and continuous optimization, thereby laying the foundation for robust, explainable, and scalable retrieval-augmented analytics across complex, multimodal domains.

Our contributions are threefold:

- A modular LangGraph-based architecture for automatic knowledge graph construction and concurrent question answering over large movie datasets.
- Detailed evaluation of five LlamaIndex-powered query engines and a router agent, each leveraging different LLMs and retrieval strategies.
- Scientific analysis of concurrency, result merging, and best practices for agentic pipeline design.

1.1 Why not Just an LLM or a Simple Database?

While prompting LLMs over raw movie records may suffice for simple queries, our findings reveal clear limitations at scale-especially for heterogeneous data and complex reasoning. This motivates our multi-agent architecture:

- **Structured Access**: Basic RAG struggles with deep, semantically diverse fields (e.g., actor-role-gender across languages), especially for alias resolution or contextual inference. Specialized tools (keyword, vector, summarization) improve precision and recall.
- **Efficiency**: Monolithic LLMs incur high cost due to prompt length and redundant context. Lightweight agents handle routine queries efficiently, reserving LLMs for complex tasks (Table 4).
- **Robustness**: Single models can hallucinate or contradict. Our ensemble verifier layer (Section 3.4) offers multi-model judgment for reliable results.
- **Observability**: LangGraph enables per-agent logging and diagnostics, unlike opaque monolithic systems.

In short, LLMs are powerful generalists-but in complex QA, delegation to specialized agents enhances scalability, transparency, and trust.

2. Dataset and Test Set Creation

2.1 Data Source and Schema

The movie dataset was sourced from AWS DynamoDB and is modeled as a nested JSON/NoSQL-style schema, including structured and unstructured fields:

- Title, Year, Release, Revenue, Rating, Votes - numeric and categorical metadata
- Plot - long-form unstructured text (average 1,280 tokens)
- Cast - list of actor names
- Characters - actor-character-role tuples with demographic metadata
- Director, Genres, Languages, Countries - categorical lists
- ID, IID - Wikipedia and IMDb-style identifiers


```
{
 "Title": "Titanic",
 "Year": 1997,
 "Revenue": 2185372302,
 "Rating": 7.9,
 "Genres": ["Historical fiction", "Romantic drama", "Disaster"],
 "Director": ["James Cameron"],
 "Plot": "84 years later, a 100 year-old woman named Rose DeWitt Bukater tells the story...",
 "Cast": ["Leonardo DiCaprio", "Kate Winslet", "Billy Zane"],
 "Characters": [
  {
   "Actor name": "Kate Winslet",
   "Character name": "Rose DeWitt Bukater",
   "Actor gender": "F",
   "Actor date of birth": "1975-10-05",
   "Actor height (in meters)": 1.69
  }
```


Figure 1: Database row example for Titanic 1997

Table 1: Dataset Statistics

Metric	Value
Total movies	105,324
Avg. plot length	213 tokens
Avg. characters/movie	8.4
Unique actors	336,142
Avg. actors/movie	7.3
Unique directors	35,278
Avg. directors/move	1.12
Genres	370
Avg. genres/movie	3
Languages	225
Countries	148
Release Year Range	1899-2025

Table 1 and Figure 1 provide a comprehensive overview of the dataset used in our study. Table 1 summarizes key dataset statistics, including the total number of movies, average plot length in tokens, average number of characters per movie, unique actors, average genres per movie, number of languages covered, unique genres, and unique countries represented. These metrics illustrate both the scale and diversity of the collection, highlighting its suitability for benchmarking complex question answering and knowledge graph construction. Figure 1 presents a representative example of a single dataset row, showcasing the nested JSON/NoSQL structure that characterizes each movie record. Fields such as Title, Year, Revenue, Rating, Genres, Director, Plot, Cast, and Characters are illustrated, reflecting the rich, heterogeneous information available for extraction and analysis. Together, Table 1 and Figure 1 contextualize the challenges and opportunities inherent in processing real-world movie data at scale.

2.2 QA Test Set Generation

To construct a challenging and diverse test set, we load the movie records into pandas dataframe and generate 1,000 question-answer pairs from the data to expose the full richness of relationships: cross-field queries (actors in movies by director X), deep narrative reasoning (movies with certain themes or rare genres), aggregation (highest revenue films per country), and entity coreference (actor aliases, multiple roles).

This approach resulted in a ground truth dataset with broad coverage:

- Factoid, list, and relationship queries
- Numeric and textual answers
- Cross-field inference and alias resolution

3. Methodology

3.1 Retrieval Query Engines

Our system utilizes five distinct retrieval engines, each represented as a QueryEngineTool in LlamaIndex (West, 2024) and orchestrated by LangGraph:

- Vector Engine: Semantic embedding search over plots and metadata.
- Keyword Engine: Exact and keyword match over movie titles, actors, genres.
- Summary Engine: High-level LLM-generated summaries per movie (for aggregation and context).
- Property Graph Engine: Schema-based property graph of movies, actors, genres, directors, supporting property/relationship queries.
- Knowledge Graph Engine: Triplet-based knowledge graph for relationship and fact extraction.

All five are run as parallel nodes in the LangGraph architecture, feeding their candidate answers to the next stage.

3.2 Dynamic Router Query Engine

A dynamic router query engine is constructed using RouterQueryEngine and a GenericMultiSelector, set to select 3 out of 5 engines from section 3.2 per query, based on prompt-driven relevance. The router engine analyzes each user query and determines-via LLM-powered GPT-4.1 prompt engineering-which subset of retrieval pipelines should be executed for optimal results. This balances accuracy, diversity, and efficiency, functioning as an ensemble method that combines the strengths of selected retrievers for improved query outcomes.

LlamaIndex offers exceptional flexibility for configuring the Router query engine, enabling prompts to dynamically select and orchestrate retrieval strategies within a unified workflow. This modular design empowers users to mix engines, tailoring execution to the complexity and ambiguity of each question. Adaptive routing lets developers optimize precision, recall, latency, and cost through context-aware choices. Such flexibility is especially valuable in complex domains-like code generation, biomedical search, or enterprise knowledge management-where no single retrieval strategy suffices. LlamaIndex streamlines integration of heterogeneous sources and models, enabling fine-grained control, experimentation, and observability across the QA pipeline. This adaptability positions LlamaIndex as a robust foundation for production-grade RAG.

3.3 Automatic Knowledge Graph Construction

We use prompt engineering to guide LLMs (GPT-4.1) in extracting semantic triples (subject, predicate, object) from raw movie data, building high-coverage knowledge graphs (KGs) without manual schema design. These triples embed semantic types and relations, supporting both programmatic and natural language queries.

In parallel, we construct property graphs from structured data in AWS DynamoDB, modeling entities (e.g., movies, actors, genres) and their attributes and relationships. Together, the KG and property graph engines capture both relational and descriptive knowledge.

Despite the semi-structured NoSQL schema, KGs offer distinct advantages:

- Semantic Inference: Reveal implicit patterns (co-actor chains, genre hierarchies), non-structured elements handling (for example, movie plots).
- Unified Representation: Align structured metadata and unstructured text (plot summaries).
- Efficient Reasoning: Enable symbolic graph traversal and interpretable logic.
- Cross-Source Enrichment: Provide a common format for integrating external data (Wikipedia, TMDB).

While NoSQL supports basic lookups, KGs offer a semantically rich, reasoning-friendly structure with minimal overhead and high value for complex, inference-driven queries.

3.4 Verification Engines

After the query engine returns an answer, we initiate a robust verification process by employing an LLM-as-judge framework, utilizing five distinct state-of-the-art LLM models to independently assess the response. Each model scores the answer on a 0 to 100 scale, enabling a multi-perspective evaluation that captures subtle nuances, potential inaccuracies, and contextual appropriateness. This ensemble-based judging mechanism not only mitigates the risk of individual model bias or blind spots, but also provides a more reliable and holistic measure of answer quality. By aggregating the scores - whether through averaging, weighted consensus, or more advanced ensemble techniques - we can make confident decisions about answer acceptance, flagging, or refinement. Moreover, this approach allows us to dynamically benchmark LLM models against each other, identifying strengths and weaknesses in specific domains or query types. Ultimately, leveraging multiple LLMs as parallel judges elevates the trustworthiness and reliability of the system, paving the way for production-ready applications that demand high precision, transparency, and user confidence.

Five LLMs serve as verifiers in this study:

- GPT-4.1-mini - Fine-tuned, mid-size model (~13B parameters), offering fast, accurate, and context-aware reasoning.
- GPT-4.1-nano - Compact version (~7B parameters) optimized for very low latency and efficient verification.
- GPT-4o-mini - Latest “Omni” model with multimodal capabilities, high benchmark scores, and strong code understanding.
- o3-mini - Specialized variant from the GPT-3.5 family, excels in logical reasoning and large-context tasks.

- o4-mini - Streamlined GPT-4 model, balancing accuracy and inference speed for reliable, real-time code and logic evaluation.

Each engine scores the pipeline answers for correctness and completeness, following ensemble verification (OpenAI, 2023). Moreover, we run these five verifiers concurrently because they are independent, saving time and merging the results in the LangGraph node. By combining these five models, our verification pipeline benefits from diverse strengths: long-context comprehension, multimodal reasoning, deep STEM/logical analysis, and rapid low-latency judgments. This ensemble ensures comprehensive quality control while maintaining efficiency and scalability - key to trustworthy agentic systems in production environments.

3.5 Observability and LangGraph Optimizations

It is crucial to implement agentic applications efficiently. Every node of our workflow extends a base class containing all the common code, including logging, initialization and error handling. Every node need to implement just execute method with the business logic, LLM calls or other functionality. For observability engine we are using serverless NoSQL DynamoDB that does not need any servers, fully integrated with AWS cloud ecosystem and has one millisecond latency and querying functionality. Figure 2 demonstrates our LangGraph base class supporting observability and modularity. Every pipeline node extends the class.

```
class BaseNode:
   """Every LangGraph node derives from this class for uniform observability behavior."""
   def __init__(self):
      self.name = self.__class__.__name__  # name of the class extracted dynamically
   async def __call__(self, state: AgentState):
      start=time()
      new_state = await self.execute(state)
      end=time()
      item = {
         "execTime": exetTime,

        ...
      }
       logItem(item) #log node execution into DynamoDB
      return new_state
   async def execute(self, state: AgentState) :  # should be overridden in every child
      raise NotImplementedError
```

Figure 2: Base node class for LangGraph pipeline supporting observability and modularity

3.6 Baseline Comparison

To establish the effectiveness of our multi-agent approach, we implemented a single-agent baseline using GPT-4.1 with simple RAG retrieval engine over the entire dataset. This baseline performs query engines search for each of the engines from 3.1 and generates answers with ensemble scoring and verification.

4. LangGraph Architecture

Our architecture is built on a directed graph where each node represents a retrieval or verification step (Kaplunovich, 2025). LangGraph's native concurrency support enables the five retrieval pipelines to run in parallel, dramatically reducing latency. The outputs of all retrieval engines are merged and dispatched to five

verification engines (one per model), which also operate concurrently. The merge node aggregates verification scores, selecting the highest-rated answer as the system's response. We generate verification scores (5 per answer) for 6 answers by each query engine (described in sections 3.1 and 3.2). Figure 3 demonstrates the LangGraph-based agentic workflow designed for orchestrated, reliable question answering and verification. The process begins at the start node, followed by a conditional check (check_load) to determine whether to load and build new resources or retrieve precomputed indices from disk (load_and_build vs. load_from_disk). Once the appropriate resources are ready, a router is instantiated (create_router) and queries are prepared (prepare_queries).

The system supports concurrent multi-engine querying through the run_queries node (running router engine), then distributing the queries across specialized engines run in parallel: knowledge graph (run_queries_kg),

keyword search (run_queries_kw), property graph (run_queries_pg), summarization (run_queries_sum), and vector retrieval (run_queries_vec). This enables the aggregation of diverse retrieval strategies tailored to the query type.

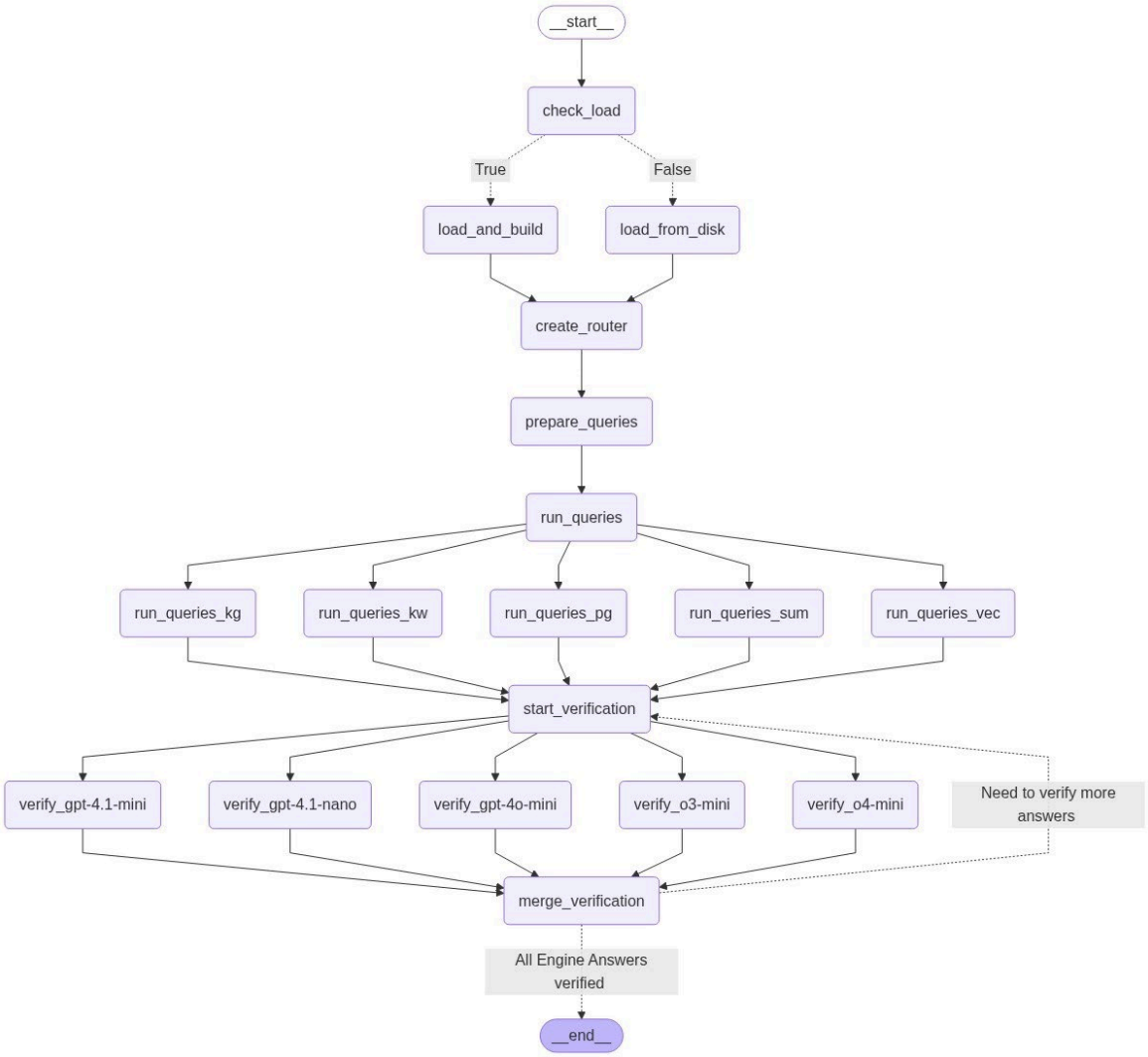

Figure 3: Multi-agent LangGraph concurrent agentic pipeline

After obtaining candidate answers for each 6 query engines, the workflow initiates a rigorous verification stage (start_verification), invoking multiple state-of-the-art LLM judges in parallel: verify_gpt-4.1-mini, verify_gpt-4.1-nano, verify_gpt-4o-mini, verify_o3-mini, and verify_o4-mini. Each judge independently assesses the answer's quality and correctness, generating scores for downstream aggregation.

For each query engine, the workflow launches five parallel verifiers to evaluate all answers produced by that engine. Each loop verifies all results from one query engine before proceeding to the next. The queryEngineKey parameter ensures that each set of five verifiers is correctly associated with its corresponding engine. This process continues until all answers from all engines have been thoroughly verified. Once verification is complete for every engine, the outputs are merged (merge_verification), and the workflow terminates successfully (end). This structure guarantees systematic and comprehensive quality control, as every result from every engine is independently judged before the workflow concludes.

This agentic, multi-stage workflow embodies modularity, scalability, and verifiability - key requirements for building production-grade retrieval-augmented generation (RAG) and agent-based QA systems. Concurrency in LangGraph is achieved through parallel node execution controlled by CONCURRENCY parameter enforced by asyncio.Semaphore. This supports efficient use of LLM APIs and local resources. LangGraph also allows flexible merging and conditional logic.

5. Experimental Results

The results reveal several key patterns about engine performance, threshold effects, and system reliability:

- Fine-Grained Threshold Trends: Using a finer threshold step (every 5 points) highlights that accuracy decreases steadily as the bar for correctness rises, but the rate and shape of this decline differ by engine.
- SUM Engine Superiority: The SUM (summarization) engine consistently achieves the highest precision at all thresholds, maintaining 97% accuracy up to threshold 55 and staying above 80% up to threshold 80. Its robust performance, even at strict thresholds, suggests that LLM-verified summarization answers are exceptionally clear and well-aligned with the ground truth. Even at a demanding threshold of 90, it remains the top performer (25%).
- Engine Robustness: KW (keyword), VEC (vector), and ROUTER engines show strong initial accuracy at low to mid thresholds, with KW and ROUTER exceeding 70% up to threshold 60. However, their performance falls off more sharply than SUM as thresholds increase, reflecting some trade-off between recall and answer specificity.
- KG and PG Engines: KG (knowledge graph) and PG (property graph) engines show more modest accuracy, starting just above 60% and dropping rapidly past threshold 70. This suggests that while they sometimes offer strong answers, these are less likely to achieve high LLM consensus - potentially due to the complexity or sparsity of structured outputs.
- Zero at Max Threshold: At a threshold of 100, no engine achieves any correct answers, underscoring the strictness of requiring all verifiers to return maximal confidence - a standard rarely met in multi-agent LLM systems.
- Implications for Deployment: The practical lesson is that threshold tuning is a powerful lever for balancing recall and precision. For applications prioritizing broad coverage, lower thresholds are optimal. For scenarios demanding only the highest certainty - such as critical decision support or automated code synthesis - higher thresholds enforce more reliable output, at the expense of recall.
- Ensemble Power: The consistently strong performance of SUM and hybrid engines (ROUTER, VEC, KW) demonstrates the importance of combining diverse retrieval paradigms. This multi-engine, multi-verifier approach provides both flexibility and robustness, adapting to the varied nature of real-world queries and data.

5.1 RAG Accuracy

Table 2: Query Engines Accuracy (Threshold Step 5)

		Threshold Accuracy									
Query Engine	**Avg score**	**50**	**55**	**60**	**65**	**70**	**75**	**80**	**85**	**90**	**95**
KG	58.47	0.61	0.6	0.56	0.53	0.39	0.36	0.3	0.21	0.12	0.11
PG	62.90	0.59	0.59	0.55	0.55	0.49	0.47	0.42	0.26	0.11	0.11
KW	73.05	0.79	0.79	0.74	0.71	0.62	0.56	0.48	0.3	0.2	0.2
SUM	85.90	0.97	0.97	0.95	0.93	0.9	0.86	0.84	0.49	0.25	0.14
VEC	74.42	0.87	0.87	0.74	0.71	0.58	0.5	0.46	0.3	0.15	0.14
ROUTER	79.55	0.81	0.81	0.71	0.67	0.6	0.51	0.46	0.29	0.18	0.18
All 5 engines	81.65	0.89	0.89	0.87	0.85	0.81	0.74	0.67	0.39	0.26	0.22

Table 2 presents the average verification score for each engine, complementing the visual analysis of score distributions. To quantify accuracy, a range of thresholds is applied to the scores: for a given threshold, an answer is considered correct if at least three of the LLM verifiers assigned a score exceeding that threshold. By sweeping the threshold from 50 to 100 (in increments of 5), we observe how each engine's apparent accuracy changes as the bar for "correctness" is raised.

Engines whose distributions are concentrated in higher score ranges - such as the SUM engine - maintain high accuracy even as the threshold increases, since a large proportion of their answers are confidently judged correct. Conversely, engines like KG and PG, whose distributions are more heavily weighted toward lower scores,

see a sharp decline in accuracy as the threshold rises. This is reflected in the table, where SUM retains the highest accuracy at every threshold, while KG and PG drop to near zero accuracy at stricter criteria.

This relationship between the score distribution and threshold-based accuracy illustrates the interplay between answer quality, verification rigor, and retrieval strategy. Engines with more robust high-confidence answers are less sensitive to strict verification, making them preferable for applications demanding reliable and trustworthy outputs. In contrast, engines with broader or lower score distributions may contribute valuable coverage in recall-oriented settings but require more cautious integration when high precision is critical.

Threshold selection should be driven by the intended application's risk tolerance and required reliability. The ensemble LLM approach, especially when combined with summarization, delivers a high-performing, resilient verification pipeline for complex, open-domain question answering and code-related tasks. The distributions of the verification scores for each engine type are shown in Figure 3. The histograms depict how frequently different verification scores occur across all evaluated queries, demonstrating variability among the retrieval methods. Notably, the summarization (SUM) engine displays a higher concentration of higher scores, whereas KG and PG show considerable lower-score density.

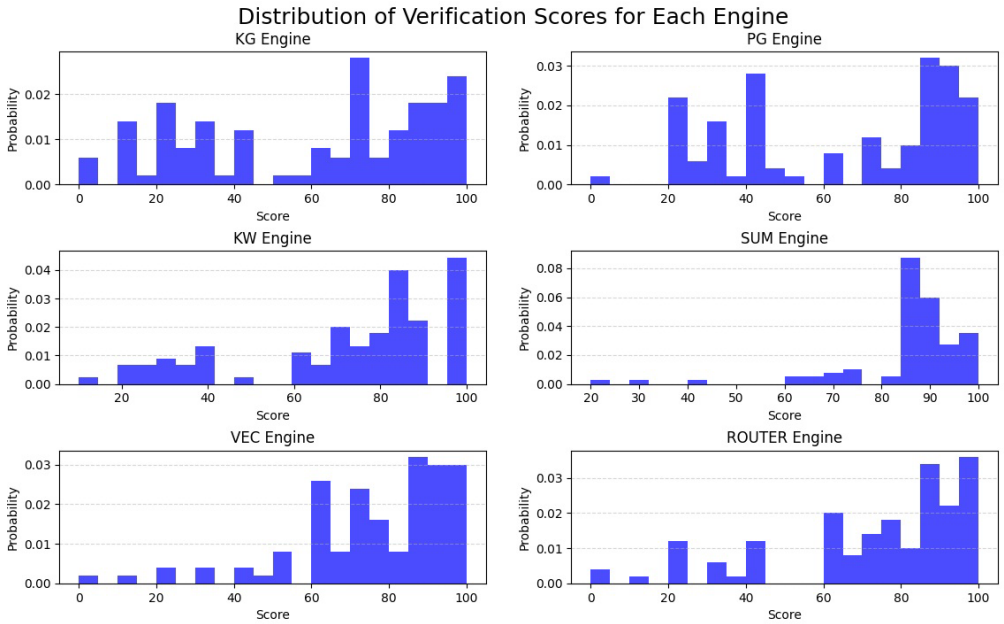

Figure 4: Score distributions for each query engine

Figure 4 shows verification score distributions for all six engines. KG and PG have moderately dispersed scores, with peaks near 70–100 but substantial mass at lower ranges. KW skews strongly toward 80–100, indicating frequent high-confidence retrievals. SUM peaks even more sharply in this range, yielding the most accurate results but at the highest cost and latency (see Section 5.3). VEC displays a broad spread with many mid- and low-confidence results, reflecting greater variability. ROUTER achieves the most balanced profile, combining strong high-end performance with coverage across mid and lower bins-leveraging multiple pipelines to maximize both precision and recall with highest score range (90-100).

5.2 Detailed Analysis of Router Engine Selections

Table 3: Frequency of Engine Selection by the Router Query Engine

Engine ID	Engine Description	Percentage (%)
1	Keyword Engine (Exact and keyword match search)	98
3	Property Graph Engine (Schema-based queries)	94
4	Knowledge Graph Engine (Triplet-based facts)	56
0	Vector Engine (Semantic embedding search)	47
2	Summary Engine (LLM-generated summaries)	5

A comprehensive statistical breakdown of engine usage frequency selected by LlamaIndex RouterQueryEngine is presented in Table 3.

The Router query engine dynamically selected combinations of 3 specialized retrieval methods tailored to individual queries. We pass a query to an LLM with query and engine descriptions (below) and it makes the choice. The retrieval engines were:

- Vector Engine (ID 0): Performs semantic embedding search over movie plots and metadata, capturing nuanced query contexts.
- Keyword Engine (ID 1): Conducts exact and keyword-based searches targeting precise movie titles, actors, and genre information.
- Summary Engine (ID 2): Generates high-level summaries using large language models, synthesizing comprehensive information from movie data.
- Property Graph Engine (ID 3): Enables schema-based graph queries about movies, actors, genres, directors, and their relationships, effective for structured data retrieval.
- Knowledge Graph Engine (ID 4): Utilizes a triplet-based knowledge graph for extracting and verifying facts and relationships from detailed movie data.

The Router's choice distribution highlights several important insights:

- Dominance of Keyword and Property Graph Engines: The Keyword engine (ID 1) was the most frequently selected (98%), closely followed by the Property Graph engine (ID 3, 94%). This preference suggests the Router predominantly favored precision-focused methods (keyword and structured schema queries) over purely semantic or summary-based approaches.
- Moderate Use of Knowledge Graph and Vector Engines: Knowledge Graph (ID 4, 56%) and Vector Engine (ID 0, 47%) were moderately employed, implying their importance as supplementary retrieval methods, particularly for nuanced or relationally complex queries. These engines likely provided complementary information where keyword or structured schema approaches might have been insufficient.
- Minimal Utilization of Summary Engine: Despite achieving notably high accuracy when used, the Summary engine (ID 2) was rarely selected (only 5%). This possibly reflects an intentional strategy by the Router - reserving summarization for queries specifically demanding high-level synthesis rather than detailed retrieval, or potentially reflecting constraints such as computational cost or response latency.

The selection patterns suggest that Router logic emphasizes:

- Precision and Speed: Frequent reliance on keyword and schema-based engines reflects a design preference for precise and rapid query responses, ideal for clearly defined factoid or structured queries.
- Contextual Adaptation: Moderate integration of semantic embedding (Vector) and Knowledge Graph approaches underscores the Router's adaptive nature, balancing efficiency with the necessity of rich semantic understanding for more ambiguous or complex queries.
- Strategic Use of Summarization: Limited Summary engine usage indicates a specialized role, suggesting potential for strategic expansion - particularly in cases benefiting from comprehensive context aggregation or nuanced understanding where traditional retrieval methods falter.

5.3 Cost and Efficiency Comparison: Multi-Agent vs. Single-Agent LLM Engines

Table 4: Multi-Agent vs Single-Agent Performance - Cost and Time per Query for Gpt-4.1

Engine Description	Avg LLM calls per query	Avg Input Tokens Per call	Avg Output Tokens Per call	Cost ($) Per call	Time (sec) Per call
Keyword Engine (Exact and keyword match search)	2	6648	143	0.01444	316
Property Graph Engine (Schema-based queries)	3.6	393	53	0.00121	177
Knowledge Graph Engine (Triplet-based facts)	1.5	3330	50	0.00706	178

Engine Description	Avg LLM calls per query	Avg Input Tokens Per call	Avg Output Tokens Per call	Cost ($) Per call	Time (sec) Per call
Vector Engine (Semantic embedding search)	1	461	126	0.00193	317
Summary Engine (LLM-generated summaries)	92	52718	525	0.1096	390
Router Engine (3 models selected by LLM)	12	11602	303	0.02563	257

Table 4 presents a comparative analysis of resource usage, efficiency, and cost for each retrieval engine, highlighting the distinctions between single-agent and multi-agent (Router) approaches. Lightweight engines like Property Graph and Vector demonstrate minimal token consumption and cost, making them attractive for rapid, low-overhead queries. In contrast, the Router engine - which orchestrates three retrieval methods per query - incurs higher resource usage (averaging 11,602 input tokens and $0.0256 per query) due to its ensemble execution design. These figures represent the cumulative resource demands of concurrently running three specialized pipelines, each optimized for different query facets.

Despite the increased overhead, the Router engine achieves markedly higher answer robustness and reliability than any single-engine baseline, with the notable exception of the Summary engine. While the Summary engine consistently yields the most comprehensive and accurate answers, it does so at a substantial computational cost, requiring an average of 92 LLM calls and over 52,000 input tokens per query - making it both the slowest and most expensive method evaluated.

Observability shows the Summary engine is rarely chosen by the Router, reserved for the most complex queries. This reflects a deliberate trade-off: although more resource-intensive, the Router delivers superior answer quality through multi-engine verification. Table 4 highlights the key insight-adaptive orchestration balances efficiency and reliability by using lightweight engines for routine queries and invoking heavier pipelines only when complexity demands it.

6. Conclusion

Orchestrating multiple retrieval pipelines and LLMs via a graph-based agentic framework enables high-precision, scalable, and trustworthy question answering over complex movie data. Our system demonstrates robust performance and sets the stage for autonomous, explainable movie intelligence - unlocking new applications in analytics and semantic search. This study demonstrates the transformative impact of agentic, multi-engine LLM frameworks for retrieval-augmented generation (RAG) in complex domains such as movie question answering. By orchestrating multiple specialized retrieval engines - including summarization, keyword, vector, knowledge graph, and property graph methods - within a unified, agentic workflow, we harness the unique strengths of each approach while dynamically adapting to the nuances of individual queries. A key innovation of our approach is the automatic construction of knowledge graphs in both triplet-based (KG) and property-based (PG) forms. Leveraging prompt engineering and advanced LLM capabilities, our system can extract semantic triples (subject, predicate, object) as well as rich property graphs from raw, unstructured data - without manual schema engineering. This dual-graph strategy empowers the pipeline to represent both relational knowledge and complex entity properties, providing a solid foundation for diverse, programmatic, and natural language queries. In the future, we are planning to integrate multi-model models to integrate images into our KG (Kaplunovich, 2024).

Our work demonstrates the power of LangGraph's concurrent, modular architecture, where retrieval engines operate in parallel for fast, diverse information access and robust answer synthesis. A dynamic router selects the most relevant engines for each query. Deep observability - enabled by a shared node class and persistent DynamoDB logging - provides transparent, fine-grained insights for auditing, diagnostics, and continuous optimization. Empirical results confirm that summarization-based retrieval consistently yields the highest verification scores, highlighting the value of LLM synthesis when grounded in rich context. Score distributions and threshold analyses expose each engine's reliability profile, enabling adaptive strategies that balance precision, coverage, and scalability in LLM-driven reasoning.

Ethics Declaration: Ethical clearance was not required for this research as it did not involve human participants, animal subjects, or sensitive data.

AI Declaration: Large Language Model (LLM) tools, specifically OpenAI's ChatGPT, were used for Grammar and spelling check.

References

Carta, S., Fenu, G., Malchiodi, G. and others (2023) "Iterative zero-shot LLM prompting for knowledge graph construction", arXiv, 2307.01128.

Han, J., Yang, M., Liang, Y. and others (2023) "PIVE: Prompting with iterative verification improving graph-based generative capability of LLMs", arXiv, 2305.12392.

Kaplunovich, A. (2024) "Illuminating the Shadows-Challenges and Risks of Generative AI in Computer Vision for Brands.", ECCV 2024 Workshop The Dark Side of Generative AIs and Beyond.

Kaplunovich, A. (2025) "Advancing LLM agents for code generation: observability, orchestration, reliable performance", Proceedings of ICCNS 2025.

Karpukhin, V., Oguz, B., Min, S. and others (2020) "Dense passage retrieval for open-domain question answering", Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 6769-6781.

Lewis, P., Perez, E., Piktus, A. and others (2020) "Retrieval-augmented generation for knowledge-intensive NLP tasks", Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459-9474.

Liang, P., Bommasani, R., Yatskar, M. and others (2022) "Holistic evaluation of language models", arXiv, 2211.09110.

Liu, J. (2022) "LlamaIndex: a data framework for LLM applications", GitHub. Available at: https://github.com/jerryjliu/llama_index (Accessed: 27 July 2025).

OpenAI (2023) "GPT-4 technical report", arXiv, 2303.08774.

Tran, K.-T., Li, J., Wang, Q. and others (2025) "Multi-agent collaboration mechanisms: a survey of LLMs", arXiv, 2501.06322.

Wang, Y., Chen, J., Zhang, L. and others (2024) "Knowledge graph prompting for multi-document question answering", Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 17.

Wu, S., Li, P., Zhao, T. and others (2024) "Retrieval-augmented generation for natural language processing: a survey", arXiv, 2407.13193.

Ye, H., Zhang, N. and Chen, H. (2022) "Generative knowledge graph construction: a review", Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1-17.

Zhu, Y., Zhang, H., Li, M. and others (2024) "LLMs for knowledge graph construction and reasoning: recent capabilities and future opportunities", World Wide Web, 27(5): 58.