RAG Application Using Knowledge Graph and Vector Search

Molly Iverson[1], **Ethan Villalovoz**[1], **Chandler Juego**[1], **Adam Shtrikman**[1]
Vikas Aditya[2], **Parteek Kumar**[1]

[1]Washington State University [2]HackerEarth

knowledge-graph-rag.github.io

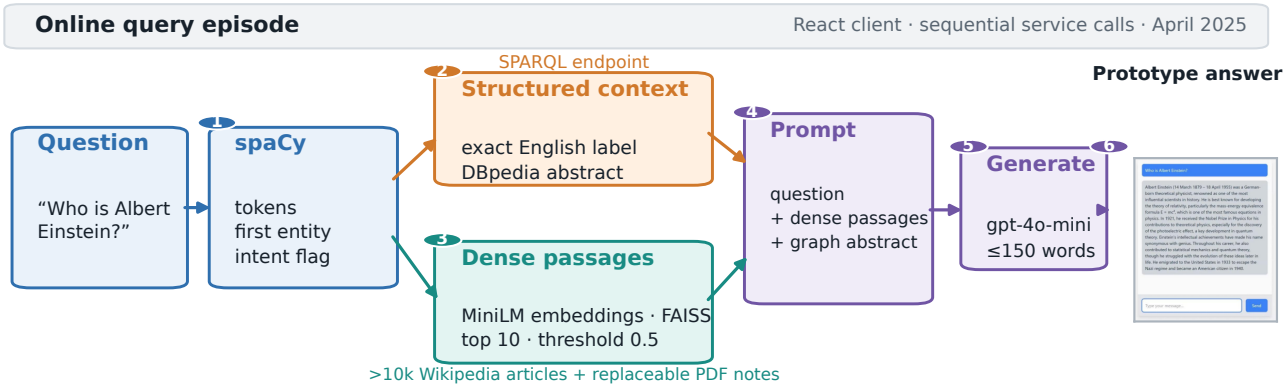

Figure 1. **System overview.** A question is analyzed for tokens, an entity, and an intent flag. The structured path requests an English DBpedia abstract; the dense path searches Wikipedia passages or replaceable PDF notes. The archived April 2025 client invokes the services sequentially and passes both contexts to the generator. The interface crop is authentic prototype evidence; it is not an answer-correctness judgment.

Abstract

Retrieval-augmented generation can ground an answer in retrieved text, while a knowledge graph can supply complementary entity-centered context. We present a deployable question-answering prototype that combines both sources: a dense retriever searches more than 10,000 Wikipedia articles or replaceable PDF course notes, and a structured retriever requests an English abstract from DBpedia before an OpenAI model generates the response. The system couples a React interface to a FastAPI backend and packages the application with Docker. We evaluate the artifact at the level supported by the preserved project record: twelve representative subsystem cases are reported as passing, a separate continuous-integration artifact records sixteen passed tests and four warnings, and three calls of one development prompt yield a final recorded mean of 7.615 seconds. These measurements establish software behavior and a historical latency trace; they do not establish retrieval recall, factuality, or superiority over single-retriever baselines. The result is an evidence-calibrated account of the implemented system, its deployment path, and the evaluation required for future comparative claims.

1. Introduction

Large language models can produce fluent answers without exposing what information supported them. Retrieval-augmented generation (RAG) addresses part of this problem by placing retrieved material in the generation context. Dense passage retrieval is useful when a question and a source passage are semantically similar, but entity-oriented questions also admit a structured path: an identified entity can be resolved against a knowledge graph and its description supplied alongside dense passages. This project explores that practical combination in a complete web application.

The resulting KG–Vector RAG prototype accepts a natural-language question, extracts an entity with spaCy,

obtains an English DBpedia abstract through SPARQL, and retrieves passages from a normalized FAISS index using `all-MiniLM-L6-v2`. The client then sends the question and both retrieved contexts to `gpt-4o-mini`. Figure 1 summarizes the implemented query episode and deliberately distinguishes the logical evidence paths from the sequential calls made by the archived React client.

This paper makes three scoped contributions:

- an end-to-end, modular implementation of structured and dense retrieval behind one conversational interface;
- a reproducible description of the data preparation, retrieval, prompt assembly, and Docker deployment paths; and
- an evidence-calibrated validation record that separates documented software tests and timing observations from unmeasured answer quality.

The contribution is a systems artifact, not a new retrieval algorithm or a claim that hybrid retrieval is more accurate. The original project did not preserve a vector-only baseline, graph-only baseline, standardized QA benchmark, or blinded human evaluation. We state those boundaries explicitly so that the architecture, implementation, and measurements can be interpreted without importing evidence that the project did not collect.

2. Problem Setting and Evidence Scope

Let q be a user question. The system seeks two context sets: structured context $C_{\mathrm{KG}}(q)$ and dense passage context $C_{\mathrm{VS}}(q)$. It forms a generation request

$$p(q) = [q;\, C_{\mathrm{KG}}(q);\, C_{\mathrm{VS}}(q)], \tag{1}$$

where brackets denote prompt concatenation rather than a learned fusion operator. The generated response is $a = G(p(q))$. This formulation describes the implemented information flow; it does not imply that either context source improves a without a controlled comparison.

2.1. Interaction contract

The capstone report specified four user-facing use cases: write a query, submit it, receive a response, and review messages in the current session. Rather than reproduce the UML diagram, Fig. 2 expresses them as a query episode with a nominal state sequence and recovery boundaries. This presentation makes the relationship between user action, system action, and visible failure state explicit.

2.2. Requirements versus observations

The report set a 15-second response-time target, support for more than 10,000 Wikipedia articles, maintainability, storage, security, and 99% uptime. These are design requirements. Only the dataset scale, functional test record, and small timing log have preserved observations. In particular, the project did not measure uptime, stress scalability, security properties, or the statistical quality of generated answers. We therefore use "target" for a requirement and "recorded" for a preserved observation throughout.

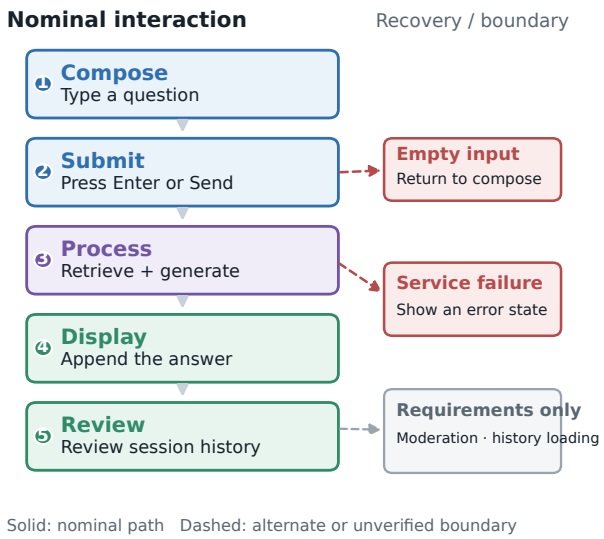

Figure 2. **Query episode and recovery paths.** Solid arrows show the nominal interaction supported by the prototype. Empty input and service failure have visible handling in the archived client. Content moderation and infinite-history loading appeared in requirements prose but are not treated as verified implementation behavior.

The primary evidence sources are the April 2025 capstone report and the archived implementation at commit `1ad5cc08cbebdeae655cad626393364b2f476556`. The current project website is a presentation and navigation layer; later interface work is not used to rewrite the behavior of the 2025 prototype.

3. System Architecture

The system separates a browser client from three FastAPI service groups: natural-language processing and generation, DBpedia access, and vector search. This modularity allowed handlers to be tested independently and packaged behind a small HTTP interface.

3.1. Structured context

The NLP route tokenizes q, detects named entities, and reports an intent flag. For the structured path, the implementation uses the first detected entity to construct an exact English-label SPARQL query. The DBpedia route submits that query to the public endpoint and requests an English abstract [1, 5]. If no binding contains an abstract, the client retains an empty or "No abstract available" context rather than preventing dense retrieval and generation. Exact-label

Table 1. Implemented retrieval configuration. Values are implementation constants, not tuned experimental results.

Component	Configuration
Entity analysis	spaCy; first detected entity
Graph query	DBpedia SPARQL; exact English label; English abstract
Embeddings	`all-MiniLM-L6-v2`
Vector index	normalized FAISS `IndexFlatIP`
Retrieval	top 10; similarity threshold 0.5
Generation	`gpt-4o-mini`; response request ≤ 150 words

matching keeps the implementation simple but creates a brittle entity-linking boundary for aliases, multiple entities, and relational questions.

3.2. Dense context

Offline, Wikipedia text is cleaned and stored in Parquet. PDF course notes can be extracted into a replaceable custom dataset. The system encodes passages with `all-MiniLM-L6-v2` from SentenceTransformers [9]. Given passage embedding x_i, the index stores $\hat{x}_i = x_i/\|x_i\|_2$ in a FAISS `IndexFlatIP` [6]. A query embedding is normalized in the same way, so inner-product search implements cosine similarity:

$$s(q, x_i) = \hat{x}_q^{\mathsf{T}} \hat{x}_i. \tag{2}$$

The route requests the ten highest-scoring entries, removes invalid indices, and filters scores below 0.5 before returning source text. These are fixed prototype parameters, not values selected through a reported tuning study.

3.3. Prompt assembly and generation

The React client first awaits NLP, then the DBpedia request when a SPARQL query exists, then dense retrieval, and finally generation. It sends the raw question, retrieved passages, and the DBpedia text to the NLP service's generation route. The OpenAI handler instructs `gpt-4o-mini` to answer from the supplied information in at most 150 words [8]. The prompt requests an informative answer but does not produce source-level citations or expose the selected passages in the 2025 interface.

4. Implementation and Deployment

Frontend. The interface is a React single-page chat application [7]. It maintains the question, message list, and loading state locally. Submitting a non-empty question appends the user message, clears the input, enables the loading indicator, and begins the service sequence. A successful generation appends the answer; an exception appends an error message. The scrollable message panel supports review of the current session. The archived client does not persist a conversation across sessions.

Backend. FastAPI exposes routes for NLP, DBpedia, vector search, and generation. CORS is configured for local frontend origins. Each retrieval handler has a narrow responsibility: construct a graph query, execute it, embed and search a dense index, or assemble a generation request. This organization supports isolated tests, although the client remains the orchestration layer.

Data preparation. The repository contains extraction scripts for Wikipedia and PDF inputs, Parquet text artifacts, NumPy embedding arrays, and a serialized FAISS index. The report describes more than 10,000 Wikipedia articles and a course-notes dataset assembled from replaceable PDFs. The dense index must be available locally before serving queries; DBpedia and OpenAI require network access.

Packaging. The backend and frontend are packaged as separate containers and started with Docker Compose [2, 3]. The runtime requires an OpenAI API key supplied by the operator; the project does not distribute credentials. GitHub Actions builds and runs tests on repository changes [4]. This delivery path was intended for local demonstration and handoff rather than a measured multi-user production deployment.

Reproducibility boundary. The archived commit records source code, test code, small processed data, and documentation. Large embedding and index artifacts are external setup dependencies described by the project documentation. Live DBpedia results, OpenAI responses, network conditions, and historical hardware are not frozen, so exact answer text and latency are not expected to replay deterministically.

5. Prototype Behavior and Use Cases

Figure 3 shows the documented interaction sequence. The interface keeps the question visible while processing, presents a loading state, and appends the generated response to the chat. The example asks about Albert Einstein. It demonstrates data flow and user-visible state transitions; the screenshot alone cannot establish that every statement in the answer is correct or attributable to a particular retrieved source.

The default Wikipedia corpus supports general factual questions. The PDF pipeline supplies course notes as a replaceable custom corpus, illustrating how the same interface can be adapted to a bounded document collection. The report also proposed private organizational documents as a future use case. Such use would require access control, provenance display, data-governance review, and a domain-specific evaluation that are outside the present artifact.

The project was delivered as a local Docker Compose application for client and course demonstration. It was not

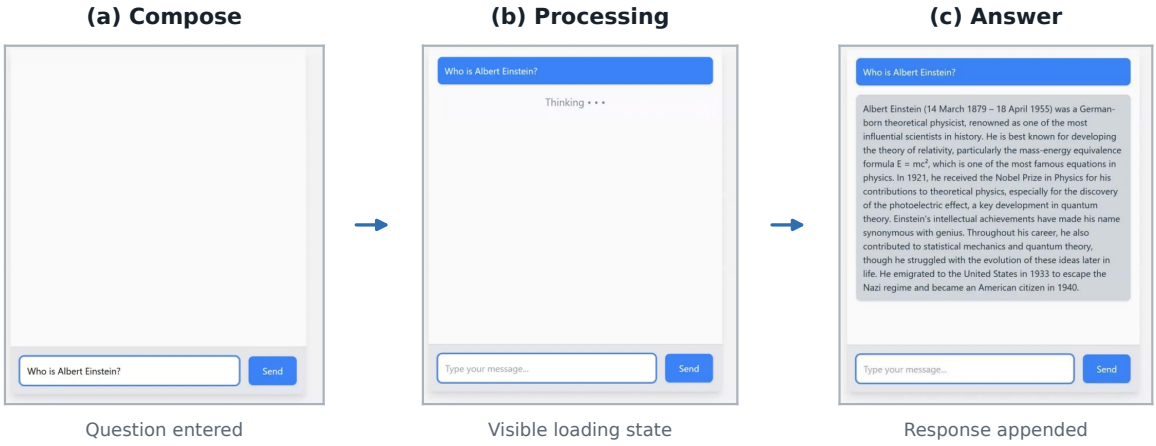

Figure 3. **Archival prototype sequence. (a)** The user composes a question. **(b)** The interface exposes a loading state while the sequential service calls execute. **(c)** The response is appended to the current-session chat. The screenshots are cropped from the original report without generative editing.

integrated into the client's production codebase, and no multi-user deployment study was conducted. This distinction is important: containerization demonstrates a repeatable packaging path, not operational reliability at production scale.

6. Engineering Validation

The preserved evaluation is software-engineering validation. It combines unit-level handler cases, integration and manual checks described in prose, a CI screenshot, and a small timing record. It is not an information-retrieval or question-answering benchmark.

6.1. Documented functional cases

The report's test table lists twelve representative cases, all marked as passing: six for NLP, two for DBpedia, three for vector search, and one for the LLM handler. The cases cover normal and edge entity extraction, tokenization, harmful-intent detection, graph-query execution and no-result handling, vector indexing and returned indices, a retrieval timing condition, and response-length compliance. Figure 4 preserves that distribution. The table's "retrieval accuracy" label refers to returning expected document indices in test data; it is not corpus-level recall, precision, or answer accuracy.

The report also describes pairwise-to-system integration checks, weekly client demonstrations, and manual functional testing. However, it does not preserve a scored acceptance rubric, annotator protocol, or raw judgments. We therefore report these as development procedures rather than quantitative outcomes.

6.2. Historical timing record

The repository preserves timing for the prompt "Who is Alan Turing?" at three development snapshots, with three calls per snapshot. The initial calls have totals of 6.108, 8.543, and 4.757 seconds, but the generator was not functioning. The intermediate snapshot has a 6.287-second arithmetic mean. The final snapshot records 7.780, 7.978, and 7.088 seconds, for a mean of 7.615 seconds. This final mean falls below the project's 15-second target for these three calls.

Figure 5 shows every preserved total and the final component means. Vector retrieval is the largest recorded final component at 4.134 seconds, followed by generation at 2.080 seconds, DBpedia at 1.368 seconds, and NLP at 0.033 seconds. Component values sum approximately to the recorded totals, up to logging precision.

The timing record is too small and insufficiently controlled to establish population latency or causality for the intervening code changes. Its useful role is diagnostic: it identifies vector retrieval as the dominant component in the final logged snapshot and motivates profiling, caching, and artifact loading improvements.

7. Limitations and Responsible Use

Retrieval and answer quality. The project did not evaluate exact match, retrieval recall, factuality, citation correctness, calibration, or human preference. Consequently, this paper does not claim that adding DBpedia improves answers over vector-only RAG. A suitable follow-up would freeze a question set and corpus, compare graph-only, vector-only, and combined variants, score retrieval and generation separately, and report uncertainty across repeated runs.

Entity handling. Using the first spaCy entity and an exact DBpedia label is fragile. Questions with multiple entities, aliases, ambiguous names, or relations can retrieve no graph context or the wrong context. Entity linking, candidate rank-

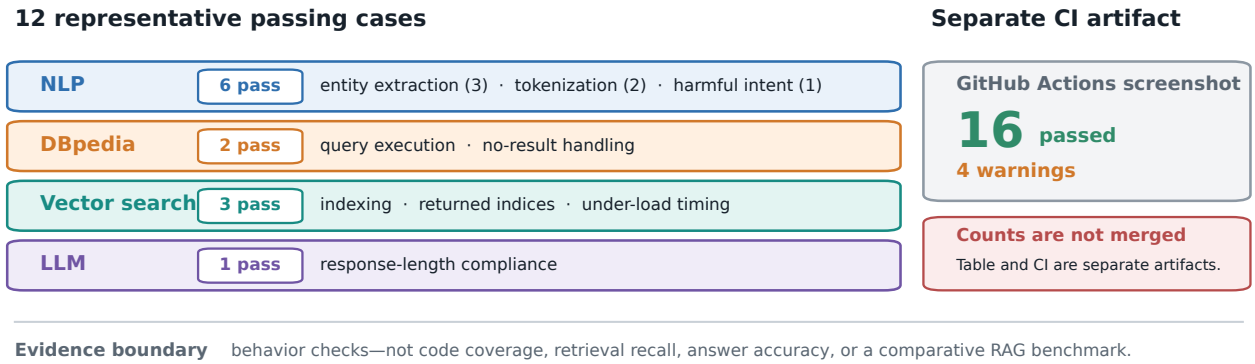

Figure 4. **Validation record.** The capstone report documents twelve representative passing cases across four subsystems. A separate CI screenshot records sixteen passed tests and four warnings. Because the two artifacts do not preserve a one-to-one mapping, their counts are shown separately. Neither artifact measures answer factuality or comparative retrieval quality.

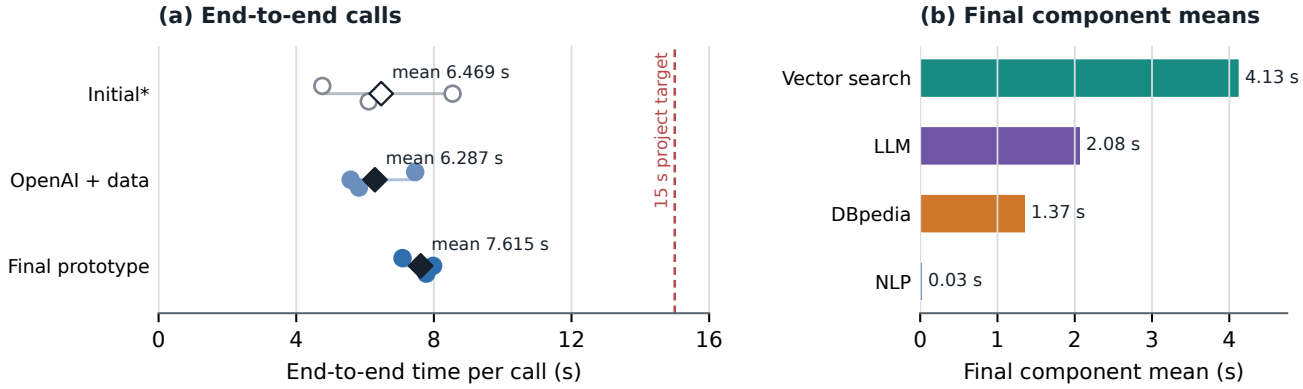

Figure 5. **Historical response-time record for one prompt. (a)** Three logged end-to-end calls at each development snapshot; diamonds show arithmetic means. Initial totals exclude a functioning generator. **(b)** Final component means. Hardware, warm-up, network, and runtime controls were not preserved, so the plot is a development record, not an ablation, uncertainty estimate, or latency guarantee.

ing, and relation-aware queries are natural extensions, but none was evaluated here.

Provenance. The 2025 interface presents only the final response. Users cannot inspect which passages or graph facts entered the prompt. This limits auditability and makes it difficult to distinguish a retrieval error from a generation error. A future interface should expose source titles, passage snippets, graph entities, scores, and failure states without implying that retrieval alone verifies a claim.

Live services and privacy. DBpedia and OpenAI are network services whose behavior and latency can change. Questions and retrieved context sent to an external model may be sensitive in private-document deployments. Operators should apply current provider terms, data-retention controls, authentication, transport security, and organizational review before using such data. The capstone prototype is not evidence that these production safeguards have been validated.

Moderation. The report includes harmful-intent detection and a content-moderation requirement. Keyword or flag-based handling is not a comprehensive safety evaluation, and historically or academically legitimate questions can contain sensitive terms. Moderation behavior should be tested with an explicit policy, representative cases, false-positive analysis, and a user-visible appeal or explanation path.

8. Conclusion

We presented a deployable RAG prototype that combines an entity-centered DBpedia abstract with dense Wikipedia or PDF passages before generation. The system contributes an integrated implementation, a clear packaging path, and a preserved engineering-validation record. Recasting the capstone artifact as a technical paper also clarifies what remains unknown: the project does not yet show that hybrid retrieval improves answer quality, nor does it establish production reliability. The next step is a controlled study with frozen

data, explicit baselines, source provenance, and repeatable timing.

Author contributions. Molly Iverson, Ethan Villalovoz, Chandler Juego, and Adam Shtrikman jointly designed, implemented, tested, documented, and delivered the capstone system. Vikas Aditya provided project guidance and client feedback, and Parteek Kumar provided capstone instruction and supervision. The original report assigns component and project-management responsibilities across the team; this adaptation retains the supplied author order and does not infer a more granular contribution taxonomy than the preserved record supports.

References

[1] DBpedia Association. About DBpedia. https://www.dbpedia.org/about/, 2025. Accessed April 2025. 2

[2] Docker, Inc. Docker documentation. https://docs.docker.com/, 2025. Accessed April 2025. 3

[3] Docker, Inc. Docker compose documentation. https://docs.docker.com/compose/, 2025. Accessed April 2025. 3

[4] GitHub. Github actions documentation. https://docs.github.com/actions, 2025. Accessed April 2025. 3

[5] Steve Harris and Andy Seaborne. SPARQL 1.1 query language. W3c recommendation, World Wide Web Consortium, 2013. 2

[6] Meta AI Research. FAISS: A library for efficient similarity search and clustering of dense vectors. https://github.com/facebookresearch/faiss, 2025. Accessed April 2025. 3

[7] Meta Open Source. React documentation. https://react.dev/, 2025. Accessed April 2025. 3

[8] OpenAI. Openai api documentation. https://platform.openai.com/docs/, 2025. Accessed April 2025. 3

[9] UKP Lab. Sentencetransformers documentation. https://sbert.net/, 2025. Accessed April 2025. 3

A. Supplementary Evidence Record

This supplement replaces the original report's screenshots of numeric output and source code with native, searchable tables. It adds no experiments.

A.1. Documented test cases

Table 2. Representative cases transcribed from the capstone report. "Pass" is the report's recorded result. The labels describe software behavior, not a standardized retrieval or QA metric.

Subsystem	Case	Recorded observation	Result
NLP	Entity extraction—normal	Named entities extracted from input queries	Pass
NLP	Entity extraction—edge cases	Expected multi-word or ambiguous entities extracted	Pass
NLP	Missing entities	No entities extracted for simple no-entity input	Pass
NLP	Tokenization—normal	Tokens match expected segmentation	Pass
NLP	Tokenization—special characters	Output matches expected segmentation without punctuation	Pass
NLP	Harmful-intent detection	`is_harmful` returns true for the tested violent-intent phrase	Pass
DBpedia	Query execution—normal	Query returns relevant information	Pass
DBpedia	No-result handling	No-result condition returned instead of an application error	Pass
Vector search	Indexing	Embeddings indexed and index returns vectors	Pass
Vector search	Retrieval result indices	Expected document indices returned on test data	Pass
Vector search	Performance under load	Retrieval remained within the report's acceptable response condition	Pass
LLM	Response-length compliance	Response stayed within the configured length limit	Pass

The report's CI screenshot separately records **16 passed tests and 4 warnings**. The project record does not preserve a mapping that reconciles those sixteen executions with the twelve representative rows above; the counts are therefore not summed or treated as a coverage measure.

A.2. Historical timing values

Table 3. Preserved response-time log for "Who is Alan Turing?" Values are milliseconds. Means are arithmetic means of the three recorded calls. The initial generator was not functioning.

Snapshot / component	Call 1	Call 2	Call 3	Mean	Status
Initial—NLP	552.30	346.40	552.00	483.57	recorded
Initial—DBpedia	1137.30	1037.40	1231.70	1135.17	recorded
Initial—vector search	4418.30	7159.30	2973.20	4850.27	recorded
Initial—LLM	–	–	–	–	not functioning
Initial—total	**6107.90**	**8543.10**	**4756.90**	**6469.30**	incomplete
OpenAI + data—NLP	341.40	34.80	345.30	240.50	recorded
OpenAI + data—DBpedia	1149.70	931.70	2112.30	1397.90	recorded
OpenAI + data—vector search	1924.70	1747.40	1753.50	1808.53	recorded
OpenAI + data—LLM	2405.30	2866.90	3245.50	2839.23	recorded
OpenAI + data—total	**5821.90**	**5581.30**	**7457.50**	**6286.90**	complete
Final—NLP	36.40	27.40	36.40	33.40	recorded
Final—DBpedia	1498.20	1392.70	1212.30	1367.73	recorded
Final—vector search	3717.20	4672.20	4011.20	4133.53	recorded
Final—LLM	2527.00	1884.80	1827.29	2079.70	recorded
Final—total	**7779.70**	**7977.60**	**7087.90**	**7615.07**	complete

The timing document preserves the prompt and values but not machine specifications, operating-system state, model-service version, network conditions, cache state, or warm-up procedure. The values should not be used as a cross-system comparison.

A.3. Figure provenance and integrity

Figures 1, 2, 4, and 5 are regenerated from archived code, the report's test table, and the preserved timing Markdown. Their editable Python source and frozen CSV inputs are included with this manuscript. Figure 3 uses the original report's interface screenshots with cropping and vector labels only; no generative image editing was applied. The project website and the two reference papers informed presentation hierarchy but supplied no system evidence.