Optimizing the Interface Between Knowledge Graphs and LLMs for Complex Reasoning*

Vasilije Marković[1], Lazar Obradović[1], Laszlo Hajdu[1,2,3], and Jovan Pavlović[3]

[1]Cognee Inc.
[2]Innorenew CoE
[3]University of Primorska, FAMNIT

Abstract

Integrating Large Language Models (LLMs) with Knowledge Graphs (KGs) results in complex systems with numerous hyperparameters that directly affect performance. While such systems are increasingly common in retrieval-augmented generation, the role of systematic hyperparameter optimization remains underexplored. In this paper, we study this problem in the context of Cognee, a modular framework for end-to-end KG construction and retrieval. Using three multi-hop QA benchmarks (HotPotQA, TwoWikiMultiHop, and MuSiQue) we optimize parameters related to chunking, graph construction, retrieval, and prompting. Each configuration is scored using established metrics (exact match, F1, and DeepEval's LLM-based *correctness* metric). Our results demonstrate that meaningful gains can be achieved through targeted tuning. While the gains are consistent, they are not uniform, with performance varying across datasets and metrics. This variability highlights both the value of tuning and the limitations of standard evaluation measures. While demonstrating the immediate potential of hyperparameter tuning, we argue that future progress will depend not only on architectural advances but also on clearer frameworks for optimization and evaluation in complex, modular systems.

1 Introduction

Large Language Models (LLMs), based on Transformer architectures [32], have demonstrated strong performance across a wide range of natural language processing tasks, including open-domain question answering, summarization, and

*This is a preliminary version. A revised and expanded version is in preparation.

generation. Their impact has been felt across diverse application areas, from healthcare and finance to legal and scientific domains [28, 35, 4, 1]. While these models store vast amounts of information in their parameters [26, 27, 23], they are also prone to producing confident but incorrect outputs [21, 17]. In addition, they lack an efficient mechanism for updating or extending knowledge without retraining.

Retrieval-Augmented Generation (RAG) has emerged as a standard approach to mitigate these issues [10]. In typical RAG pipelines, a dense retriever selects relevant textual context for a given query, and the retrieved content is appended to the query before being processed by the LLM. This design improves factuality and allows models to reference external sources. However, standard RAG systems often struggle with questions that involve multi-step reasoning or require structured access to relational knowledge. In such cases, relying solely on dense or sparse document retrieval is insufficient [34, 7].

To address these limitations, hybrid approaches that integrate knowledge graphs into RAG workflows have gained attention. These systems, sometimes referred to as GraphRAG, use graphs to represent relational structures and support retrieval based on symbolic queries or multi-hop graph traversal [25, 14]. Graph-based retrieval augments LLMs with access to explicit, structured context and has shown promise in tasks requiring deeper reasoning.

One challenge that persists across both classical and graph-based RAG systems is hyperparameter sensitivity. The performance of these pipelines depends heavily on a wide range of configuration choices, including chunk size, retriever type, top-k thresholds, and prompt templates. As pipelines grow more modular and sophisticated, the number of tunable parameters increases, and their interactions become more complex. While hyperparameter optimization has been explored in standard RAG systems, its role in graph-enhanced pipelines remains underexamined.

This paper addresses that gap. We present a structured study of hyperparameter optimization in graph-based RAG systems, with a focus on tasks that combine unstructured inputs, knowledge graph construction, retrieval, and generation. Our experiments use Cognee, an open-source framework that supports end-to-end graph-based memory construction and retrieval. Cognee's modularity allows for clean separation and independent configuration of pipeline components, making it well-suited for controlled optimization studies.

We evaluate on three established multi-hop QA benchmarks: HotPotQA, TwoWikiMultiHop, and Musique. Each configuration is scored using one of three metrics: exact match (EM), token-level F1, or correctness. The correctness score is computed using DeepEval, an LLM-based grading tool that evaluates answer plausibility against the gold reference. Our study demonstrates that even modest parameter changes can lead to measurable improvements, but also that metric choice and task characteristics strongly influence outcomes.

The next section reviews related work on retrieval-augmented generation, graph-based systems, and hyperparameter optimization. In Section 3, we describe the Cognee framework and its architecture for knowledge graph construction and retrieval. Section 4 outlines the hyperparameter optimization setup,

including the parameter space and optimization method. Section 5 details the experimental design, benchmarks, and evaluation metrics. Section 6 presents the results and discusses their implications. We conclude with a summary of findings and directions for future work in Section 7.

2 Background and Related Work

We review key developments relevant to our work, focusing on retrieval-augmented generation (RAG), multi-hop and graph-based question answering, and recent advances in hyperparameter optimization for LLM pipelines. Particular attention is given to methods that combine structured retrieval with neural generation and those that treat pipeline configuration as an optimization problem.

Advances and Challenges in RAG Systems Retrieval-augmented generation (RAG) systems extend language models with a retrieval module to ground outputs in external knowledge [20, 13]. This basic two-stage architecture has become the de facto standard, with numerous refinements proposed over time [10]. Recent work includes Self-RAG [2], which enables LLMs to reflect on their outputs and dynamically trigger retrieval, and CRAG [36], which filters low-confidence documents using a retrieval evaluator and escalates to web search when needed. Comprehensive surveys summarize the state of the field and its variants [11, 8, 42].

Multi-Hop Question Answering Multi-hop QA extends standard QA by requiring reasoning over multiple documents. Early datasets like HotPotQA [37] crowdsourced such questions over Wikipedia. 2WikiMultiHopQA [16] improves on this by leveraging Wikidata relations to enforce structured, verifiable reasoning paths. MuSiQue [31] takes a bottom-up approach, composing multi-step questions from single-hop primitives and filtering out spurious shortcuts, offering a more robust benchmark for compositional reasoning.

Knowledge Graph Question Answering KGQA systems answer questions via structured reasoning over graphs [39, 41, 29, 38], increasingly integrating LLMs to bridge symbolic and neural reasoning [24, 19]. RoG [22] prompts LLMs to generate abstract relation paths that are grounded via graph traversal before final answer generation. Other work includes trainable subgraph retrievers [40] and decomposed logical reasoning chains over subgraphs [6], demonstrating measurable gains in both interpretability and performance.

GraphRAG GraphRAG generalizes RAG to arbitrary graph structures, extending its use beyond knowledge bases [25, 14]. Early systems like Microsoft's summarization pipeline [7] use LLMs to build knowledge graphs, partition them with community detection [30], and summarize each component. Other variants use GNNs with subgraph selection [15], graph-traversal agents [34], or Personalized PageRank over schemaless graphs [12]. These systems span a wide range of

tasks but share a common structure: dynamic subgraph construction followed by prompt-based reasoning.

Hyperparameter Optimization in RAG Optimizing RAG systems requires balancing retrieval coverage, generation accuracy, and resource constraints. Recent work applies Bayesian optimization under budget limits [33], formulates context usage as a tunable variable [18], and introduces full-pipeline tuning via reinforcement learning [5, 9]. Multi-objective frameworks have also emerged to trade off accuracy, latency, and safety [3]. While methodologically diverse, all aim to expose and control the critical degrees of freedom in modern RAG pipelines.

3 Automated Knowledge Graph Construction: Cognee

Cognee is an open-source framework for end-to-end knowledge graph (KG) construction, retrieval, and completion. It supports heterogeneous inputs (e.g., text, images, audio) from which it extracts entities and relations, possibly with the support of an ontology schema. The extraction process runs in containerized environments and is based on tasks and pipelines, with each stage extensible via configuration or code.

The default pipeline includes ingestion, chunking, large language model (LLM)-based extraction, and indexing into graph, relational, and vector store backends. Downstream of indexing, Cognee provides built-in components for retrieval and completion. A unified interface supports vector search, symbolic graph queries, and hybrid graph–text methods. Completion builds on the same infrastructure, enabling both prompt-based LLM interaction and structured graph queries.

Cognee also includes a configurable evaluation framework for benchmarking retrieval and completion workflows. The framework is based on multi-hop question answering, providing a structured evaluation setting for graph-based systems using established benchmarks (HotPotQA, TwoWikiMultiHop). Evaluation proceeds sequentially through distinct phases. It begins with corpus construction, followed by context-conditioned answering that leverages the retrieval and completion components. Answers are then compared to gold references and graded using multiple metrics. The final output includes confidence-scored performance reports. For more technical details, including the ingestion, retrieval, and evaluation architecture, see Appendix A.

Cognees's modularity enables targeted hyperparameter tuning across ingestion, retrieval, and completion stages. The evaluation framework provides structured, quantitative feedback, allowing the system as a whole to be treated as an objective function. This setup enables the direct application of standard hyperparameter optimization algorithms, which we describe in the following section.

Parameter	Description
Chunk size	Number of tokens per document segment used during graph extraction
Retriever type	Strategy used to retrieve context (text chunks or graph triplets)
Top-k	Number of retrieved items passed to the language model
QA prompt	Instruction template used for answer generation
Graph prompt	Template guiding entity and relation extraction during graph construction
Task getter	Configuration for dataset preprocessing and summary handling

Table 1: Parameters used during Dreamify optimization.

4 Hyperparameter Optimization Setup

This section describes the structure of the optimization process and the parameters explored during tuning. We first outline the end-to-end setup and optimization method, followed by a detailed description of the tunable parameters.

4.1 Optimization Framework

Cognee exposes multiple configurable components that influence retrieval and generation behavior. These include parameters related to preprocessing, retriever selection, prompt design, and runtime settings. While default values are often selected heuristically, their impact on performance is not always predictable. To evaluate the effect of these design choices systematically, we developed a hyperparameter optimization framework named Dreamify.

Dreamify treats the entire Cognee pipeline as a parameterized process. This includes ingestion, chunking, LLM-based extraction, retrieval, and evaluation. A single configuration defines the behavior of all stages. Each trial corresponds to a complete pipeline run, starting from corpus construction and ending with evaluation against a benchmark dataset. The output is a scalar score based on one of several metrics, such as F1, exact match, or LLM-based correctness. These metrics are computed as averages over all questions in the dataset and return values between 0 and 1.

The optimization is performed using a Tree-structured Parzen Estimator (TPE). This algorithm is well suited to the search space in question, which combines categorical and ordered integer-valued parameters. Grid search is not practical at this scale, and random search underperformed in early tests. While TPE was sufficient for our experiments, other optimization strategies remain open for future work.

The pipeline behavior is deterministic with respect to a fixed configuration,

though some components, such as LLM-generated graph construction, exhibit minor variation across runs. These differences do not materially affect overall evaluation scores within a single configuration. Trials are independent and reproducible. Further experimental details and results are presented in Section 5.

4.2 Tunable Parameters

The optimization process considers six core parameters that influence document processing, retrieval behavior, prompt selection, and graph construction. Each parameter affects how information is segmented, retrieved, or used during answer generation. Table 1 provides a summary; below we describe each parameter in more detail.

Chunk Size (`chunk_size`) This parameter controls the number of tokens used to segment documents before graph extraction. In the Cognee pipeline, it influences both the structure of the resulting graph and the granularity of context available during retrieval. The range used in this study (200–2000 tokens) was chosen based on preliminary testing to balance extraction accuracy, retrieval specificity, and processing time.

Retrieval Strategy (`search_type`) This parameter determines how context is selected for answer generation. The `cognee_completion` strategy retrieves text chunks using vector search and passes them directly to the language model. The `cognee_graph_completion` strategy retrieves knowledge graph nodes and their associated triplets by combining vector similarity with graph structure. Retrieved nodes are briefly described, and the surrounding triplets are formatted as structured text. The structured format of retrieved nodes and triplets emphasizes relational context and may support more effective multi-hop reasoning.

Top-K Context Size (`top_k`) This parameter sets the number of items retrieved per query. With `cognee_completion`, it controls the number of text chunks; with `cognee_graph_completion`, it controls the number of graph triplets. The retrieved context is passed to the language model for answer generation. In our experiments, values ranged from 1 to 20.

QA Prompt Template (`qa_system_prompt`) This parameter selects an instruction template used for answer generation. Templates differ in style and specificity, ranging from concise prompts to more detailed instructions that encourage justification or structured outputs. Prompt selection can influence both answer format and factual precision.

Prompt Templates (`qa_system_prompt`, `graph_prompt`) These parameters control the instruction templates used during answer generation and graph construction. For question answering, we evaluated three prompt variants differing

primarily in tone and verbosity. While the underlying instruction remained consistent, more constrained and direct prompts often produced outputs that aligned more closely with the expected answer format. This had a notable impact on evaluation scores, particularly for exact match and F1, though correctness scores were also affected to a lesser degree. For graph construction, three prompts were also tested, differing in how they guided the LLM to extract entities and relations from text—either in a single step or through more structured, incremental instructions. This choice influenced the granularity and consistency of the resulting graph structures used during retrieval.

Task Processing Method (`task_getter_type`) This parameter controls how question–answer pairs are preprocessed during evaluation. While the system can support arbitrary pipeline variants, we focus on two representative configurations. In the first, document summaries are generated during graph construction and made available to the retriever. In the second, summary generation is omitted.

5 Experimental Setup

We conducted a series of nine hyperparameter optimization experiments to evaluate the impact of configuration choices on Cognee's end-to-end performance. Each experiment corresponds to a distinct combination of benchmark dataset and evaluation metric. The datasets used were HotPotQA, TwoWikiMultiHop, and Musique. Each experiment targeted one of three metrics: exact match (EM), F1, or DeepEval's LLM-based correctness.

For each experiment, we created a filtered subset of the benchmark. Instances were randomly sampled and then manually reviewed prior to experimentation. We excluded examples that were ungrammatical, ambiguous, mislabeled, or unsupported by the provided context. Similar issues have been noted in prior literature. The resulting evaluation set consisted of 24 training instances and 12 test instances per dataset. This filtering step was performed once, before any tuning, to avoid bias or cherry picking.

Within each trial, the knowledge graph was constructed using all context passages from the training set. This produced a single merged graph per trial, which was then used to answer all training questions. The structure of the pipeline remained consistent across all datasets and metrics.

Each experiment consisted of 50 trials. In each trial, a configuration was sampled by the optimizer and executed as a full pipeline run, including ingestion, graph construction, retrieval, and answer generation. The selected metric was computed over all training questions, and the resulting score was used as the objective value for the trial. EM and F1 scores were computed deterministically. The DeepEval correctness score required a separate LLM-based evaluation step.

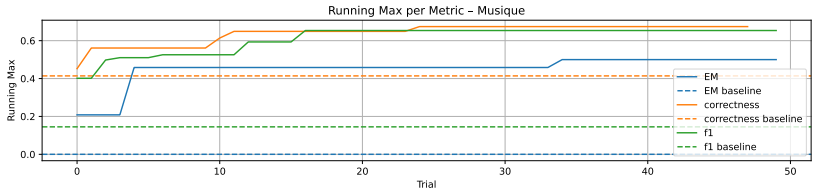
Trials were run sequentially without parallelization. Execution time per trial was approximately 30 minutes. Final results report performance on the test set using the best-performing configuration selected from training. In addition to

point estimates, we report confidence intervals computed using non-parametric bootstrap resampling over individual QA pairs.

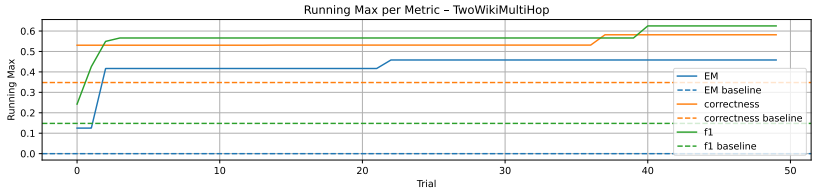
6 Results and Discussion

In this section, we summarize the outcomes of the optimization experiments, including training and hold-out set performance. We also discuss generalization, parameter effects, and broader implications in the context of knowledge-graph-based retrieval-augmented systems.

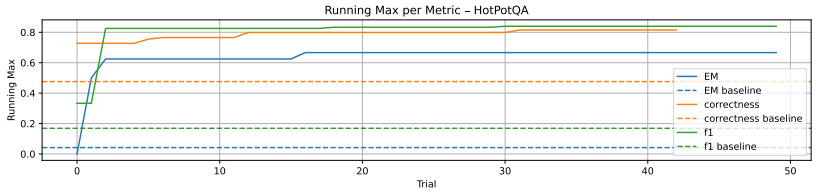
6.1 Training Set Performance



(a) Musique



(b) TwoWikiMultiHop



(c) HotPotQA

Figure 1: Running maximum performance curves for Musique, TwoWikiMultiHop, and HotPotQA.

Table 2 reports performance across all benchmark–metric combinations. The baseline corresponds to the default configuration used prior to optimization. Gains are expressed as the ratio between optimized and baseline scores, where defined.

Benchmark	Metric	Baseline	Optimized	Relative Gain (%)
Musique	Correctness	0.414	0.674	62.8
Musique	EM	0.000	0.500	–
Musique	F1	0.145	0.654	351.0
TwoWikiMultiHop	Correctness	0.348	0.582	67.2
TwoWikiMultiHop	EM	0.000	0.458	–
TwoWikiMultiHop	F1	0.148	0.625	321.6
HotPotQA	Correctness	0.476	0.815	71.2
HotPotQA	EM	0.042	0.667	1496.0
HotPotQA	F1	0.169	0.840	396.7

Table 2: Training set performance of baseline and optimized configurations. Relative gain is computed as the percentage increase from baseline. Undefined where the baseline is zero.

Optimization led to consistent improvements across all datasets and metrics. While the baseline settings were reasonable and manually selected, they were not tuned for the specific evaluation conditions. Relative improvements were often substantial, particularly for exact match, where several baselines were close to or exactly zero. This is largely due to a mismatch in answer style: the system's default configuration was tuned for more conversational output, whereas the benchmarks favored shorter, drier answers. Given EM's strictness as a metric, even factually correct responses were frequently penalized.

Despite the apparent improvements, these results should be interpreted with care. We return to this point in the discussion below.

6.2 Hold-Out Set Performance

Benchmark	Metric	Train Set	Hold-Out Set
HotPotQA	EM	0.667	0.583
HotPotQA	Correctness	0.815	0.715
HotPotQA	F1	0.840	0.819
Musique	EM	0.500	0.375
Musique	Correctness	0.674	0.596
Musique	F1	0.654	0.581
TwoWikiMultiHop	EM	0.458	0.417
TwoWikiMultiHop	Correctness	0.582	0.482
TwoWikiMultiHop	F1	0.625	0.704

Table 3: Performance of best configuration from each experiment on the train and held-out test sets.

To assess generalization, we evaluated the best configuration from each ex-

periment on a held-out test set. Table 3 shows the test results alongside the corresponding training scores. Gains over the baseline remained visible but were somewhat less pronounced than in training. Most metrics degraded moderately, and in one case (F1 on TwoWikiMultiHop), test performance slightly exceeded the training score. These results indicate that task-specific optimization generalizes reasonably well, even when applied to unseen examples from the same benchmark.

Some variability is likely attributable to the small size of the hold-out sets and the uneven quality of benchmark QA instances, a limitation noted throughout literature. We used a simple training setup without early stopping or regularization, which may also explain part of the observed degradation. Nevertheless, the fact that improvements persisted in most cases indicates that even basic optimization processes can yield generalizable gains. While not the primary focus of this study, these outcomes suggest that future work could explore more robust tuning regimes, especially on larger or domain-specific datasets.

A broader interpretation of these results, including implications for generalization and tuning strategies, is provided in the following discussion.

6.3 Discussion

As noted in Section 4, the optimization process used the Tree-structured Parzen Estimator (TPE), selected for its ability to navigate discrete and mixed parameter spaces. TPE was effective in identifying improved configurations, though trial-level performance was sometimes volatile. More stable or expressive optimization strategies may yield more consistent outcomes, and exploring such alternatives remains a direction for future work.

The experiments also underscored limitations in standard evaluation metrics. Exact match and F1 frequently penalized outputs that were semantically correct but phrased differently from the reference. In contrast, LLM-based correctness scores were more tolerant of lexical variation but introduced inconsistencies of their own. Several near-verbatim answers received less than full credit, suggesting that the LLM grader introduced noise, particularly around format sensitivity and implicit assumptions.

High-performing configurations often shared parameter settings, particularly for chunk size and retrieval method. However, most effects were nonlinear and task-specific, and no single configuration performed best across all benchmarks. This highlights the importance of empirical tuning in retrieval-augmented pipelines and suggests that generalization across tasks requires adaptation, not just reuse.

While full generalization remains outside the scope of this study, the results support the claim that systematic tuning is both achievable and useful in practice. The observed gains, while modest in some cases, show that configuration-level changes alone can influence downstream performance. Retrieval-augmented systems benefit from targeted, task-aware tuning, and performance–overfitting trade-offs can be managed without significant architectural change or added complexity.

7 Conclusion

We demonstrated that systematic hyperparameter tuning in graph-based retrieval-augmented generation systems can lead to consistent performance improvements. Cognee's modular architecture allowed us to isolate and vary configuration parameters across graph construction, retrieval, and prompting. Applied to three multi-hop QA benchmarks, this setup enabled us to examine how tuning affected standard evaluation metrics. While improvements were observed across tasks, their magnitude varied, and gains were often sensitive to both the metric and dataset.

Looking ahead, there are several natural directions for further work. Technically, the optimization process could be extended using alternative search algorithms, broader parameter spaces, or multi-objective criteria. Our evaluation focused on well-known QA datasets, but custom benchmarks and domain-specific tasks would help probe generalization. A leaderboard or shared benchmark infrastructure for graph-augmented RAG systems could also support progress in this area.

While QA-based metrics offer a practical means of evaluating pipeline performance, they do not fully capture the complexity of graph-based systems. The variability in outcomes across configurations suggests that gains are unlikely to come from generic tuning alone. Instead, our results point to the potential of task-specific optimization strategies, particularly in settings where domain structure plays a central role. We expect that future work at the intersection of academic and applied contexts will uncover further opportunities for targeted tuning.

More broadly, we think it is useful to view this process through the lens of cognification[1], a concept that describes how intelligence becomes embedded in physical systems. We see the development of frameworks like Cognee as part of a broader shift toward systems that reflect this paradigm, and their optimization plays an equally important role. The cognification of these systems will not happen through design alone, but through how they are tuned, measured, and adapted over time.

[1] See Appendix A.

A Cognee

Cognee is an open-source Python framework for end-to-end knowledge graph (KG) construction, retrieval, and completion. Its architecture is organized around a modular Extract–Cognify–Load (ECL) pipeline. The term cognify, introduced by Kevin Kelly, describes the process of adding intelligence to already digitized systems. In Cognee, it refers to the transformation of unstructured input into structured, semantically grounded graph representations.

The Extract stage ingests heterogeneous inputs such as text, images, and audio. The Cognify stage applies schema-based transformations using Pydantic models to identify entities, relations, and attributes. The Load stage writes the resulting data to graph, relational, or vector stores. Each stage is independently configurable and replaceable, which allows adaptation to diverse data types and scaling requirements.

Cognee also supports retrieval and reasoning over constructed graphs, with integration of large language models (LLMs). Users can submit structured graph queries, prompt-based interactions, or hybrid retrievals through a single interface. The system is distributed as a Python package with containerized deployment tools and a browser-accessible user interface.

A.1 Default Pipeline

The default Cognee pipeline processes unstructured inputs such as text, PDFs, images, audio transcripts, and source code into structured graph representations and vector embeddings. It operates through a sequence of modular components that can be configured independently.

Inputs are ingested from local directories or remote sources and stored in a file system or object store. Metadata including file name, media type, content hash, and source location is recorded in a relational database. Files are then classified by MIME type, optionally enriched with additional metadata, and deduplicated using content hashes. They can be organized into datasets to support reuse and incremental updates.

Documents are segmented into token-limited chunks using a configurable strategy. Each chunk is processed by a language model that fills structured schema objects representing entities, relations, and summaries. These outputs are converted into graph fragments and linked to their source. Optionally, additional nodes summarizing subgraphs or lengthy input chunks can be generated and integrated into the graph.

Processing is coordinated by an orchestration layer that manages input validation, scheduling, and endpoint checks. Configuration parameters are supplied through files or API calls. Final outputs are indexed in three storage systems: a graph database for entity and relation queries, a relational store for metadata, and a vector index for similarity-based retrieval.

Stage	Description
Ingestion	Load and normalize raw inputs into file or object storage
Tagging	Classify by media type, merge metadata, deduplicate, and organize into datasets
Chunking	Segment documents and prepare structured inputs for extraction
Graph Construction	Extract entities and relations using language models and assemble graph fragments
Indexing	Write outputs to graph, relational, and vector storage systems

Table 4: Summary of processing stages in the default Cognee pipeline.

A.2 Retrieval Strategies

Cognee provides several retrievers that differ in how they select and prepare context for question answering and generation. Each retriever operates over one or more indexed data sources, including vector stores and knowledge graphs. Some return retrieved content directly, while others invoke a language model to generate an answer based on the retrieved context. Multiple retrieval steps may be involved, depending on the retriever.

Retrievers can be selected through a single configuration parameter and do not require changes to the pipeline. The interface supports adjustments such as the number of retrieved items, scoring method, and prompt format where applicable. Further implementation details are available in the project repository.[2] The retrievers used in the hyperparameter optimization experiments are described in Section 4.

A.3 Evaluation Framework

Cognee includes an evaluation framework for benchmarking retrieval and generation components using multi-hop question answering datasets. The framework is organized as a four-stage pipeline: corpus construction, question answering, answer evaluation, and metric aggregation. All configuration is handled through a single declarative file.

In the corpus construction stage, a dataset adapter loads question–answer pairs and their associated source texts (e.g., HotPotQA, TwoWikiMultiHop). Before each run, all memory layers are cleared and documents are reprocessed using the default pipeline. Questions and reference answers are stored in a relational table to support later evaluation.

In the answering stage, the framework instantiates a specified retriever, such as a vector-based method or graph-based strategy. Each question is paired with retrieved context, optionally passed through a language model, and the

[2] `https://github.com/cognee-ai/cognee`

Retriever	Description
Summary-Based	Retrieves chunk-level summaries using semantic similarity
Chunk-Level	Retrieves original text chunks based on embedding similarity
Graph Neighborhood	Retrieves nodes adjacent to a matched graph entity
RAG	Passes retrieved text chunks to a language model for answer generation
Graph Completion	Retrieves graph triples and uses a language model to generate a response
Graph-Summary Completion	Summarizes a subgraph using a language model before generating a response

Table 5: Retriever types supported in Cognee.

generated answer is recorded alongside its input and gold reference.

Evaluation supports two modes. One uses structured LLM-based grading (e.g., GEval) to score outputs on metrics such as correctness, exact match, F1, and contextual coverage. The other performs a direct LLM-based comparison and returns a correctness score, optionally with justification.

Aggregated results are computed using bootstrap estimates of mean performance and confidence intervals. Output includes both tabular summaries and an interactive dashboard with plots and error breakdowns.

The framework is modular and allows new tasks, retrievers, and evaluation strategies to be added with minimal changes to configuration.

References

[1] Microsoft Research AI4Science and Microsoft Azure Quantum. "The impact of large language models on scientific discovery: a preliminary study using gpt-4". In: *arXiv preprint arXiv:2311.07361* (2023).

[2] Akari Asai et al. "Self-rag: Learning to retrieve, generate, and critique through self-reflection". In: *The Twelfth International Conference on Learning Representations*. 2023.

[3] Matthew Barker et al. "Faster, Cheaper, Better: Multi-Objective Hyperparameter Optimization for LLM and RAG Systems". In: *arXiv preprint arXiv:2502.18635* (2025).

[4] Ilias Chalkidis, Ion Androutsopoulos, and Nikolaos Aletras. "Neural legal judgment prediction in English". In: *arXiv preprint arXiv:1906.02059* (2019).

[5] Yiqun Chen et al. “Improving Retrieval-Augmented Generation through Multi-Agent Reinforcement Learning”. In: *arXiv preprint arXiv:2501.15228* (2025).

[6] Nurendra Choudhary and Chandan K Reddy. “Complex logical reasoning over knowledge graphs using large language models”. In: *arXiv preprint arXiv:2305.01157* (2023).

[7] Darren Edge et al. “From local to global: A graph rag approach to query-focused summarization”. In: *arXiv preprint arXiv:2404.16130* (2024).

[8] Wenqi Fan et al. “A survey on rag meeting llms: Towards retrieval-augmented large language models”. In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024, pp. 6491–6501.

[9] Jia Fu et al. “AutoRAG-HP: Automatic Online Hyper-Parameter Tuning for Retrieval-Augmented Generation”. In: *arXiv preprint arXiv:2406.19251* (2024).

[10] Yunfan Gao et al. “Retrieval-augmented generation for large language models: A survey”. In: *arXiv preprint arXiv:2312.10997* 2 (2023).

[11] Shailja Gupta, Rajesh Ranjan, and Surya Narayan Singh. “A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions”. In: *arXiv preprint arXiv:2410.12837* (2024).

[12] Bernal Jiménez Gutiérrez et al. “Hipporag: Neurobiologically inspired long-term memory for large language models”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024.

[13] Kelvin Guu et al. “Retrieval augmented language model pre-training”. In: *International conference on machine learning*. PMLR. 2020, pp. 3929–3938.

[14] Haoyu Han et al. “Retrieval-augmented generation with graphs (graphrag)”. In: *arXiv preprint arXiv:2501.00309* (2024).

[15] Xiaoxin He et al. “G-retriever: Retrieval-augmented generation for textual graph understanding and question answering”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 132876–132907.

[16] Xanh Ho et al. “Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps”. In: *arXiv preprint arXiv:2011.01060* (2020).

[17] Lei Huang et al. “A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions”. In: *ACM Transactions on Information Systems* 43.2 (2025), pp. 1–55.

[18] Kush Juvekar and Anupam Purwar. “Introducing a new hyper-parameter for RAG: Context Window Utilization”. In: *arXiv preprint arXiv:2407.19794* (2024).

[19] Ernests Lavrinovics et al. "Knowledge Graphs, Large Language Models, and Hallucinations: An NLP Perspective". In: *Journal of Web Semantics* 85 (2025), p. 100844.

[20] Patrick Lewis et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in neural information processing systems* 33 (2020), pp. 9459–9474.

[21] Junyi Li et al. "The dawn after the dark: An empirical study on factuality hallucination in large language models". In: *arXiv preprint arXiv:2401.03205* (2024).

[22] Linhao Luo et al. "Reasoning on graphs: Faithful and interpretable large language model reasoning". In: *arXiv preprint arXiv:2310.01061* (2023).

[23] Alex Mallen et al. "When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 9802–9822. DOI: 10.18653/v1/2023.acl-long.546. URL: https://aclanthology.org/2023.acl-long.546/.

[24] Shirui Pan et al. "Unifying large language models and knowledge graphs: A roadmap". In: *IEEE Transactions on Knowledge and Data Engineering* 36.7 (2024), pp. 3580–3599.

[25] Boci Peng et al. "Graph retrieval-augmented generation: A survey". In: *arXiv preprint arXiv:2408.08921* (2024).

[26] Fabio Petroni et al. "Language models as knowledge bases?" In: *arXiv preprint arXiv:1909.01066* (2019).

[27] Adam Roberts, Colin Raffel, and Noam Shazeer. "How much knowledge can you pack into the parameters of a language model?" In: *arXiv preprint arXiv:2002.08910* (2020).

[28] Karan Singhal et al. "Toward expert-level medical question answering with large language models". In: *Nature Medicine* (2025), pp. 1–8.

[29] Alon Talmor and Jonathan Berant. "The web as a knowledge-base for answering complex questions". In: *arXiv preprint arXiv:1803.06643* (2018).

[30] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. "From Louvain to Leiden: guaranteeing well-connected communities". In: *Scientific reports* 9.1 (2019), pp. 1–12.

[31] Harsh Trivedi et al. "MuSiQue: Multihop Questions via Single-hop Question Composition". In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 539–554.

[32] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[33] Chi Wang, Xueqing Liu, and Ahmed Hassan Awadallah. "Cost-effective hyperparameter optimization for large language model generation inference". In: *International Conference on Automated Machine Learning*. PMLR. 2023, pp. 21–1.

[34] Yu Wang et al. "Knowledge graph prompting for multi-document question answering". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 17. 2024, pp. 19206–19214.

[35] Shijie Wu et al. "Bloomberggpt: A large language model for finance". In: *arXiv preprint arXiv:2303.17564* (2023).

[36] Shi-Qi Yan et al. "Corrective retrieval augmented generation". In: (2024).

[37] Zhilin Yang et al. "HotpotQA: A dataset for diverse, explainable multi-hop question answering". In: *arXiv preprint arXiv:1809.09600* (2018).

[38] Wen-tau Yih et al. "The value of semantic parse labeling for knowledge base question answering". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2016, pp. 201–206.

[39] Jing Zhang et al. "Neural, symbolic and neural-symbolic reasoning on knowledge graphs". In: *AI Open* 2 (2021), pp. 14–35.

[40] Jing Zhang et al. "Subgraph retrieval enhanced model for multi-hop knowledge base question answering". In: *arXiv preprint arXiv:2202.13296* (2022).

[41] Yuyu Zhang et al. "Variational reasoning for question answering with knowledge graph". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[42] Penghao Zhao et al. "Retrieval-augmented generation for ai-generated content: A survey". In: *arXiv preprint arXiv:2402.19473* (2024).