SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Leveraging Bayesian Optimization for Accelerating RAG Pipeline Optimization

Xueru Zheng

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Leveraging Bayesian Optimization for Accelerating RAG Pipeline Optimization

Nutzung der Bayesschen Optimierung zur Beschleunigung der RAG-Pipeline-Optimierung

Author:	Xueru Zheng
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Anum Afzal
Submission Date:	15.10.2025

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.10.2025 Xueru Zheng

Location, Submission Date Author

AI Assistant Usage Disclosure

Introduction

Performing work or conducting research at the Chair of Software Engineering for Business Information Systems (sebis) at TUM often entails dynamic and multi-faceted tasks. At sebis, we promote the responsible use of *AI Assistants* in the effective and efficient completion of such work. However, in the spirit of ethical and transparent research, we require all student researchers working with sebis to disclose their usage of such assistants.

For examples of correct and incorrect AI Assistant usage, please refer to the original, unabridged version of this form, located at this link.

Use of *AI Assistants* for Research Purposes

I have used AI Assistant(s) for the purposes of my research as part of this thesis.

[x] Yes ◯ [ ] No ◯

Explanation:
ChatGPT were used in this research for the following purposes:

1. Obtaining LaTeX syntax guidance and retrieving relevant literature references
2. Enhancing text clarity and coherence, correcting grammatical issues, and refining academic language
3. Translating the abstract from English to German

I confirm in signing below, that I have reported all usage of AI Assistants for my research, and that the report is truthful and complete.

Munich, 15.10.2025	Xueru Zheng
Location, Date	Author

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Anum Afzal, for her exceptional guidance, continuous support, and invaluable feedback during the thesis. Her dedication, responsibility, and insightful suggestions greatly contributed to the completion of this work.

I am also sincerely thankful to Prof. Dr. Florian Matthes for providing me the opportunity to conduct my thesis at the Chair of Software Engineering for Business Information Systems (SEBIS) and for his overall support.

Furthermore, I would like to extend my appreciation to Tobias Müller and Atreya Biswas from SAP for their assistance and collaboration. Their help in understanding and deploying the LLM and reranker models was instrumental in the practical implementation of my research.

Finally, I would like to extend my heartfelt thanks to my family, friends, and my boyfriend for their constant support, encouragement, and understanding. Their presence and belief in me have been a constant source of motivation throughout this journey.

Abstract

Retrieval-Augmented Generation (RAG) systems have become essential for providing domain specific and up-to-date information while reducing hallucination in large language models. RAG works by retrieving relevant documents from a knowledge base and using them as context for generation. For each dataset, a different RAG configuration might be optimal because datasets vary in document structure, query types, and information requirements. Finding optimal configurations through traditional grid search methods remains computationally expensive as the number of components and hyperparameters grows. Grid search is also unintuitive as it provides no insight into the parameter space and cannot learn from previous evaluations.

This thesis presents a comprehensive empirical study investigating Bayesian Optimization (BO) as an efficient alternative to grid search. BO builds a surrogate model of the objective function and uses acquisition functions to balance exploration and exploitation in the parameter space. For RAG optimization, BO can be helpful by modeling complex interactions between pipeline components with fewer evaluations. AutoRAG is a framework that automates the configuration selection for RAG systems using grid search. We extend this framework by integrating BO methods to improve optimization efficiency. The main idea behind our approach is treating the entire RAG pipeline as a black box function mapping configurations to performance metrics. This allows BO algorithms to optimize without requiring detailed knowledge of component interactions. Our study compares seven BO strategies combining four surrogate models with two multi-fidelity methods across FiQA, SciFact, and HotpotQA datasets.

Our experiments demonstrate that BO achieves comparable model performance to grid search while reducing optimization time by up to 85 percent. SMAC3 using Random Forest and Optuna TPE emerged as the most effective algorithms. The study reveals that local optimization provides rapid results matching baseline performance while global optimization can exceed it with additional computational investment. BO effectiveness varies significantly across data domains through adaptive exploration strategies. These findings establish empirical evidence for BO superiority in complex pipeline optimization and provide practical guidelines for implementing efficient RAG optimization in resource-constrained production environments, making sophisticated hyperparameter tuning accessible without sacrificing model quality.

Kurzfassung

Retrieval-Augmented Generation (RAG)-Systeme sind unverzichtbar geworden, um domänenspezifische und aktuelle Informationen bereitzustellen und gleichzeitig Halluzinationen in großen Sprachmodellen zu reduzieren. RAG funktioniert durch das Abrufen relevanter Dokumente aus einer Wissensdatenbank und deren Verwendung als Kontext für die Generierung. Für jeden Datensatz könnte eine unterschiedliche RAG-Konfiguration optimal sein, da Datensätze in Dokumentstruktur, Abfragetypen und Informationsanforderungen variieren. Das Finden optimaler Konfigurationen durch traditionelle Grid-Search-Methoden bleibt rechnerisch aufwendig, da die Anzahl der Komponenten und Hyperparameter wächst. Grid-Search ist zudem unintuitiv, da es keine Einblicke in den Parameterraum bietet und nicht aus vorherigen Evaluierungen lernen kann.

Diese Arbeit präsentiert eine umfassende empirische Studie, die Bayessche Optimierung (BO) als effiziente Alternative zur Grid-Search untersucht. BO erstellt ein Surrogat-Modell der Zielfunktion und nutzt Akquisitionsfunktionen, um Exploration und Exploitation im Parameterraum auszubalancieren. Für die RAG-Optimierung kann BO hilfreich sein, indem es komplexe Interaktionen zwischen Pipeline-Komponenten mit weniger Evaluierungen modelliert. AutoRAG ist ein Framework, das die Konfigurationsauswahl für RAG-Systeme mittels Grid-Search automatisiert. Wir erweitern dieses Framework durch die Integration von BO-Methoden zur Verbesserung der Optimierungseffizienz. Die Hauptidee unseres Ansatzes besteht darin, die gesamte RAG-Pipeline als Black-Box-Funktion zu behandeln, die Konfigurationen auf Leistungsmetriken abbildet. Dies ermöglicht es BO-Algorithmen zu optimieren, ohne detaillierte Kenntnisse über Komponenteninteraktionen zu benötigen. Unsere Studie vergleicht sieben BO-Strategien, die vier Surrogat-Modelle mit zwei Multi-Fidelity-Methoden über die Datensätze FiQA, SciFact und HotpotQA kombinieren.

Unsere Experimente zeigen, dass BO eine vergleichbare Modellleistung wie Grid-Search erreicht, während die Optimierungszeit um bis zu 85 Prozent reduziert wird. SMAC3 mit Random Forest und Optuna TPE erwiesen sich als die effektivsten Algorithmen. Die Studie zeigt, dass lokale Optimierung schnelle Ergebnisse liefert, die der Baseline-Leistung entsprechen, während globale Optimierung diese mit zusätzlichem Rechenaufwand übertreffen kann. Die BO-Effektivität variiert erheblich über Datendomänen hinweg durch adaptive Explorationsstrategien. Diese Erkenntnisse liefern empirische Belege für die Überlegenheit von BO bei der komplexen Pipeline-Optimierung und bieten praktische Richtlinien für die Implementierung effizienter RAG-Optimierung in ressourcenbeschränkten Produktionsumgebungen, wodurch anspruchsvolle Hyperparameter-Abstimmung zugänglich wird, ohne die Modellqualität zu opfern.

Contents

List of Figures

List of Tables

1. Introduction

While large language models demonstrate remarkable capabilities in understanding and generating human-like text, they face fundamental limitations that restrict their practical deployment in knowledge-intensive applications. These models cannot access information beyond their training data cutoff date, making them unable to provide current information about recent events or evolving topics. They generate responses based solely on parameters learned during training, which leads to hallucinations when answering questions about specialized domains or factual queries requiring precise information. Models also lack transparency in their knowledge sources, making it impossible to verify claims or trace information back to authoritative documents. These constraints become particularly problematic in enterprise settings where accuracy, verifiability, and currency of information are essential for decision-making and regulatory compliance.

Retrieval-Augmented Generation systems address these limitations by enabling large language models to access and utilize external information. RAG enables models to retrieve relevant documents from knowledge bases and incorporate them into generation processes. Lewis et al. [1] introduced RAG as a framework consisting of two fundamental components: a retriever that finds relevant documents and a generator that produces augmented responses. This architecture addresses critical limitations of standalone language models by providing access to external knowledge, ensuring up-to-date information retrieval, and significantly reducing hallucination rates. While the core RAG architecture comprises these two basic components, real-world implementations often incorporate additional modules such as rerankers, filters, and compressors to enhance performance. Organizations now deploy these advanced RAG systems for customer support, technical documentation, regulatory compliance, and decision support across industries.

Despite these advantages, the architecture of RAG systems introduces substantial optimization challenges. Each component in the pipeline contains numerous hyperparameters requiring careful tuning. The retriever requires choosing between dense retrieval with embedding models, sparse retrieval with tokenizers and BM25 variants, or hybrid approaches combining both methods. Configuration also includes top-k counts and similarity thresholds that significantly impact retrieval quality and computational cost. Filters involve threshold and percentile settings. Generators demand configuration for temperature and maximum tokens. A complete RAG pipeline can easily generate thousands of possible configurations. These parameters interact in complex ways where changes in a single component can significantly affect overall system performance [2]. The interdependence between components makes optimization particularly challenging as improvements in one module may degrade performance in another.

Traditional optimization methods fail to scale for RAG pipelines due to combinatorial

explosion. Grid search exhaustively evaluates all parameter combinations, but with dozens of configurable parameters across multiple components, the search space easily exceeds millions of possible configurations. With each configuration requiring several minutes for evaluation on standard benchmarks, exhaustive search becomes computationally intractable. For example, evaluating just 10,000 configurations at 5 minutes each would require over 800 hours of computation. Random search reduces the computational burden but lacks the systematic exploration needed to identify optimal configurations in high-dimensional spaces. Manual tuning based on domain expertise provides no guarantees of optimality and suffers from poor reproducibility and incomplete coverage of the configuration space.

Production environments demand more efficient optimization strategies. Companies need to deploy RAG systems quickly to meet business requirements. Cloud computing costs make extensive optimization prohibitively expensive. Development teams require rapid iteration to improve model performance. Extended optimization cycles delay product launches and increase development costs. Many organizations accept suboptimal RAG performance rather than invest in comprehensive tuning. This gap between optimization needs and available methods limits RAG adoption and effectiveness.

Bayesian Optimization offers a principled solution to expensive optimization problems. BO builds a probabilistic surrogate model of the objective function and uses this model to guide the search process. Instead of evaluating configurations randomly or exhaustively, BO selects the most promising candidates based on prior observations. This approach balances exploration of unknown regions with exploitation of promising areas, making it particularly suitable for expensive function evaluations. The efficiency of BO in handling expensive evaluations has made it a standard approach for hyperparameter optimization in machine learning, where it consistently outperforms grid search and random search by requiring fewer evaluations [3, 4]. BO has been successfully applied across diverse ML applications including neural network optimization [5], automated machine learning systems [6], and combined algorithm selection and hyperparameter optimization [7]. These successes demonstrate BO's effectiveness in navigating complex, high-dimensional parameter spaces where traditional exhaustive methods become computationally prohibitive.

This thesis investigates whether Bayesian Optimization can effectively optimize RAG pipelines while reducing computational costs. We address four critical research questions. First, which BO algorithms perform best for RAG optimization among available options. Second, whether optimization should target the entire pipeline globally or individual components locally. Third, how many configurations BO needs to sample for effective optimization. Fourth, how data domain characteristics influence BO effectiveness and stability.

We conduct comprehensive experiments comparing seven distinct BO strategies, combining four surrogate models (Random Forest, Gaussian Process, Tree-structured Parzen Estimator, and Heteroscedastic Gaussian Process) with two multi-fidelity methods (Successive Halving and Hyperband). The evaluation uses FiQA, SciFact, and HotpotQA datasets representing different domains. Our framework measures both optimization efficiency and final model performance against grid search baselines. This systematic evaluation aims to provide practitioners with evidence based guidelines for selecting appropriate optimization strategies

for their specific RAG deployment requirements.

This work makes the following contributions to RAG optimization research. First, we provide systematic empirical evidence that Bayesian Optimization is a viable alternative to grid search for RAG pipelines, achieving significant computational savings without sacrificing performance. Second, we establish clear guidance on the global versus local optimization trade-off, demonstrating when each approach is most beneficial. Third, we quantify the relationship between the number of configurations explored and the resulting optimization performance, establishing practical guidelines for defining trial budgets and early stopping criteria. Fourth, we analyze how dataset domain characteristics influence optimization stability and effectiveness, revealing domain-dependent performance patterns that guide algorithm selection. Together, these contributions bridge the gap between theoretical BO capabilities and practical RAG system optimization.

The remainder of this thesis is organized as follows. Chapter 2 provides background on RAG systems, hyperparameter optimization challenges, and Bayesian Optimization fundamentals. Chapter 3 reviews related work in RAG tuning approaches, AutoRAG frameworks, applications of Bayesian Optimization in machine learning and existing studies on hyperparameter optimization within RAG pipelines. Chapter 4 describes the datasets used, including data preprocessing, validation set construction, and the handling of annotation limitations. Chapter 5 presents the methodology, covering research questions, pipeline architecture, optimization framework, and evaluation metrics. Chapter 6 details the experimental setup, implementation details, and reproducibility measures. Chapter 7 presents comprehensive evaluation results for each research question, analyzing algorithm performance, optimization scope, sample efficiency, and cross-dataset robustness. Chapter 8 discusses the interpretation of results, trade-offs between optimization strategies, experimental limitations and future work. Chapter 9 concludes the thesis by summarizing the key findings, highlighting the main contributions, and outlining directions for future research.

2. Background

2.1. Retrieval-Augmented Generation (RAG)

2.1.1. Foundation Concepts and Architecture

Large Language Models (LLMs) are neural networks trained on vast text corpora to generate human-like text. These models learn patterns from training data and produce responses based on learned representations. However, LLMs face inherent limitations. They cannot access information beyond their training cutoff date. They struggle with domain-specific or proprietary knowledge that is absent from their training data, which often leads to hallucinated outputs [8]. Models trained on general corpora lack specialized knowledge required for technical domains. They cannot adapt to new information without expensive retraining processes. These limitations restrict their utility in dynamic enterprise environments.

Retrieval-Augmented Generation addresses LLM limitations by combining retrieval with generation. Lewis et al. [1] introduced RAG as a framework with two fundamental components: a retriever and a generator. The retriever finds relevant documents from a knowledge base using dense or sparse retrieval methods. The generator produces responses conditioned on both the query and retrieved documents. This architecture enables access to external knowledge without retraining the model. The system maintains a separate knowledge base that can be updated independently. New information becomes immediately available through the retrieval mechanism.

Retrieval in RAG systems employs two primary approaches: sparse and dense retrieval. Sparse retrieval methods like BM25 use term frequency and inverse document frequency to match keywords between queries and documents [9]. BM25 remains effective for exact term matching and requires no training. Dense retrieval uses embeddings to capture semantic meaning beyond literal keywords. Embeddings are dense vector representations that encode text into high-dimensional spaces. Similar texts produce similar vectors in the embedding space, enabling efficient similarity search across large document collections [10]. Modern embedding models like BERT [11] and Sentence Transformers [12] create high-quality representations for retrieval tasks. These models transform both queries and documents into comparable vector spaces. The cosine similarity between vectors determines semantic relevance. Dense retrieval surpasses traditional keyword matching by understanding conceptual relationships between texts. Hybrid approaches combining sparse and dense retrieval can achieve improved performance by leveraging both exact matching and semantic understanding [13, 10, 14].

Vector databases enable efficient similarity search through specialized indexing structures. ChromaDB [15] provides a developer friendly interface for vector storage and retrieval,

automatically handling index management and supporting both in memory and persistent storage modes. FAISS [16] offers more granular control with multiple indexing algorithms including flat indexes for exact search and approximate methods like IVF [17] and HNSW [18] for scalability. Pinecone [19] and Weaviate [20] provide managed vector database services with built in scaling and query optimization. These databases must balance index construction time, memory usage, query latency, and ease of integration. ChromaDB's simplicity makes it ideal for prototyping and moderate scale deployments, while FAISS excels in high performance scenarios requiring fine tuned index configurations. The choice of vector database and indexing strategy significantly impacts retrieval performance, especially for large document collections exceeding millions of vectors.

The RAG process follows a systematic workflow that begins with query processing. First, the system receives a user query and preprocesses it for optimal retrieval. The retrieval method determines the subsequent processing steps. For dense retrieval, the retriever encodes the query into an embedding vector using the same model that encoded the document collection. It searches a vector database to find semantically similar documents through approximate nearest neighbor search. For sparse retrieval like BM25, the system tokenizes the query and matches terms against an inverted index based on term frequency statistics. Hybrid systems may execute both retrieval methods in parallel and combine their results. The system retrieves k most relevant documents, where k is a critical hyperparameter requiring careful tuning. These documents provide essential context for the generation phase. The generator then combines the original query with retrieved documents to produce an informed response. The final output reflects both the model's parametric knowledge and the retrieved information.

2.1.2. Advanced Components and Implementation

Modern RAG implementations extend significantly beyond the basic two-component architecture. Advanced systems enhance each stage of the retrieval and generation process to improve overall performance.

Document chunking strategies critically affect retrieval effectiveness [1]. Fixed size chunking divides documents into uniform token segments but may split semantic units. Sliding window chunking creates overlapping segments to preserve context across boundaries. Semantic chunking uses natural language processing to identify coherent text units such as paragraphs or sections. Recent approaches employ recursive chunking that adapts segment sizes based on content structure [21]. The chunk size parameter directly impacts the tradeoff between context completeness and retrieval precision. Smaller chunks improve precision but may lack sufficient context, while larger chunks provide more context but reduce specificity.

Query expansion techniques address vocabulary mismatch between user queries and document content. These methods transform the original query before retrieval to improve coverage. Hypothetical Document Embeddings (HyDE) [22] generates a hypothetical answer to create a more document-like query for dense retrieval. Multi-query expansion creates multiple query variations to capture different aspects of the information need. Query decomposition [23] breaks complex questions into simpler sub-queries that are easier to

match. These techniques help bridge the semantic gap between how users express queries and how information is stored in documents.

After initial retrieval, rerankers improve result quality by reordering documents. Rerankers address a fundamental limitation of first-stage retrieval: the tradeoff between efficiency and accuracy. While initial retrievers must search a very large collection of documents quickly, rerankers can apply more sophisticated models to a smaller candidate set [24].

Reranking approaches fall into three main categories. Cross-encoder models, such as MonoT5 [25] and UPR [26], encode query–document pairs jointly and capture detailed interactions between terms. Late-interaction models, such as ColBERT [27], keep separate query and document representations but compute token-level interactions, balancing accuracy with efficiency. Lightweight rerankers, such as FlashRank [28] and FlagEmbedding [29], use optimized or embedding-based architectures to achieve fast inference with lower resource costs. The choice of reranker significantly impacts system performance and latency. In practice, reranker selection depends on accuracy requirements, latency constraints, and available computational resources.

After reranking improves the relevance ordering of retrieved documents, additional processing stages further refine the context provided to the generator. These post-retrieval components address specific challenges in managing retrieved information effectively.

Filters serve as quality control mechanisms in the RAG pipeline. They remove irrelevant or low quality documents that could introduce noise into the generation context. Filtering strategies include threshold-based cutoffs that exclude documents below a minimum relevance score. Percentile cutoffs retain only the top percentage of retrieved documents. Similarity-based filters compare semantic similarity scores against predefined thresholds. Filters can also consider document metadata such as source credibility, publication date, or content type. The filtering stage prevents marginally relevant content from degrading the context quality. This becomes particularly important when initial retrieval returns documents with varying relevance levels.

Compressors address the context length limitations of language models. They reduce token counts while preserving essential information, enabling more efficient processing [30]. Compression techniques range from extractive summarization that selects key sentences to abstractive methods that generate condensed representations [21, 31]. These components work together in a pipeline where each stage refines the information flow.

Effective prompt design significantly influences how generators utilize retrieved context. Common prompting strategies vary in complexity and structure. The f-string method is based on Python's f-string formatting to insert retrieved documents directly into predefined templates, providing consistent structure across queries. Window replacement techniques [32] substitute placeholder tokens with retrieved content, allowing flexible positioning of context within the prompt. Long context reorder [33] reorganizes retrieved documents based on relevance scores or other criteria, placing the most important information in positions where language models pay most attention. Recent work shows that document position significantly affects how well models utilize retrieved information, with many models exhibiting a "lost in the middle" phenomenon where they better utilize information at the beginning or end of the

context window. The choice of prompt template and document ordering strategy becomes a critical hyperparameter affecting both response quality and hallucination rates.

The integration of these components requires careful orchestration, beginning with query expansion decisions that affect all subsequent stages. Expanded queries flow to retrievers configured for dense, sparse, or hybrid search. Rerankers then use more computationally expensive models than initial retrievers, scoring each document pair with the query to determine relevance more accurately. Filters apply various criteria including relevance thresholds, document freshness, and source credibility. Compression strategies range from extractive summarization to abstractive methods, operating on the filtered document set. Each component introduces its own hyperparameters and configuration choices, with query expansion parameters particularly influential as they determine the input characteristics for the entire pipeline. The interactions between components create complex optimization challenges that traditional methods struggle to address, especially when considering how query expansion strategies must align with retriever architectures and downstream processing requirements.

2.1.3. RAG Evaluation Metrics

The effectiveness of RAG systems fundamentally depends on comprehensive evaluation at multiple levels: component level metrics assess individual modules, while end-to-end metrics measure overall pipeline performance. These evaluation metrics become particularly critical when optimizing RAG pipelines, as they define the objective function that guides any optimization process. The selection of these metrics directly influences what configurations are considered optimal. A system optimized for retrieval precision may differ substantially from one optimized for generation fluency or end-to-end accuracy.

Retrieval metrics form the foundation for evaluating how well the system identifies relevant documents from the knowledge base. Precision@k measures the fraction of retrieved documents that are relevant among the top k results. Recall@k calculates the proportion of all relevant documents that appear in the top k retrieved items [34]. These metrics often conflict, as improving recall by retrieving more documents typically reduces precision. The F1 score provides a harmonic mean of precision and recall, offering a single metric that balances both concerns. These retrieval metrics can be computed efficiently without requiring expensive language model inference, making them suitable for rapid configuration evaluation during optimization.

Generation metrics evaluate the quality of the final output produced by the language model. Exact match scores determine whether the generated response perfectly matches reference answers, providing a strict evaluation criterion. Token level F1 scores offer a softer metric by computing precision and recall at the word level between generated and reference texts. Semantic similarity metrics assess meaning preservation beyond surface level matching.

Recently, LLM based evaluation has emerged as a scalable alternative to human judgment [35, 36]. Language models can assess aspects like faithfulness to retrieved context, answer relevance, and response coherence [37]. These LLM evaluators provide consistent scoring across large test sets, though they may inherit biases from their training data. The computational

cost of generation metrics, especially those requiring LLM inference, makes them expensive objective functions for optimization.

The properties of evaluation metrics significantly impact optimization strategy selection. Metrics must provide consistent and meaningful signals about configuration quality for Bayesian Optimization to function effectively. Noisy or unreliable metrics can mislead the surrogate model, causing it to build incorrect beliefs about the parameter landscape. The computational cost of metric evaluation also affects strategy choice. Expensive metrics that require full pipeline execution favor sample efficient methods like Bayesian Optimization, while cheap component level metrics might permit more exhaustive search approaches. In multi objective scenarios, the choice of which metrics to optimize simultaneously determines the trade offs embodied in the final system. These diverse metrics reflect different aspects of system performance, and their selection directly determines what type of RAG system emerges from the optimization process.

2.2. Hyperparameter Optimization

Hyperparameter optimization determines the configuration settings that control machine learning model behavior and training processes [38]. Unlike model parameters learned during training, hyperparameters must be set before training begins. These settings fundamentally influence model performance, training efficiency, and generalization capabilities. In complex systems like RAG pipelines, hyperparameter optimization becomes even more critical as multiple components interact through shared configurations.

Traditional optimization approaches include grid search, random search, and manual tuning. Grid search exhaustively evaluates all combinations of predefined hyperparameter values. This method guarantees finding the best configuration within the search space but suffers from exponential scaling. Random search samples configurations from specified distributions rather than testing all combinations. Empirical studies demonstrate that random search often matches or exceeds grid search performance while evaluating fewer configurations [38]. This efficiency gain occurs because many hyperparameters have minimal impact on performance, making exhaustive search wasteful.

The computational demands of hyperparameter optimization escalate dramatically in modern deep learning contexts. Each configuration evaluation requires complete model training and validation. For large language models, a single training run can take hours or days on expensive GPU hardware. RAG systems compound this challenge by requiring evaluation across multiple components. A configuration must be assessed on retrieval metrics, generation quality, and overall pipeline performance. The evaluation process involves processing hundreds or thousands of test queries through the entire pipeline. This makes traditional optimization methods prohibitively expensive for production systems.

Recent advances in optimization leverage adaptive and model-based approaches. Population-based methods like evolutionary algorithms explore multiple configurations simultaneously [39]. These techniques maintain diversity while converging toward promising regions. Gradient-based hyperparameter optimization computes gradients with respect to validation

loss [40]. This enables efficient optimization but requires differentiable validation procedures. Multi-fidelity methods, including Successive Halving [41] and Hyperband [42], allocate resources adaptively by evaluating configurations on reduced datasets or with fewer training epochs. These approaches identify promising configurations quickly before committing full resources.

The challenge of hyperparameter optimization intensifies when dealing with multiple objectives. RAG systems must balance retrieval accuracy, generation quality, and computational efficiency. Configurations that maximize retrieval precision may increase latency beyond acceptable limits. Settings that improve generation fluency might reduce factual accuracy. Multi-objective optimization frameworks address these tradeoffs by finding Pareto-optimal configurations, where no objective can be improved without degrading at least one other objective [43]. A configuration is Pareto-optimal if no other configuration exists that performs better in all objectives simultaneously [44]. These Pareto-optimal points form the Pareto front, representing the set of all optimal tradeoffs between competing objectives. However, selecting among Pareto-optimal solutions still requires human judgment about relative importance of objectives, as all points on the Pareto front are mathematically equivalent in terms of optimality.

Hyperparameter optimization for RAG systems presents unique challenges beyond traditional ML settings. The sequential nature of pipeline components creates dependencies between optimization stages. Changes in early components propagate through the pipeline, affecting downstream performance in unpredictable ways. Furthermore, the discrete choices of component selection combine with continuous hyperparameters to create mixed optimization spaces. Selecting between BM25 and dense retrieval represents a discrete choice, while filter thresholds for relevance cutoffs require continuous optimization. This mixture complicates optimization algorithms designed for purely continuous or discrete spaces. Additionally, evaluation costs vary significantly across components, with some requiring expensive LLM inference while others involve only embedding computations. These characteristics make RAG optimization particularly suited for intelligent search methods that can navigate complex parameter spaces efficiently while managing computational budgets.

2.3. Bayesian Optimization Fundamentals

Bayesian Optimization (BO) provides a principled framework for optimizing expensive black box functions where evaluations are costly and gradients are unavailable [45]. Unlike traditional optimization methods that treat each evaluation independently, BO builds a probabilistic model of the objective function and uses this model to guide the search process. This model based approach fundamentally changes how the optimization proceeds. While grid search and random search evaluate configurations without learning from previous evaluations, BO actively learns the function landscape and focuses on promising regions. The framework consists of two essential components: a surrogate model that approximates the objective function with uncertainty estimates and an acquisition function that uses these predictions to select the next evaluation point. The surrogate model predicts both the expected

performance and uncertainty at unobserved points, enabling intelligent selection of the next configuration to evaluate. This approach is particularly effective when function evaluations involve time consuming processes such as training machine learning models or running complex simulations.

The sample efficiency of Bayesian Optimization makes it especially valuable for expensive optimization problems like RAG pipeline configuration. Empirical studies show BO requires orders of magnitude fewer evaluations than random search for achieving similar performance [3]. This efficiency stems from two key capabilities of the surrogate model. First, it predicts poor configurations without requiring actual evaluation, effectively pruning large portions of the search space. Second, the uncertainty quantification ensures sufficient exploration to avoid premature convergence to local optima. For RAG pipelines, where each configuration evaluation requires processing numerous test queries through multiple components, this efficiency translates to significant time and computational cost savings. A single RAG evaluation might take minutes to hours depending on the dataset size and model complexity, making the sample efficiency of BO crucial for practical optimization.

The surrogate model in Bayesian Optimization approximates the unknown objective function based on observed evaluations. Common surrogate models include Gaussian Processes (GP) [46], Random Forests [47], and Tree-structured Parzen Estimators (TPE) [48]. Each model offers distinct advantages for hyperparameter optimization, including applications in RAG pipelines. Gaussian Processes provide well-calibrated uncertainty estimates but scale poorly with observations, typically being limited to hundreds of evaluations. Random Forests handle discrete and continuous parameters naturally but provide only approximate uncertainty estimates. Tree-structured Parzen Estimators model parameter distributions directly and scale better to high dimensions, though they assume parameter independence. The surrogate maintains a posterior distribution over the objective function, providing both predictions and uncertainty estimates for any point in the parameter space. As new observations are collected, the model updates its beliefs about the function landscape. This probabilistic representation enables informed decisions about where to sample next. The choice of surrogate model influences optimization efficiency and the types of parameters that can be optimized effectively.

The acquisition function determines which configuration to evaluate next by balancing exploration and exploitation. It leverages the surrogate model's predictions and uncertainty estimates to identify promising regions of the search space. Expected Improvement (EI) quantifies the expected gain over the current best observation [49, 50], while Probability of Improvement (PI) measures only the likelihood of surpassing the current optimum [51]. Upper Confidence Bound (UCB) trades off predicted value and uncertainty through an exploration parameter [52]. These acquisition functions reformulate the expensive optimization problem into a series of manageable optimization steps over the surrogate. The selected point is then evaluated on the true objective, and the surrogate model is updated with this new information.

While standard Bayesian Optimization assumes uniform evaluation cost across all configurations, practical optimization scenarios often provide opportunities to reduce computational

expense through approximate evaluations. Multi-fidelity optimization extends Bayesian Optimization by leveraging cheaper approximations of the objective function. These methods evaluate configurations at different fidelity levels, such as using data subsets or fewer training epochs [53]. Successive Halving [41] allocates resources through a tournament style elimination process. It starts with many configurations at low fidelity and progressively eliminates poor performers while increasing resources for survivors. Hyperband [42] combines Successive Halving with different initial budget allocations to balance exploration and exploitation. These methods assume that relative performance at low fidelity correlates with high fidelity performance. For RAG optimization, this might mean evaluating on fewer test queries initially.

The integration of multi-fidelity methods with Bayesian Optimization creates powerful hybrid approaches. BOHB (Bayesian Optimization and Hyperband) [54] uses a TPE surrogate model to select configurations within Hyperband's framework. This combination leverages both the sample efficiency of BO and the adaptive resource allocation of Hyperband. The surrogate model learns from evaluations at all fidelity levels, improving its predictions across the entire space. Early stopping mechanisms prevent wasting resources on clearly inferior configurations. These hybrid methods are particularly effective for RAG optimization where evaluation costs vary significantly between quick retrieval tests and full pipeline evaluations with expensive LLM inference.

While multi-fidelity methods address computational efficiency, practical RAG systems must also handle multiple competing objectives simultaneously. Multi-objective Bayesian Optimization extends the standard framework to optimize several goals at the same time. In RAG pipelines, this often means balancing accuracy with latency or cost. The optimization problem can be written as

$$\min_{\lambda \in \Lambda} \mathbf{f}(\lambda) = \big(f_1(\lambda), f_2(\lambda), \ldots, f_k(\lambda)\big)$$

where λ is a configuration in the search space Λ, and each $f_i(\lambda)$ represents an objective such as error rate, latency, or resource usage. The outcome is not a single solution but a set of Pareto-optimal configurations. A configuration is Pareto-optimal if no objective can be improved without degrading another. This representation allows system designers to choose the most suitable trade-off for deployment needs. Approaches such as ParEGO [55] use scalarization to combine multiple objectives into a single function, while other methods rely on hypervolume improvement to guide the search.

These advanced Bayesian Optimization techniques collectively address the unique challenges of RAG pipeline optimization. The combination of surrogate modeling for sample efficiency, multi-fidelity evaluation for computational tractability, and multi-objective optimization for practical deployment requirements provides a comprehensive framework for navigating the complex configuration spaces of modern RAG systems.

3. Related Work

This chapter reviews prior work on Retrieval-Augmented Generation (RAG) optimization and the use of Bayesian Optimization (BO) in machine learning. We first outline current practices in RAG system design and tuning, highlighting the reliance on manual configuration and the limitations of existing optimization workflows. We then discuss recent AutoRAG frameworks, general methods for Bayesian hyperparameter optimization, and their applications in natural language processing. The chapter concludes by identifying key research gaps that motivate our proposed approach to systematic RAG pipeline optimization.

3.1. Current Practices in RAG Optimization

Early RAG implementations relied on manual configuration selection through trial and error. Practitioners typically selected components based on intuition or limited experimentation. This approach proved inefficient as the number of possible configurations grows exponentially with the number of components and parameters. Manual tuning also lacks reproducibility and fails to explore the configuration space systematically. These limitations highlight the need for automated optimization approaches.

Recent toolkits provide building blocks for RAG systems but leave the optimization challenge unresolved. In particular, Jin et al. [56] introduce FlashRAG, which emphasizes comparing different RAG pipeline architectures rather than automating component selection. Their toolkit implements 23 distinct algorithms across four paradigms (sequential, conditional, branching, and loop), enabling evaluation of architectural strategies. However, users must still manually select which components to use within their chosen algorithm and tune corresponding hyperparameters. This manual selection process remains a bottleneck for achieving optimal performance.

Moreover, Abdallah et al. [57] present Rankify, which illustrates the component selection challenge. Their framework supports 7 retrieval techniques and 24 re-ranking models combined via a unified API. Given the vast number of possible component combinations and hyperparameters, identifying the optimal configuration demands extensive manual experimentation. The absence of automated tuning mechanisms in Rankify reinforces the case for using structured optimization like Bayesian methods.

Additionally, Y. Chen et al. [58] propose UltraRAG, focusing on domain-specific adaptation via automated knowledge generation, but they do not address pipeline configuration optimization. Although UltraRAG adapts data to particular domains, it assumes fixed component choices and hyperparameter settings. This limitation indicates that even advanced, adaptive RAG systems still do not resolve the underlying challenge: automatically selecting and tuning

components tailored to datasets. Thus, the disconnect between component availability and optimization capabilities motivates applying Bayesian optimization to RAG pipelines.

3.2. AutoRAG Frameworks and Their Limitations

While the previously discussed toolkits require manual configuration, recent work has attempted to automate the RAG optimization process. D. Kim et al. [59] introduce AutoRAG, the first comprehensive attempt at automated RAG pipeline optimization. The framework provides an end-to-end solution covering the entire RAG workflow, from data preprocessing and chunking to automatic component selection and evaluation. AutoRAG employs a node-based architecture where each node represents a pipeline component such as retriever, reranker, or generator. Users can configure multiple options for each node, including different chunking strategies, retrieval methods, and evaluation metrics. The framework then systematically explores these configurations to identify the best performing combination for a given dataset.

Despite its comprehensive nature, AutoRAG's optimization strategy presents fundamental limitations that restrict its effectiveness. The system uses local optimization, evaluating each component independently before combining the best-performing modules. In local optimization, each component focuses solely on optimizing its own configurations, such as top-k values for retrievers or threshold values for filters. The system selects the best configuration for the current component and passes this fixed choice to the next component, which then optimizes its own parameters based on the output from the previous stage. This sequential approach assumes that optimal components at each stage will combine to form an optimal pipeline. However, this assumption may not capture the full picture as components can have complex interactions that influence overall performance. In contrast, global optimization considers all components and their corresponding configurations simultaneously, evaluating the entire pipeline performance rather than individual component performance. Global optimization captures interdependencies between components that local optimization misses. For example, a retriever that performs well with one reranker may perform poorly with another, and these subtle interdependencies cannot be detected through independent evaluation. While local optimization can find reasonably good configurations efficiently, it potentially misses superior configurations that arise from synergistic component combinations.

The computational efficiency of AutoRAG's optimization approach poses additional challenges. The system relies on grid search methods that exhaustively evaluate predefined configuration sets without intelligent sampling strategies. While the discrete number of component options may be manageable, grid search becomes inefficient as it scales with the number of components and their parameters. More critically, AutoRAG cannot effectively handle continuous hyperparameters. Parameters such as threshold values, percentiles, and temperature settings must be discretized into predefined values (e.g., [0.1, 0.3, 0.5, 0.7, 0.9] for temperature). This discretization limits the search to a fixed set of points, potentially missing optimal values that lie between the predefined options. For continuous parameters that significantly impact performance, this limitation prevents fine-grained optimization and

forces users to either accept coarse-grained search or manually test intermediate values. Furthermore, the framework does not leverage information from previous evaluations to guide the search process, missing opportunities for more efficient exploration of the configuration space. Such challenges motivate the application of advanced hyperparameter optimization methods.

3.3. Bayesian Optimization in Machine Learning

Bayesian Optimization (BO) has established itself as a leading approach for hyperparameter optimization in machine learning systems. Snoek et al. [5] demonstrated that BO can achieve state-of-the-art performance for neural network optimization while requiring significantly fewer evaluations than traditional methods. Their implementation, Spearmint, outperformed manual tuning and random search across several challenging benchmarks. This success motivated widespread adoption of BO across the machine learning community and inspired its integration into automated machine learning (AutoML) frameworks.

The integration of BO into AutoML systems marked a significant advance in automated model design. Thornton et al. [7] introduced Auto-WEKA, which formulated the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem and used BO to jointly optimize algorithms and hyperparameters. Feurer et al. [6] developed Auto-sklearn, extending this concept with meta-learning and ensemble construction. Mendoza et al. [60] proposed Auto-PyTorch, applying BO to deep neural network architecture search and demonstrating that BO can manage complex, hierarchical configuration spaces. Empirical studies further confirmed BO's superiority over alternative optimization methods. Bergstra and Bengio [38] showed that BO consistently outperforms grid search and random search across diverse datasets and model types, typically requiring an order of magnitude fewer evaluations. These results highlight BO's efficiency and scalability for optimizing large, expensive, and structured search spaces—properties that closely align with the requirements of RAG pipeline optimization.

Beyond these early AutoML systems, several modern frameworks have further advanced BO for practical large-scale applications. SMAC3 [47] extends BO using random forest surrogates and is widely used for algorithm configuration. Optuna [61] and Ray Tune [62] provide flexible, scalable implementations that support Tree-structured Parzen Estimators and parallelized optimization. More recently, HEBO [63] introduced a heteroscedastic Gaussian Process surrogate with input/output warping and multi-objective acquisition ensembles for more robust performance in high-dimensional search spaces. These toolkits demonstrate the maturity and adaptability of BO across diverse domains and reinforce its suitability for complex configuration optimization tasks such as RAG pipelines.

Given these successes, applying BO to the configuration optimization of RAG systems represents a natural and promising progression.

3.4. Prior Work on RAG Pipeline Optimization

Beyond manual configuration approaches, several research efforts have explored automated optimization techniques for RAG pipelines. LlamaIndex [64] provides ParamTuner, an experimental framework for hyperparameter optimization in RAG systems. The base ParamTuner iterates through all parameter combinations using exhaustive grid search, while the AsyncParamTuner enables parallel evaluation of multiple configurations. The framework also integrates with Ray Tune through RayTuneParamTuner, which provides access to more advanced optimization algorithms, though the default implementation uses grid search. ParamTuner allows optimization of parameters such as chunk size and top-k retrieval values, evaluating configurations based on metrics like semantic similarity. While ParamTuner provides basic optimization functionality, it primarily targets continuous parameters within fixed pipeline architectures and requires manual specification of the parameter grid to search.

To address the inefficiency of exhaustive search, more sophisticated algorithms have been applied to RAG optimization. Fu et al. [65] propose AutoRAG-HP, which formulates hyperparameter tuning as an online multi-armed bandit problem and introduces a two-level Hierarchical MAB method to explore search spaces. The framework optimizes top-k retrieved documents, prompt compression ratio, and embedding methods, achieving comparable performance using only 20% of the LLM API calls required by grid search through intelligent sampling strategies.

Building on the success of intelligent sampling, recent work has adopted Bayesian optimization as a principled approach for navigating complex RAG configuration spaces. Barker et al. [66] introduce multi-objective parameter optimization for RAG systems, simultaneously optimizing cost, latency, safety, and alignment. Their method employs Bayesian optimization with the qLogNEHVI acquisition function to handle noisy objective evaluations and discovers Pareto-optimal configurations. Experimental results on financial and medical QA benchmarks demonstrate that Bayesian optimization significantly outperforms random search in obtaining superior Pareto fronts.

Several frameworks have since leveraged Bayesian optimization for different aspects of RAG pipeline tuning. RAGBuilder [67] focuses on continuous hyperparameter tuning using Bayesian optimization, targeting parameters such as chunk size, temperature, and top-k values. The tool provides pre-defined RAG templates and synthetic test dataset generation capabilities.

Extending Bayesian optimization to handle the full complexity of modern RAG systems, Conway et al. [68] introduce Syftr, the most comprehensive multi-objective optimization framework for RAG pipelines to date. The system performs efficient search over a vast configuration space containing over 10^{23} unique RAG flows, including both agentic and non-agentic configurations. Syftr employs Bayesian optimization to discover Pareto-optimal flows that jointly optimize task accuracy and cost. Across multiple benchmarks, Syftr identifies flows that are approximately nine times cheaper while preserving most of the accuracy.

3.4.1. Research Gaps and Motivation

While the aforementioned approaches have demonstrated the potential of automated optimization for RAG pipelines, they collectively exhibit several critical limitations that constrain their practical applicability and optimization effectiveness.

First, most existing methods optimize only a limited subset of hyperparameters rather than the full configuration space. Current approaches typically handle either discrete component selection or continuous hyperparameter tuning, but rarely address both simultaneously within a unified framework. Second, there is no systematic comparison of different Bayesian optimization algorithms or surrogate models to identify which configurations are most effective for RAG optimization. Third, the influence of global versus local optimization strategies remains largely unexplored, leaving uncertainty about which approach better balances exploration and exploitation in complex configuration spaces. Finally, existing studies seldom examine how optimization performance varies across dataset domains or analyze the stability and transferability of optimized configurations.

These gaps motivate the need for a comprehensive investigation of Bayesian optimization techniques specifically tailored to RAG pipeline configuration challenges. To address these limitations, we extend AutoRAG's modular architecture with advanced Bayesian optimization capabilities. Our framework handles the full complexity of RAG configuration through hierarchical optimization that addresses three levels of decisions. First, it determines whether to include optional components such as rerankers, filters, and compressors. Second, it selects the specific method to use within each included component. Third, it optimizes the continuous hyperparameters associated with the selected methods. This unified approach enables joint optimization of both discrete choices and continuous parameters, overcoming the separation that limits existing methods.

To comprehensively evaluate the effectiveness of different Bayesian optimization strategies, we experiment with various surrogate models, including Random Forest (RF), Tree-Structured Parzen Estimator (TPE), Gaussian Process (GP), and Heteroscedastic Gaussian Process (HGP), together with multi-fidelity algorithms, namely Successive Halving and Hyperband. These configurations are evaluated across three datasets (SciFact, FiQA, and HotpotQA) to examine performance and generalizability across diverse domains.

4. Datasets

This chapter describes the datasets used to evaluate the proposed optimization framework. It outlines the dataset selection process, data preprocessing workflow, construction of validation sets, and challenges encountered during dataset analysis. The datasets represent diverse domains and reasoning requirements, enabling a comprehensive evaluation of retrieval-augmented generation (RAG) pipelines under varied conditions.

Each dataset was selected based on three main criteria: (1) availability of query–document pairs suitable for retrieval-based evaluation, (2) manageable corpus size for repeated optimization runs, and (3) sufficient query diversity to assess generalization across domains. These criteria ensure that the experiments capture realistic retrieval challenges while remaining computationally feasible for large-scale optimization.

The chapter also details the creation of consistent validation subsets for all datasets to ensure fair comparisons between optimization runs. This design allows the optimizer to explore configuration performance systematically without variance introduced by query sampling.

Finally, it discusses practical issues observed during data preparation, such as incomplete or inconsistent ground-truth annotations that affect retrieval metrics. These limitations motivated the introduction of large language model (LLM)–based semantic evaluation, providing a more accurate measure of retrieval relevance.

4.1. Data Preprocessing and Validation Set Construction

The experiments utilize three established benchmark datasets from different domains: SciFact for scientific fact verification, FiQA-2018 for financial question answering, and HotpotQA for multi-hop reasoning tasks. SciFact and FiQA-2018 were obtained through institutional access via SAP's research collaboration portal, while HotpotQA was sourced from the BEIR benchmark collection [69]. These datasets were selected to represent diverse retrieval challenges across scientific, financial, and general knowledge domains while maintaining computational feasibility for repeated optimization experiments.

4.1.1. Data Format Conversion

The datasets required conversion from their original JSONL format to Parquet files compatible with the optimization framework. Following AutoRAG's data schema, we structured the data into two Parquet files: `qa.parquet` containing query-answer pairs with columns for `qid` (unique query identifier), `query` (question text), `retrieval_gt` (ground truth document IDs),

and `generation_gt` (reference answer text); and `corpus.parquet` containing document collections with `doc_id` (unique document identifier), `contents` (document text), and `metadata` (additional document properties). Parquet format was chosen for its columnar storage efficiency, which significantly reduces file sizes and improves I/O performance when processing large document collections, particularly beneficial for the 57,638 documents in FiQA-2018.

The conversion process loads corpus documents and queries from their respective JSONL files, then maps retrieval ground truth document IDs using qrels (query relevance) files that specify which documents are relevant for each query. Document chunking was not applied as the datasets already provide documents in appropriate granularity for retrieval tasks, with each document representing a complete semantic unit—scientific abstracts in SciFact, financial articles in FiQA-2018, or Wikipedia passages in HotpotQA. Additionally, embedding generation for dense retrieval methods occurs dynamically during optimization runs rather than during preprocessing. When a configuration selects dense retrieval methods, the system generates embeddings on demand if not already cached, then stores them for subsequent runs. This lazy evaluation approach avoids generating embeddings for retrieval methods that do not require them, such as BM25. The reported optimization times in our results exclude initial embedding generation to ensure fair comparison across different retrieval methods.

4.1.2. Ground Truth Generation

Since the original datasets lack generation ground truths necessary for evaluating the generator component, we employed AutoRAG's generation ground truth creation framework using GPT-4. This automated process provides the retrieval ground truth documents to the language model, which generates reference answers that serve as targets for generation metrics evaluation. The generated answers establish a baseline for assessing the quality of the RAG pipeline's final outputs, enabling comprehensive evaluation beyond retrieval accuracy alone.

The ground truth generation process follows a structured approach to ensure consistency across datasets. For each query, the system retrieves the annotated ground truth documents from the corpus and constructs a prompt that includes both the original question and the relevant documents. GPT-4 then generates a comprehensive answer based solely on the provided documents, ensuring that the reference answers remain grounded in the actual retrieval results rather than relying on the model's parametric knowledge. This approach creates evaluation targets that accurately reflect what an ideal RAG system should produce when given perfect retrieval results.

The quality and consistency of these generated ground truths directly impact the reliability of generation metrics. While human-annotated answers would provide the gold standard for evaluation, the automated approach offers practical advantages for large-scale optimization experiments, including consistent answer formatting across hundreds of queries and reproducibility across different experimental runs. The use of GPT-4 ensures high-quality reference answers that capture the essential information from the retrieved documents while maintaining natural language fluency.

4.1.3. Validation Set Selection

Each dataset's validation set consists of 200 randomly sampled queries from the available query pool, providing sufficient statistical power while maintaining computational feasibility. For SciFact and FIQA, we directly used the provided corpus files since their sizes were appropriate for our experimental setup. In contrast, the original HotpotQA corpus exceeds 2GB, which was not suitable for our setup. Therefore, we constructed a reduced corpus by first randomly selecting 200 queries without replacement and then including all documents referenced in the retrieval ground truth of these queries, along with additional documents to preserve realistic retrieval challenges. This approach ensures that each validation set contains both relevant and nonrelevant documents, preventing artificially high retrieval scores that would occur if only ground truth documents were included.

The 200 query sample size balances evaluation thoroughness with computational efficiency. Preliminary experiments with 500 samples demonstrated that larger validation sets significantly increased processing time, particularly for configurations involving reranking components that must score all retrieved documents for each query. The computational burden scales linearly with sample size, with each configuration evaluation requiring processing all queries through the complete pipeline. For a 50 configuration optimization run with 200 samples, this results in approximately 10,000 total query evaluations, making larger validation sets impractical for extensive optimization experiments.

Validation sets remain fixed across all experiments to ensure fair comparison between optimization methods and configurations. The same 200 queries are used for all optimization runs on a given dataset, eliminating variance from query selection and enabling direct performance comparison. No separate training set was created as Bayesian Optimization learns directly from evaluation results rather than requiring pretraining on dataset specific patterns. The optimizer discovers effective configurations through iterative evaluation on the validation set, with the surrogate model learning the relationship between configurations and performance without access to held out data.

4.2. Dataset Selection and Characteristics

This section introduces the three benchmark datasets used in this study: SciFact, FiQA-2018, and HotpotQA. These datasets were selected to represent distinct domains and reasoning requirements, enabling comprehensive evaluation of retrieval-augmented generation (RAG) pipelines across scientific, financial, and general multi-hop contexts. The selection ensures realistic retrieval challenges while maintaining computational feasibility for large-scale optimization experiments. All length-related statistics in this chapter are reported in tokens, corresponding to the model's input units, to provide a consistent measure across retrieval and generation components.

SciFact

The SciFact dataset contains 5,183 scientific abstracts focused on fact verification within research literature. Each query represents a scientific claim, and the task involves retrieving abstracts that support or contradict the claim. The dataset was obtained through institutional access via SAP's research collaboration portal. SciFact was selected for its domain-specific language, moderate corpus size, and clearly defined ground truth labels, which make it suitable for repeated optimization runs. Its structured and concise abstracts provide an effective setting for evaluating retrieval accuracy and optimization efficiency in specialized scientific contexts.

The full SciFact corpus contains 5,183 documents and 1,409 annotated claims, while 200 queries were randomly sampled for validation experiments. The average document length is 215 tokens, and the average query length is 12 tokens. On average, each query is associated with 1.15 relevant documents, indicating a focused retrieval task where precision plays a central role.

Figure 4.1a illustrates the distribution of document lengths. Most abstracts range between 100 and 300 tokens, reflecting consistent structure and concise writing typical of scientific literature. The x-axis is truncated at 500 tokens to highlight the main distribution range; fewer than 1% of documents exceed this length. Figure 4.1b shows the number of ground truth documents per query. Nearly 90% of the queries are associated with a single relevant document, and only a few require multiple sources for verification. This confirms that SciFact primarily evaluates precise retrieval rather than large-scale recall, making it an appropriate dataset for analyzing fine-grained optimization behavior in RAG systems.

Overall, SciFact provides a compact and well-structured benchmark for evaluating retrieval and generation performance under domain-specific constraints. Its concise abstracts, consistent annotations, and low redundancy make it ideal for testing optimization frameworks that rely on precise retrieval and efficient configuration tuning.

FiQA-2018 Dataset

The FiQA-2018 dataset comprises 57,638 financial documents drawn from diverse sources such as news articles, company reports, and social media posts. It was originally developed for financial question answering and sentiment analysis tasks, focusing on domain-specific reasoning over economic and investment focused content. This dataset provides a valuable benchmark for evaluating RAG optimization in specialized and information rich domains.

The validation subset used in this study includes 200 randomly sampled queries paired with ground truth document identifiers. The corpus presents a moderately sized retrieval challenge suitable for large-scale optimization experiments while remaining computationally feasible. The average document length is 133 tokens, and the average query length is 12 tokens. Each query is associated with an average of 2.68 relevant documents, indicating that queries often require integrating information across multiple short documents.

Figure 4.1c shows the distribution of document lengths in FiQA. Most documents contain fewer than 200 tokens, reflecting the concise style typical of financial writing. A small number

of documents exceed 500 tokens, corresponding to long-form reports or analytical summaries. Figure 4.1d displays the number of ground truth documents per query. While the majority of queries have one or two relevant documents, a subset of queries links to over ten relevant entries, suggesting uneven annotation density across the dataset.

Additional analysis of annotation inconsistencies and their implications for retrieval evaluation are discussed in Section 4.3.

HotpotQA Dataset

The HotpotQA dataset contains 13,783 documents and was obtained from the BEIR benchmark collection [69]. It is designed for multi-hop question answering tasks, where each query requires reasoning across multiple supporting documents to produce an answer. This property makes HotpotQA an effective benchmark for evaluating retrieval-augmented generation systems in complex reasoning scenarios.

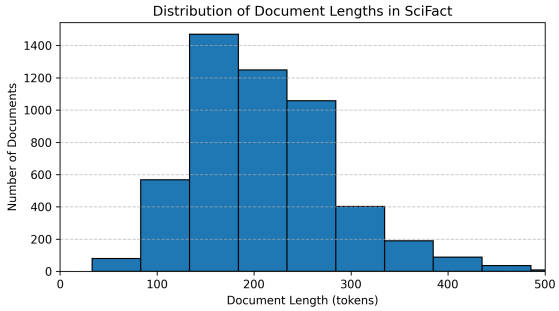
For this study, 200 queries were randomly selected for validation. To maintain computational feasibility while preserving realistic retrieval conditions, a reduced corpus was constructed by including all documents referenced in the retrieval ground truths of the sampled queries, along with additional unrelated documents. The average document length is 69 tokens, and the average query length is 16 tokens. Each query is linked to exactly two ground truth documents, reflecting the multi-step reasoning structure of the dataset.

Figure 4.1e shows the distribution of document lengths in HotpotQA. Most documents are short, typically under 100 tokens, representing concise information passages from Wikipedia articles. Figure 4.1f illustrates the number of ground truth documents per query, confirming that all queries are associated with two relevant documents. This consistent structure provides a stable setting for evaluating retrieval and reranking components that rely on multi-document reasoning.

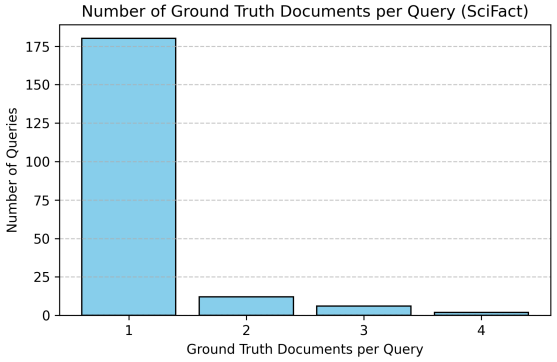
4.3. Annotation Limitations and Semantic Evaluation Adjustment

A critical observation during dataset analysis revealed limitations in the retrieval ground truth annotations, particularly in FiQA-2018. The ground truth document IDs do not always comprehensively capture all documents containing answer-relevant information. This incompleteness likely stems from the practical constraints of manual annotation, where annotators may not exhaustively review all 57,638 documents for each query, leading to false negatives where relevant documents remain unmarked. Table 4.1 illustrates four representative examples where retrieved documents contain accurate, answer-relevant information but are marked as incorrect by ID-based metrics.

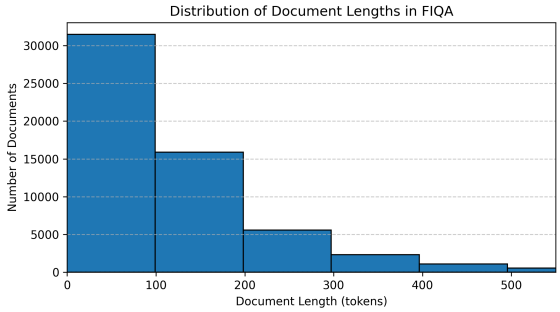
These examples demonstrate that retrieved documents not present in the ground truth IDs can provide factually correct and relevant answers. Among 200 validation samples in FiQA, approximately 20 cases (10%) demonstrated high LLM relevance scores (above 0.7) despite retrieval ID mismatches. This pattern suggests systematic under-annotation rather than isolated errors, as the overlooked documents often contain substantive relevant content



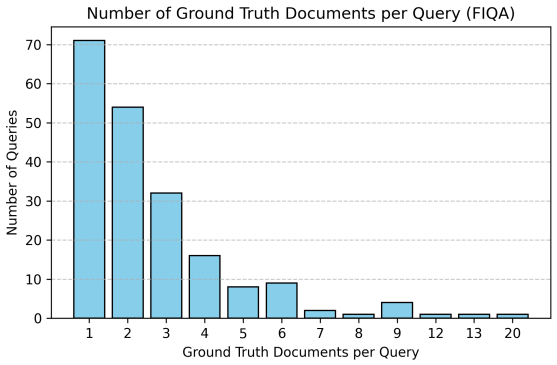
(a) SciFact: document lengths (tokens).



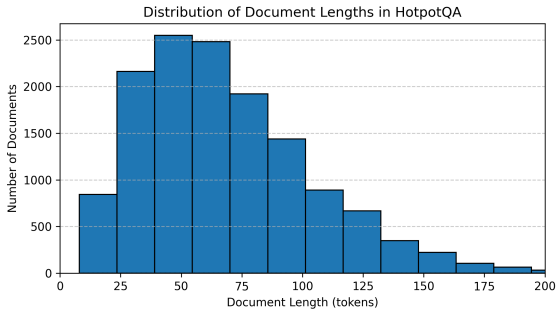
(b) SciFact: ground truth per query.



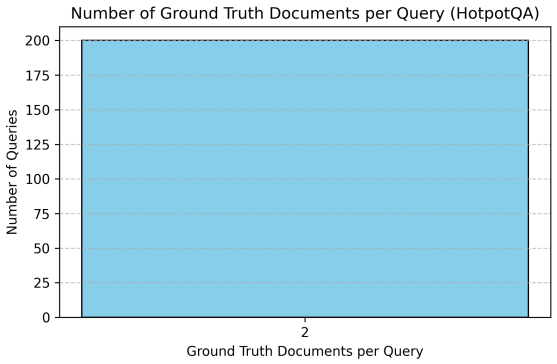
(c) FiQA: document lengths (tokens).



(d) FiQA: ground truth per query.



(e) HotpotQA: document lengths (tokens).



(f) HotpotQA: ground truth per query.

Figure 4.1.: Comparison of dataset characteristics across SciFact, FiQA-2018, and HotpotQA. Each row shows the distribution of document lengths (left) and the number of ground truth documents per query (right) for one dataset.

rather than peripheral or incidental references.

The implications of these annotation limitations extend beyond simple metric calculation. ID-based evaluation would penalize a retrieval system for finding these semantically relevant documents, potentially steering optimization toward configurations that miss valuable information sources. This discrepancy between ID-based retrieval metrics and semantic relevance evaluations influenced our decision to incorporate LLM-based evaluation for compression. By using semantic evaluation alongside traditional metrics, the optimization process can recognize and reward the retrieval of semantically relevant content regardless of ID matching, leading to more robust configurations that prioritize actual information utility over strict adherence to potentially incomplete ground truth annotations.

Table 4.1.: Examples of ground truth annotation limitations in FiQA-2018

Query	**Ground Truth IDs**	**Retrieved ID**	**Retrieved Content (excerpt)**	**LLM Score**
"What does APR mean I'm paying?"	263949, 279845, 551424	59540	"APR stands for 'annual percentage rate.' This means when you see a loan with a 6% rate, it is 6% per year..."	0.85
"What does inflation actually mean?"	302792, 593820	117578, 513249	"Inflation is an attempt to measure how much less money is worth. It is a weighted average of some bundle of goods and services price's increase..."	0.80
"Can I pay estimated taxes based on last year's taxes if I anticipate more income this year?"	191965	418871, 127974	"You can simply use the previous year's tax liability as your basis for payments... You can completely base your estimated taxes for this year on last year's tax return and avoid any underpayment penalty..."	0.95
"When after a companys IPO date can I purchase shares?"	591436	200894, 573600	"By definition, an IPO'd stock is publicly traded, and you can buy shares if you wish... The first moment of trading usually occurs even later than that. It may take a few hours to balance the current buy/sell orders and open the stock..."	0.9

5. Methodology

This chapter presents the experimental methodology for optimizing Retrieval-Augmented Generation pipelines using Bayesian Optimization techniques. The methodology encompasses five primary aspects that form the foundation of this research.

First, the research questions section establishes the specific inquiries that guide our investigation and experimental design. Second, the pipeline architecture establishes the modular RAG framework used for evaluation, defining how components interact and process information from query to response generation. Third, the detailed pipeline components section examines each module's available algorithms and configuration options, creating the vast search space that requires intelligent optimization. Fourth, the optimization framework introduces Bayesian Optimization as the core approach for efficiently exploring this high-dimensional configuration space, comparing global and local optimization strategies to understand their effectiveness in finding optimal pipeline configurations. Fifth, the evaluation methodology defines the metrics and assessment criteria used to measure both component-level and end-to-end performance, enabling multi-objective optimization that balances quality, latency, and computational efficiency.

Together, these methodological components provide a systematic approach to answering the research questions about which Bayesian Optimization algorithms perform best for RAG optimization, whether global or local optimization yields superior results, how the number of explored configurations affects final performance, and how dataset domains influence optimization stability.

5.1. Research Questions

This study addresses four fundamental research questions that guide the experimental design and evaluation approach for optimizing RAG pipelines through Bayesian Optimization. Each question targets a specific aspect of the optimization challenge and requires distinct experimental methodologies to provide comprehensive answers.

The research questions are formulated as follows:

- **RQ1:** Which Bayesian Optimization algorithms are most effective for optimizing Retrieval-Augmented Generation pipelines in terms of achieving high quality scores and computational efficiency?
- **RQ2:** Should Bayesian Optimization be applied globally across the entire RAG pipeline or locally at individual module levels to achieve optimal end-to-end generation quality?

- **RQ3:** What is the relationship between the number of configurations explored through Bayesian Optimization and the final performance scores achieved in RAG pipeline tuning?
- **RQ4:** How do dataset domain characteristics influence the effectiveness and stability of Bayesian Optimization when tuning RAG pipeline configurations across different application contexts?

To identify the most suitable optimization algorithms, we examine the comparative performance of multiple Bayesian Optimization variants including different surrogate models such as Gaussian Processes, Random Forests, and Tree Parzen Estimators. The experimental design evaluates each algorithm's ability to navigate the complex mixed discrete and continuous search space while balancing exploration of new configurations against exploitation of promising regions. Performance metrics include both the final achieved scores and the convergence speed to identify algorithms that not only reach optimal solutions but do so efficiently within limited computational budgets.

Beyond algorithm selection, the optimization scope presents another critical design decision. Global optimization treats the entire pipeline as a single black box function and considers interactions between components to optimize the pipeline holistically. This approach potentially discovers synergistic configurations that component level optimization might miss. Conversely, local optimization decomposes the problem into smaller subproblems and optimizes each module separately before combining the best configurations. This comparison reveals whether component interactions significantly influence overall performance and determines the most effective optimization granularity for practical deployments.

Equally important is understanding how sampling budget affects optimization outcomes. This investigation examines whether doubling the configuration budget from 50 to 100 samples yields substantial performance improvements or if the baseline budget already captures most achievable gains. By comparing the final scores achieved with different sampling budgets, we determine whether extended exploration justifies the additional computational cost. This analysis helps establish practical guidelines for budget allocation and identifies the point where additional sampling provides diminishing returns.

Finally, the influence of dataset characteristics on optimization behavior requires careful examination. The study employs datasets from distinct domains including scientific literature, financial documents, and multi hop reasoning tasks to analyze how different optimization landscapes affect Bayesian Optimization performance. We investigate whether difficult optimization landscapes with clear failure modes actually benefit BO by providing strong gradient signals for learning, while seemingly easier domains with acceptable performance plateaus might trap the optimizer in premature convergence. This analysis reveals how dataset characteristics such as query complexity, document length distribution, and retrieval difficulty patterns influence BO convergence behavior and stability across different domains.

These research questions collectively address the practical challenges of deploying Bayesian Optimization for RAG systems in production environments. The answers guide practitioners in selecting appropriate algorithms, determining optimization scope, allocating computational

budgets, and understanding how domain characteristics affect optimization behavior. The experimental methodology designed to answer these questions ensures reproducible results while providing actionable insights for real world RAG optimization tasks.

5.2. Pipeline Architecture

Overall RAG Pipeline Design

The proposed framework employs a modular RAG pipeline architecture consisting of six sequential processing stages that transform user queries into generated responses. The pipeline begins with optional query expansion followed by document retrieval from a knowledge base. Retrieved documents pass through reranking, filtering, and compression stages before reaching the final generator component. Each stage offers multiple algorithm choices and hyperparameter configurations, creating a vast combinatorial search space. The architecture draws inspiration from AutoRAG [59], a framework that provides automated search capabilities for finding optimal pipeline configurations. This modular design enables systematic evaluation of different component combinations while maintaining clear interfaces between processing stages.

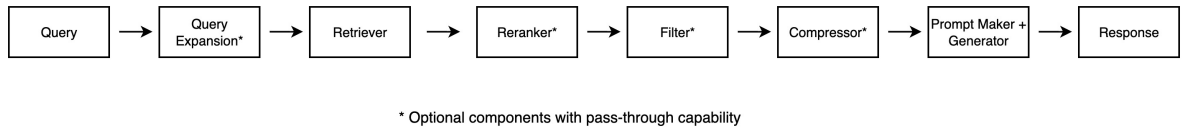


Figure 5.1.: Sequential RAG pipeline architecture with six processing stages from query to response generation. Components marked with asterisks support pass-through functionality, enabling the optimizer to bypass stages when they do not improve performance.

Modular Approach and Component Flow

The pipeline illustrated in Figure 5.1 follows a strictly sequential data flow where each component processes the output from its predecessor and passes refined results to the next stage. The architecture begins with an optional query expansion phase that enriches the original query, followed by retrieval that identifies potentially relevant documents from the corpus. Retrieved documents then flow through progressive refinement stages including reranking for improved relevance scoring, filtering to remove low-quality results, and compression to fit within context limits. Finally, the prompt maker combines the processed context with the query for the generator to produce the final response. Each component maintains standardized interfaces that accept specific input formats and produce outputs compatible with downstream components, enabling modular replacement during optimization. This sequential architecture ensures that each stage can focus on its specialized task while building

upon previous processing results. The pass-through option available in every component allows the optimizer to discover simplified pipelines by bypassing unnecessary processing stages.

Component Interactions and Dependencies

The pipeline architecture enforces specific constraints and dependencies between components to ensure valid configurations. The reranker's top-k parameter must not exceed the retriever's top-k value since the reranker can only reorder documents that were initially retrieved. When the reranker selects only a single document by setting top-k to one, the filter component automatically bypasses processing as no further selection is necessary. The filter component itself maintains a minimum output constraint, ensuring at least one document per query is retained even when all documents fall below the specified threshold, preventing empty inputs to downstream components. These dependencies create a complex optimization landscape where component interactions significantly influence overall performance. The handling of these constraints differs between optimization strategies, where global optimization must respect all constraints simultaneously while local optimization can handle component-specific constraints independently.

5.3. Pipeline Components

This section presents the six primary components that constitute the RAG pipeline, each offering multiple algorithmic choices and parameter configurations that collectively define the optimization search space. Each component represents a distinct processing stage with specialized functionality, from initial query understanding through final response generation. The diversity of algorithms within each component creates a heterogeneous search space combining discrete method selection with continuous hyperparameter tuning.

In addition to the six core components, the pipeline supports two categories of language models: open-source models from Hugging Face and proprietary models accessed through SAP APIs. These model sources differ in transparency, configurability, and deployment environment, which affects how the optimization framework interacts with them. Their characteristics are described in the final subsection of this section.

Query Expansion

Query expansion enhances the original user query to improve retrieval effectiveness by generating alternative formulations or additional context. The component offers four strategies: pass-through which uses the original query unchanged, query decomposition [23] that breaks complex queries into simpler sub-questions, HyDE (Hypothetical Document Embeddings) [22] which generates hypothetical answer documents to match against the corpus, and multi-query expansion that creates multiple query variations to capture different aspects of the information need. Each strategy except pass-through requires language model

support and introduces additional latency, creating a trade-off between retrieval improvement and computational cost that the optimization framework must balance.

Retriever

The retriever component performs the initial document selection from the corpus using either sparse or dense retrieval methods. Sparse retrieval employs BM25 [9] and its variants for keyword-based matching, providing efficient and interpretable results without requiring specialized infrastructure. Dense retrieval [10] uses embedding models to capture semantic similarity between queries and documents through vector representations. The retriever's top-k parameter determines the number of documents passed to subsequent stages, directly impacting both recall potential and computational requirements for downstream components. This component serves as the foundation for the entire pipeline's performance, as documents not retrieved cannot be recovered by later stages.

Reranker

The reranker refines the initial retrieval results using more computationally intensive but accurate relevance models. Available methods include pass-through for no reranking, MonoT5 [25] for sequence-to-sequence scoring, UPR (Universal Passage Reranking) [26], cross-encoders such as MiniLM [24, 12], late-interaction models like ColBERT [27], and lightweight rerankers including FlagEmbedding [29] and FlashRank [28]. Each reranking method typically offers multiple model variants that differ in size, training data, and performance characteristics. The reranker's top-k parameter must respect the retriever's output size while determining how many documents proceed to filtering, creating a critical bottleneck that balances quality against computational efficiency.

Filter

The filter component removes documents below quality thresholds using four distinct strategies that evaluate either similarity scores or retrieval scores. Pass-through bypasses filtering entirely when all retrieved documents are considered necessary. Threshold cutoff filters documents based on the retrieval scores from the previous component, removing those below an absolute threshold value. Percentile cutoff similarly uses retrieval scores but filters based on a percentile of the score distribution, retaining only documents above a specified percentile threshold. Similarity threshold cutoff recalculates each document's similarity with the original query and removes documents below a similarity threshold, providing query-aware filtering independent of retrieval scores. Similarity percentile cutoff also recalculates query similarity but applies percentile-based filtering relative to the content length distribution. The component ensures pipeline robustness by maintaining at least one document per query regardless of scores, preventing downstream component failures. The distinction between retrieval-score-based filtering and similarity-based filtering allows the optimizer to choose between trusting upstream component scores or performing independent relevance assessment,

adding another dimension to the optimization space.

Compressor

The compressor reduces document content to fit within generator context limits while preserving essential information. Available methods include pass-through for no compression, tree summarization [21] for hierarchical content reduction, Refine [21] for iterative summarization, linguistic feature-based extraction, and graph-based sentence selection methods. Compression becomes critical when filtered documents exceed token limits, requiring intelligent content selection that maintains answer-relevant information while discarding redundancy. The choice of compression method significantly impacts both the quality of generated responses and processing latency.

Generator

The generator produces final responses by combining compressed context with the original query through prompt templates. The component supports various language models with configurable parameters including temperature for response variability and maximum token limits for output length. Prompt maker strategies include simple template filling, long context reorder [33] to optimize context window usage, and window replacement [21] for managing extensive contexts. The generator's performance depends heavily on the quality of its input context, making it sensitive to all upstream component choices while simultaneously being influenced by the prompt template selection. The interaction between context quality and prompt strategy creates a complex optimization surface where optimal prompt templates may vary depending on the characteristics of the filtered and compressed documents.

Model Sources and Characteristics

The RAG pipeline uses two types of language models that differ in design and deployment: open-source models from Hugging Face and proprietary models accessed through SAP's API services.

Open-source models provide full access to their architecture and parameters, allowing detailed tuning of embeddings, tokenization, and generation settings. This flexibility supports fine-grained optimization and makes experiments easier to reproduce.

In contrast, SAP-provided models are available only through managed APIs. Their internal architecture and parameters are not publicly accessible, so they are treated as black-box systems. While this limits direct tuning, these models offer stable performance, strong integration with enterprise systems, and consistent inference times.

Using both model categories allows the framework to test optimization strategies in different environments. This setup helps assess whether methods that work well with open, tunable models also perform effectively with closed, service-based models.

5.4. Optimization Framework

This section presents the Bayesian Optimization framework employed to efficiently navigate the vast configuration space of the RAG pipeline. Given the combinatorial explosion of possible configurations from multiple components and their parameters, exhaustive evaluation is computationally infeasible, necessitating intelligent search strategies that can identify well-performing configurations within limited evaluation budgets. The optimization framework addresses this challenge through Bayesian Optimization, which builds probabilistic models of the objective function to guide the search toward promising regions of the configuration space. The framework encompasses both global optimization strategies that evaluate complete pipeline configurations to capture component interactions, and local optimization approaches that decompose the problem into component-wise optimization subproblems. Figure 5.2 illustrates the iterative optimization process, showing how configurations are sampled, evaluated, and used to update the surrogate model that guides future sampling decisions.

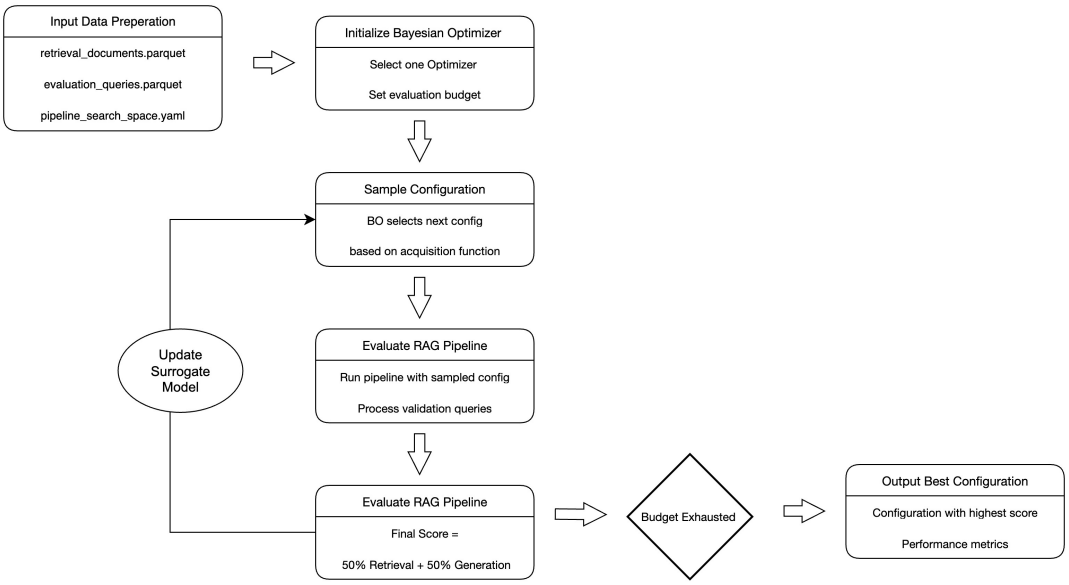


Figure 5.2.: Bayesian Optimization framework for RAG pipeline configuration. The iterative process begins with input data preparation including retrieval documents, evaluation queries, and search space definition. The optimizer samples configurations based on its acquisition function, evaluates them on the RAG pipeline, and updates its surrogate model with the results. This cycle continues until the evaluation budget is exhausted, outputting the configuration with the highest combined retrieval and generation score.

The optimization process shown in Figure 5.2 operates through a continuous feedback loop where each evaluation informs subsequent configuration sampling. The Bayesian Optimizer maintains a surrogate model that approximates the relationship between pipeline configurations and performance metrics, enabling it to predict which unexplored configu-

rations are likely to yield improvements. This approach significantly reduces the number of evaluations required compared to random or grid search methods, making it practical to optimize complex pipelines despite computational constraints. The framework must handle both discrete choices such as algorithm selection and continuous parameters like thresholds and top-k values, requiring sophisticated optimization techniques that can navigate mixed variable types while respecting component dependencies and constraints.

5.4.1. Bayesian Optimization Libraries for RAG Pipelines

The optimization experiments employ established Bayesian Optimization libraries to implement the described framework. SMAC3 [70] provides the Random Forest surrogate models with support for mixed parameter types and conditional dependencies. Optuna [61] implements Tree-structured Parzen Estimator as its primary approach and provides Gaussian Process models through BoTorch integration. RayTune [62] enables distributed evaluation of configurations across multiple compute nodes. HEBO [63] offers Heteroscedastic Gaussian Process models that account for varying noise levels across the parameter space. These implementations handle the computational complexities of surrogate model fitting and acquisition function optimization while providing the flexibility required for RAG pipeline optimization. Our experiments evaluate multiple surrogate models to determine which approach most effectively navigates the mixed discrete-continuous search space of RAG pipelines. Each surrogate model offers different trade-offs between modeling accuracy, computational efficiency, and ability to handle the hierarchical structure of component configurations.

For RAG pipeline optimization, the Bayesian Optimizer must handle the mixed-variable nature of the search space where each component involves both categorical choices of algorithms and continuous hyperparameters specific to the selected algorithm. This hierarchical structure creates conditional dependencies where certain hyperparameters only exist when specific algorithms are selected, such as temperature settings only being relevant when particular generators are chosen. The optimizer addresses this challenge through structured approaches that model these conditional relationships explicitly, enabling efficient exploration of the structured search space while maintaining the probabilistic framework that guides sampling decisions.

5.4.2. Search Space Definition

The RAG pipeline optimization involves navigating a vast hierarchical search space comprising both discrete algorithm selections and continuous hyperparameter configurations. This space exhibits a three-level hierarchical structure where the first level selects which components to activate, the second level chooses specific algorithms for each active component, and the third level determines the hyperparameter values for the selected algorithms.

The search space complexity increases substantially due to conditional dependencies where certain parameters only become relevant when specific algorithms are selected. For instance, selecting BM25 as the retriever requires choosing among different tokenizer options such as space [34], GPT-2[71], or other tokenization strategies, while dense retrieval methods

require embedding model selection. These conditional relationships create a tree-structured search space where different branches represent different algorithm choices, each with its own parameter subspace. The optimizer must efficiently navigate this structure, avoiding evaluation of irrelevant parameter combinations while ensuring comprehensive exploration of valid configurations.

The mixed-variable nature of the search space requires optimization techniques capable of handling both categorical and continuous variables simultaneously. Categorical variables include algorithm selections for each component, tokenizer choices for BM25, embedding model selections for dense retrieval, and prompt template strategies. Continuous variables encompass retrieval parameters such as top-k values, filtering thresholds and percentiles, compression ratios and iterations, and generation parameters including temperature and maximum token limits. The primary inter-component constraint ensures the reranker's top-k parameter cannot exceed the retriever's value, while the filter component maintains at least one document output regardless of threshold settings. These constraints reduce the effective search space while ensuring all evaluated configurations produce valid pipeline executions. The specific parameter ranges and configurations explored in our experiments, including the exact bounds for continuous variables and the complete set of algorithm choices, are detailed in the Experimental Setup chapter where we present the actual search space instantiation used for evaluation.

5.4.3. Multi-Objective Optimization

The optimization of RAG pipelines involves balancing multiple competing objectives beyond simply maximizing performance metrics. While achieving high retrieval and generation quality remains the primary goal, the computational cost and time required to find optimal configurations represent equally critical considerations. Bayesian Optimization addresses this multi-objective challenge by efficiently exploring the configuration space to minimize the number of expensive pipeline evaluations needed to identify high-performing configurations. This efficiency gain becomes particularly important when optimizing pipelines for production deployment where both model performance and optimization time directly impact system feasibility.

The framework considers three primary objectives that often conflict with each other. First, maximizing pipeline performance through the weighted combination of retrieval and generation metrics ensures the system produces accurate and relevant responses. Second, minimizing the number of configuration evaluations reduces computational costs and optimization time, making the approach practical for real-world applications. Third, reducing pipeline latency ensures that selected configurations meet production requirements for response time. These objectives create trade-offs where the highest-performing configuration may require extensive computational resources or exhibit unacceptable latency, while faster configurations may sacrifice quality.

In the implementation, latency per configuration is incorporated as a second optimization criterion alongside the primary metric score. The component-specific metrics are combined

into a final optimization objective defined as:

$$\text{Final Score} = 0.5 \times \text{Retrieval (last component)} + 0.5 \times \text{Generation Score}.$$

The retrieval score represents the F1 score from the last active retrieval component in the pipeline, as some components may be configured as pass-through operations. The final metric score remains the primary optimization goal, with the highest scoring configuration being selected as the best result, while latency serves as a secondary objective to guide the search toward efficient solutions. This dual-objective approach enables the discovery of configurations that balance quality with computational efficiency.

The time efficiency of Bayesian Optimization compared to exhaustive search methods represents a crucial advantage for practical deployment. While grid search must evaluate predetermined configurations regardless of their performance, and random search explores the space without learning from previous evaluations, Bayesian Optimization intelligently focuses computational resources on promising regions. This targeted exploration can reduce optimization time from days to hours while achieving comparable or superior performance. The surrogate model's ability to predict performance without evaluation enables rapid assessment of potential configurations, directing actual evaluations only to those most likely to improve upon current results.

5.4.4. Optimization Efficiency Strategies

Two distinct early stopping strategies enhance the multi-objective nature of the optimization by preventing wasted computation on inferior or sufficient configurations. Multi-fidelity early stopping evaluates configurations with progressively larger budgets, terminating unpromising configurations early based on performance at lower fidelity levels. This approach reduces computational waste on configurations that show poor performance even with limited evaluation resources.

Threshold-based early stopping terminates the optimization process under two conditions: when a configuration performs below a predefined threshold for any component, indicating poor quality, or when the overall score exceeds 0.9, suggesting that further optimization would yield minimal improvements. Poor configurations terminate early when components fail performance thresholds, saving the computational cost of completing full pipeline evaluation. Exceptional configurations trigger optimization termination when performance targets are met, recognizing that further search may yield marginal improvements at substantial computational cost.

These mechanisms implicitly encode a preference for computational efficiency alongside performance optimization, automatically balancing the exploration budget against the expected performance gains. Note that early stopping is not applied to grid search experiments, as exhaustive exploration of all predefined configuration options is required for complete baseline comparison. The framework thus achieves practical optimization that considers not only what configuration performs best, but also how quickly and efficiently that configuration can be discovered.

5.4.5. Local vs. Global Optimization Strategies

The optimization framework implements two distinct strategies for navigating the RAG pipeline configuration space, each with different assumptions about component interactions and computational trade-offs. Global optimization treats the entire pipeline as a single black-box function, sampling complete configurations that specify algorithms and parameters for all components simultaneously. This approach captures complex interactions between components where upstream choices influence downstream performance, such as how retriever quality affects optimal reranker selection or how document characteristics impact compression effectiveness. The global strategy requires evaluating full pipeline configurations for each sample, resulting in higher computational cost per evaluation but potentially discovering synergistic component combinations that local approaches might miss.

Local optimization decomposes the pipeline optimization into separate subproblems, optimizing each component independently while holding others fixed. This strategy assumes that components can be optimized in isolation, treating each module as an independent optimization problem with its own objective function based on component-specific metrics. Local optimization significantly reduces the search space complexity by transforming a single high-dimensional problem into multiple lower-dimensional problems, enabling more thorough exploration within each component's configuration space.

The computational efficiency of local optimization comes from its ability to evaluate components individually while reusing results from previously optimized stages. The process begins with optimizing the first component in the pipeline, then passes its best output to the next component as fixed input data. Each subsequent component optimization uses the preprocessed data from all previous stages rather than recomputing them, only calculating its own component-specific metrics. For example, when optimizing the reranker, it receives the retriever's output documents as input and only computes reranking metrics, avoiding the need to re-execute retrieval or any downstream processing. This sequential optimization with data passing between stages dramatically reduces computational cost and processing time. The efficiency gain allows for more configuration samples within the same computational budget, potentially achieving better component-level performance through more thorough exploration of each module's search space. However, this approach may miss important interdependencies where optimal configurations depend on upstream and downstream component choices, potentially leading to suboptimal end-to-end performance despite strong individual component metrics.

Both strategies incorporate early stopping mechanisms but apply them differently. Global optimization implements bidirectional early stopping, terminating poor configurations when components fail to meet minimum thresholds and ending the entire optimization process when exceptional configurations exceed performance targets. Local optimization only employs early stopping for high-performing components, bypassing subsequent retrieval-focused modules when earlier components achieve sufficient quality. These different stopping criteria reflect the fundamental difference in optimization objectives, with global optimization targeting end-to-end performance while local optimization focuses on component-specific excellence. The choice between strategies involves balancing the desire to capture component

interactions against computational constraints and the dimensionality of the search space.

5.5. Evaluation

The evaluation framework employs a comprehensive set of metrics to assess pipeline performance at both component and end-to-end levels, providing the necessary feedback signals for Bayesian Optimization to effectively navigate the configuration space. Since the optimizer learns from these metric scores to guide its sampling decisions, the choice and calculation of evaluation metrics directly influences optimization effectiveness. The evaluation methodology draws inspiration from the AutoRAG framework while extending it with additional metrics and refined calculations to ensure robust performance assessment across diverse pipeline configurations.

5.5.1. Component-Specific Metrics

Evaluation at the component level enables fine-grained performance assessment and supports local optimization strategies by providing targeted feedback for each pipeline stage. For components focused on retrieval including the retriever, reranker, and filter, the framework implements standard information retrieval metrics calculated against groundtruth document sets. The implementation supports multiple retrieval metrics including F1, recall, precision, NDCG (Normalized Discounted Cumulative Gain) [72], MAP (Mean Average Precision) [34], and MRR (Mean Reciprocal Rank) [73], allowing flexible evaluation based on task requirements. When multiple metrics are specified, the framework computes their arithmetic mean to provide a single optimization objective, though our experiments primarily utilize F1 score as it balances precision and recall considerations. These metrics evaluate document-level retrieval quality by comparing retrieved document identifiers against ground-truth relevant documents, providing clear signals about each component's ability to identify and rank relevant content.

The compressor component requires specialized evaluation metrics that assess information preservation during content reduction. The framework implements token level evaluation metrics including token F1, token recall, and token precision, which measure the overlap between tokens in the compressed passage and tokens in the ground truth answer. Token recall captures the proportion of answer tokens that appear in the compressed output, token precision measures what fraction of compressed tokens are relevant to the answer, and token F1 provides a balanced assessment combining both perspectives. These metrics evaluate compression effectiveness by directly comparing the compressed text against ground truth answers on a per token basis, ensuring that compression retains information necessary to answer the query while removing irrelevant content. Our experiments primarily utilize token F1 as the optimization objective for compression, as it provides a balanced signal that prevents both over compression leading to answer relevant information loss and under compression resulting in unnecessary token usage.

Query expansion evaluation presents unique challenges as its quality cannot be measured

directly but only through its impact on downstream retrieval performance. In our framework, query expansion and retriever evaluations are unified, with the pass through option serving as a baseline where original queries are used without expansion. This integrated approach allows the optimizer to learn whether query expansion techniques improve retrieval metrics (F1, recall, precision) for specific dataset characteristics. The evaluation framework treats the query expansion and retriever combination as a single evaluation unit, computing retrieval metrics on the final retrieved documents regardless of whether queries were expanded. This design eliminates redundant evaluation while allowing the optimizer to discover when query expansion adds value versus when simpler pass through retrieval suffices.

The prompt maker component, unlike query expansion, cannot be bypassed as it performs the essential function of formatting retrieved context with the query for generation. Since prompt maker lacks standalone metrics, its evaluation occurs indirectly through generation performance. Different prompt templates such as Fstring, long context reorder, or window replacement affect how effectively the generator utilizes retrieved information, with this impact captured in generation metrics. The evaluation framework thus handles components without direct metrics by measuring their contribution to downstream performance, allowing the optimizer to select configurations that maximize end-to-end effectiveness rather than isolated component scores.

5.5.2. LLM-Based Compressor Evaluation

While traditional metrics provide valuable signals for most components, certain pipeline stages require specialized evaluation approaches to capture their true effectiveness. The compressor component exemplifies this challenge, where conventional token based metrics consistently favor uncompressed text regardless of the compression algorithm employed. This bias toward original text prevents the optimizer from discovering effective compression strategies, as token F1 scores invariably decrease when any compression is applied, even when the compressed text retains all answer relevant information.

Beyond the inherent limitations of token F1 metrics, the evaluation datasets themselves introduce additional challenges. The ground truth answer sets may not capture all valid retrieved passages that contain correct information. A retrieved passage might provide accurate and relevant information for answering the query but receive a low score because its ID does not appear in the annotated answer set. This occurs when multiple passages in the corpus contain similar or complementary information, yet only a subset receives ground truth labels. The retrieval metrics therefore penalize correct retrievals that fall outside the predetermined answer set. Such incomplete annotations create misleading signals for the optimizer, as components that successfully retrieve relevant information receive unfairly low scores. This issue particularly affects the optimization of retrieval and reranking components, where the optimizer may discard effective configurations based on artificially deflated metrics.

To address these limitations, we developed an LLM-based evaluation approach specifically for the compressor component that assesses compression quality through multiple dimensions. The LLM evaluator employs GPT-4o to score compressed contexts on a scale from 0.0 to 1.0 based on four weighted criteria. Atomic fact preservation (50% weight) verifies

that all specific facts from the ground truth answer remain in the compressed context, with penalties for missing critical information such as methods, measurements, or specific comparators. Completeness (15% weight) evaluates whether the compressed context contains sufficient information to reconstruct the exact ground truth answer. Relevance and accuracy (20% weight) penalizes irrelevant or incorrect information that could mislead the generator. Efficiency and precision (15% weight) rewards brevity when all atomic facts are preserved while penalizing excessive length with unrelated content.

This evaluation across multiple criteria enables the optimizer to discover compression strategies that balance information preservation with context reduction, rather than defaulting to pass through configurations. In addition, the LLM-based approach also helps correct unfairly low scores in retrieval evaluation when correct answers are present in the compressed text but absent from the annotated ground truth set. By rewarding factual correctness and relevance directly, the evaluator reduces the bias of token-based metrics and incomplete answer sets, providing fairer signals for both compression and retrieval components.

The LLM evaluator processes samples in batches to improve efficiency while maintaining consistent scoring across evaluations. When combined with traditional metrics for other components, this evaluation across multiple objectives ensures that each pipeline stage receives appropriate signals that reflect its actual contribution to overall performance. The detailed evaluation prompt and implementation are provided in Appendix A.

5.5.3. End-to-End Generation Metrics

End-to-end generation metrics evaluate the quality of the final system output, measuring how well the generated response matches expected answers. These metrics provide the ultimate assessment of pipeline effectiveness, capturing the combined impact of all component choices on generation quality. The framework implements multiple complementary generation metrics from AutoRAG to capture different aspects of response quality, recognizing that no single metric fully characterizes generation performance.

The evaluation framework incorporates standard natural language generation metrics with specific interpretations for RAG systems. BLEU (Bilingual Evaluation Understudy) [74] measures the extent to which words in the LLM generated response appear in the ground truth answer, providing an assessment based on n-gram precision. ROUGE (Recall Oriented Understudy for Gisting Evaluation) [75] evaluates the extent to which words from the ground truth answer appear in the generated response, focusing on recall rather than precision. METEOR (Metric for Evaluation of Translation with Explicit ORdering) [76] computes the harmonic mean of unigram precision and recall, with recall weighted more heavily than precision, while also incorporating stemming and synonym matching beyond exact word matching. This metric was designed to address limitations in BLEU and to achieve better correlation with human judgments at the sentence level.

Beyond lexical matching, the framework includes semantic similarity capabilities through semantic score, which measures semantic similarity between ground truth and generated responses using an embedding model [77]. This provides assessment of semantic alignment regardless of specific word choices, capturing when responses convey correct information

through different phrasings. While the framework also implements BERT Score and G-Eval metrics for comprehensive evaluation capabilities, our experiments focus on the core metrics that provide stable and interpretable signals for optimization.

For optimization purposes, our experiments compute the arithmetic mean of BLEU, METEOR, ROUGE, and semantic score to create a balanced generation quality measure. This ensemble approach prevents optimization from overfitting to any single metric's biases, as BLEU emphasizes precision while ROUGE focuses on recall, and semantic metrics capture meaning beyond surface forms. The combined generation score, when integrated with retrieval metrics in a weighted way, provides the complete performance signal that guides the optimizer toward configurations performing well at both information retrieval and accurate response generation. The decision to use these four metrics balances comprehensive evaluation with computational efficiency, as these metrics provide reliable signals without the additional computational overhead of more complex evaluation methods.

5.5.4. RAGAS Evaluation Framework

The RAGAS (Retrieval Augmented Generation Assessment) framework [37] provides advanced LLM-based evaluation metrics that assess RAG pipeline quality through multiple dimensions beyond traditional lexical matching. While RAGAS offers comprehensive pipeline evaluation capabilities, its computational expense and time requirements make it impractical for iterative optimization where hundreds of configurations require evaluation. Therefore, our methodology employs RAGAS exclusively for post-optimization validation, evaluating only the final best configurations identified by different optimization approaches to verify that our primary metrics accurately guided the optimization process.

The RAGAS metrics employed in our validation encompass both retrieval and generation quality dimensions. Faithfulness measures the factual consistency between generated responses and retrieved context, ensuring that the model does not hallucinate information beyond what the context supports. Response Relevancy evaluates how well the generated answer addresses the specific question asked, using both LLM judgment and assessment based on embeddings to identify irrelevant or off-topic content.

For retrieval quality assessment, RAGAS provides context-aware metrics that evaluate the retrieved documents' utility. LLM Context Precision Without Reference measures the precision of retrieved context by evaluating whether each retrieved passage contains information relevant to answering the query, using an LLM judge without requiring reference passages. Context Recall assesses the proportion of answer relevant information successfully retrieved from the corpus, requiring reference contexts to measure retrieval completeness. These metrics offer more nuanced evaluation than traditional retrieval metrics by considering semantic relevance at the passage level.

The framework additionally includes direct answer quality metrics through Factual Correctness and Semantic Similarity. Factual Correctness evaluates whether the generated response contains accurate information when compared to ground truth answers, computing an F1 based score for factual alignment. Semantic Similarity measures the semantic alignment between generated and reference answers using embedding based comparison. The imple-

mentation computes mean scores across retrieval metrics, generation metrics, and an overall RAGAS mean score, providing comprehensive validation that examines multiple quality dimensions.

The validation results using RAGAS metrics serve to confirm that configurations achieving high scores on our primary optimization metrics also perform well on these sophisticated evaluation criteria. This validation approach ensures that the optimization process, guided by computationally efficient metrics, produces configurations that perform well not only in traditional metrics but also in advanced dimensions such as faithfulness and contextual relevance. The strong correlation between optimization metrics and RAGAS scores validates our evaluation methodology while avoiding the computational burden of using RAGAS throughout the optimization process.

6. Experimental Setup

This chapter details the experimental configuration used to evaluate the Bayesian Optimization approaches for RAG pipeline optimization. Building on the theoretical framework from the previous chapter, we present the concrete experimental choices made to validate our four research questions across different domains and optimization strategies.

The experimental setup encompasses three main components. First, we present the implementation details, including model configurations, hardware specifications, and software frameworks necessary for reproducibility. Second, we outline the experimental procedures governing optimization runs, evaluation budgets, and baseline comparisons. Third, we describe the detailed experimental designs corresponding to each research question, including the baselines used, the model sets evaluated, and the number of samples selected for each optimization process. The experiments explore a search space exceeding 50 million possible configurations while maintaining computational feasibility through strategic sampling and early stopping mechanisms.

6.1. Implementation Details

Model Configurations

The optimization framework evaluates a diverse set of models spanning both proprietary APIs and open source implementations to comprehensively explore the configuration space. Model access was provided through two distinct infrastructures: SAP's AI Core platform for commercial models and local VLLM deployment for open source models, enabling comparison across different model architectures and sizes.

For query expansion and generation tasks, the framework includes commercial models accessed via SAP APIs: Claude 4 Sonnet (Anthropic), GPT-3.5 Turbo (OpenAI), Mistral Large Instruct (MistralAI), and Gemini 2.0 Flash (Google). Open-source alternatives, sourced from the Hugging Face model repository, range from compact models such as TinyLlama 1.1B and Qwen 2.5 1.5B to mid-sized models including Llama 2 7B and Llama 2 13B. This diversity allows the optimizer to discover whether task performance correlates with model size or if smaller, faster models suffice for specific pipeline components.

Embedding models for dense retrieval include OpenAI's text-embedding series (ada-002, 3-small, 3-large), Google's Gemini embeddings, and open-source alternatives such as BGE models (small, m3), all-mpnet-base-v2, and domain-specific models like rubert-tiny2. The embedding models are integrated with ChromaDB for persistent vector storage, with embeddings generated on demand and cached for subsequent runs.

Reranking components leverage specialized cross-encoder models including MonoT5 variants (base and large), multiple BAAI BGE rerankers, FlashRank models, and sentence transformer cross-encoders. Additionally, Cohere's Rerank v3.5, accessed via the SAP API, provides a commercial reranking option. These rerankers offer varying trade offs between accuracy and latency, allowing the optimizer to select appropriate models based on dataset characteristics and performance requirements.

All models support configurable parameters including temperature (0.0 to 1.0), maximum token limits, and model specific settings such as batch sizes for embedding generation. The configuration space includes both the discrete choice of model selection and continuous hyperparameter tuning within each model, creating a hierarchical optimization problem. The complete configuration specifications detailing all available models, and their parameters are provided in Appendix B for reproducibility.

Implementation Environment

The experiments were conducted on a high-performance computing cluster equipped with GPUs providing 80GB of memory for model loading and inference. Hardware configuration significantly influenced component performance variability, particularly for computationally intensive operations like reranking. During experiments, identical reranker configurations exhibited substantial runtime variations across different runs, with some executions taking several times longer than others despite using the same dataset and parameters. This variability likely stems from GPU resource contention in the shared cluster environment, where concurrent jobs and memory allocation patterns affect individual component performance. Such performance fluctuations impact Bayesian Optimization decisions, as execution time serves as one of the optimization objectives alongside quality metrics.

The optimization framework was implemented in Python using multiple Bayesian optimization libraries to compare different surrogate modeling approaches, including SMAC3 for Random Forest surrogates, Optuna for Tree-Structured Parzen Estimator and Gaussian Process surrogates, Ray Tune for the BOHB implementation, and HEBO for Heteroscedastic Gaussian Process–based optimization. The AutoRAG framework provided the foundational pipeline architecture including data schemas, evaluation metrics, and component interfaces. Complete hardware specifications and software library versions are detailed in Appendix C.

6.2. Experimental Procedures

6.2.1. Reproducibility Measures

The experimental framework employs standard reproducibility practices, though complete determinism remains challenging due to the inherent stochasticity in language model inference and distributed computing environments. Random seeds were fixed at 42 for Bayesian Optimization algorithms and model initialization. Python environments were managed through virtual environments with pinned package versions as specified in the previous section, ensuring consistent dependencies across experimental runs. While generator model

temperatures varied as part of hyperparameter optimization (0.0 to 1.0), evaluation models including the LLM-based compressor evaluator maintained temperature 0 to minimize scoring variability.

Despite these measures, evaluation noise persists from multiple sources that affect optimization trajectories. Language models, particularly when accessed through APIs, exhibit nondeterministic behavior even with temperature set to 0, producing slightly different outputs for identical inputs across runs. Embedding models accessed through SAP APIs demonstrate similar variability, where identical queries to the same model can return different retrieved documents across runs. For example, using Gemini embeddings with top-k=2 on FiQA, examination of the first 10 queries showed that 2 queries retrieved completely different document IDs between runs, directly impacting retrieval metrics. Latency measurements vary due to GPU resource contention, API response times, and network conditions, introducing variance into the execution time objective. These sources of noise compound to create situations where identical configurations yield different scores and latencies across runs.

The impact of this nondeterminism on Bayesian Optimization is particularly notable. Since the optimizer adapts its sampling strategy based on observed results, small variations in early evaluations can lead to divergent search trajectories. In HotpotQA experiments with SMAC3, identical configurations evaluated at different points produced score variations exceeding 0.02 with latency differences over 80 seconds. These differences cause the surrogate model to build different representations of the objective landscape, leading subsequent samples to explore different regions of the configuration space. Consequently, while the optimizer consistently identifies high performing configurations, the specific configurations discovered and their evaluation order vary across runs. While perfect reproducibility cannot be guaranteed due to the stochastic nature of LLM-based components and API services, these measures ensure that the experimental methodology can be replicated and that similar performance ranges can be achieved.

6.2.2. Optimization Runs

The standard optimization experiments employed 50 configuration evaluations as the budget for Bayesian Optimization methods, determined through preliminary analysis showing that most optimizers achieved substantial performance improvements within this range while maintaining reasonable computational costs. Additional experiments with varying budgets were conducted to examine whether extended exploration yields proportional performance gains.

The early stopping thresholds for poor performing configurations were set based on component specific minimum viable scores: retrieval and query expansion components required scores above 0.1, rerankers above 0.2, filters above 0.25, and compressors above 0.3. These thresholds were empirically determined from preliminary experiments as the minimum scores below which downstream components cannot recover meaningful performance. When any component fails to meet its threshold, the entire configuration evaluation terminates immediately, saving computational resources for more promising candidates. For high performing configurations, optimization terminates when the combined score exceeds 0.9, as

further improvements beyond this threshold typically yield marginal gains.

Due to computational constraints, the grid search baseline employed a component-wise evaluation strategy rather than exhaustive full pipeline search. This approach optimizes each component independently, selecting the best configuration per component and combining them for the final pipeline, providing an approximation of the best achievable performance. The grid search configuration space was necessarily restricted compared to Bayesian Optimization: continuous parameters such as temperature and thresholds were fixed to typical values (e.g., 0.5 for compression ratio, 0.0 for temperature), top-k ranges were limited to [2, 4] rather than the broader ranges available to Bayesian methods, and only the most promising algorithm variants were included based on preliminary experiments. This substantially reduced search space contrasts with the millions of possible configurations available to adaptive methods, highlighting the computational advantage of guided exploration over exhaustive search.

To account for the nondeterministic effects described in previous section, multiple independent runs were conducted for each experimental condition. To compare different Bayesian Optimization algorithms, each optimizer was evaluated with three independent runs to assess performance stability and variance. The best performing optimizers identified from these comparisons were then applied to subsequent research questions. Random search was allocated four times the budget of Bayesian methods (200 configurations for standard experiments) to provide a stronger baseline, as random search typically requires more samples to achieve comparable performance without learning from previous evaluations. The convergence criterion for all methods was budget exhaustion rather than performance plateau, ensuring fair comparison across different optimization strategies that may exhibit varying convergence rates.

6.3. Experimental Designs

This section details the specific experimental configurations designed to address each research question. The experiments progressively build upon each other, with findings from earlier questions informing the design of subsequent investigations.

6.3.1. Comparative Analysis of Bayesian Optimization Algorithms

To investigate which Bayesian Optimization algorithms are most effective for optimizing RAG pipelines, we evaluated seven strategies derived from different combinations of surrogate models, multi-fidelity techniques, and multi-objective formulations (Table 6.1). In total, four surrogate models were selected: Random Forest, Tree-structured Parzen Estimator (TPE), Gaussian Process, and Heteroscedastic Gaussian Process. Two multi-fidelity methods, Successive Halving and Hyperband, were used to allow early stopping of unpromising configurations with lower budgets.

All experiments in this research question utilized open-source models from Hugging Face to ensure reproducibility and accessibility. Each Bayesian Optimization (BO) method was

allocated 50 sampled configurations, and results were compared against a random search baseline consisting of 50 randomly sampled configurations drawn from a pool of more than 34,000 possible combinations. In total, this setup resulted in 24 optimization runs, including the random baseline.

For the compressor component, we did not use the LLM-based evaluator; instead, we relied on token-wise precision, recall, and F1 as provided by the AutoRAG evaluation framework. Each optimizer was executed multiple times on three datasets, and only the highest score achieved by each optimizer is reported in the results tables.

The optimization objectives were twofold:

1. **Multi-objective optimization**: maximize evaluation scores while minimizing total pipeline execution time (latency).
2. **Multi-fidelity optimization**: evaluate performance under varying computational budgets (e.g., epochs, dataset subsets) before full training completes.

Performance comparison was carried out across global optimization runs to identify the most effective BO strategy.

Table 6.1.: Overview of Bayesian Optimization strategies

#	Library	Surrogate Model	Multi-Fidelity	Multi-Objective
1	SMAC3	Random Forest + Hyperband	Yes	Yes
2	SMAC3	Random Forest + Successive Halving	Yes	Yes
3	SMAC3	Random Forest	No	Yes
4	Optuna	Tree-structured Parzen Estimator (TPE)	No	Yes
5	Optuna	Gaussian Process	No	Yes
6	RayTune	TPE + Hyperband	Yes	No
7	HEBO	Heteroscedastic Gaussian Process	No	Yes

6.3.2. Scope of Bayesian Optimization in RAG Pipelines

Building upon the algorithm comparison results, this experiment investigates whether optimization should target the entire pipeline globally or individual components locally. Based on insights from the previous research question, we selected two representative Bayesian optimization algorithms for further evaluation in this experiment.

To ensure sufficient configuration diversity within each pipeline component, we expanded the parameter options available at the component level. As a result, the overall search space increased from approximately 34,000 to more than 50 million possible configurations, reflecting the cumulative effect of the extended component-level settings.

For global optimization, we allocated 50 sampled configurations per optimizer. This decision was informed by our previous analysis, where 50 samples provided a reasonable balance between runtime and the ability to identify competitive configurations relative to grid search.

For local optimization, each component was restricted to 20 sampled configurations. Since the configuration space at the component level is much smaller than the global search space, fewer samples are required to cover it meaningfully. At the same time, allocating 20 samples ensures that the optimizer has a sufficient number of initial points to train its surrogate model and refine the search effectively. This choice therefore reflects a compromise between coverage of the component-level space and computational efficiency.

To ensure a comprehensive evaluation, we conducted experiments using two distinct sets of models:

- A configuration consisting solely of embedding and generator models provided by SAP. These models represent strong industrial baselines and are designed for enterprise-grade applications, making them a valuable benchmark for assessing optimization strategies.
- A configuration using open-source embedding and generator models from Hugging Face, combined with compressor modules that rely on OpenAI models. This setup allows us to evaluate optimization strategies under a more diverse set of publicly available models.

By evaluating both model sets, we aim to test the robustness of optimization strategies across different deployment contexts, ranging from proprietary industrial models to open-source alternatives. The baseline for this research question is a component-wise grid search. Since the configuration space for global optimization extends to 50 million possible combinations, grid search is computationally infeasible at the global level. Instead, grid search is applied locally within each component to provide an approximate view of the best possible configuration achievable in isolation.

In total, we conducted 30 experiments: 15 using SAP-provided models (via API access) and 15 using open-source models on GPUs. For each dataset and model set, we performed one component-wise grid search, one global optimization run with each selected Bayesian optimization algorithm, and one local optimization run per algorithm.

6.3.3. Determining Optimal Sample Size for Configuration Exploration

Having established the effectiveness of different optimization scopes, we next investigate the impact of sample size on optimization performance. We focus on global optimization, where the search space is largest and the potential benefits of increased sampling are most pronounced. Based on previous findings showing that local optimization with limited samples already approximates grid search performance, we concentrate our analysis on the more challenging global optimization scenario.

We conducted experiments comparing two sample sizes for Bayesian optimization: 50 configurations serving as the baseline established in earlier experiments, and 100 configurations to test whether doubling the sample size yields continued improvement. As a baseline for comparison, we use random search with 200 configurations. Since exhaustive grid search is computationally infeasible in the global space (exceeding 50 million combinations),

random search with substantial sampling provides a reasonable upper bound for expected performance without intelligent exploration.

All experiments utilize Hugging Face models, which exhibited greater sensitivity to optimization strategies and thus provide a more informative test case for analyzing sample size effects. We evaluate both selected Bayesian optimization algorithms to ensure robust conclusions about the relationship between sample size and optimization performance.

6.3.4. Outcome Robustness Across Datasets

The final experimental design investigates how dataset characteristics influence the performance of Bayesian optimization methods. This experiment builds upon the configurations from the previous research question, focusing on the optimization results obtained with 100 samples for each Bayesian optimization algorithm and 200 samples for random search. To analyze differences across domains, we employ Pareto front charts to visualize the trade-off between performance and latency for each dataset—SciFact (scientific literature), FiQA (financial documents), and HotpotQA (multi-hop reasoning). This visualization allows us to examine how Bayesian optimization behaves under varying dataset characteristics and to assess how domain complexity affects the optimization dynamics.

7. Evaluation Results

In this chapter, we present the evaluation results of our proposed optimization framework. The experiments were conducted on three datasets: SciFact, FIQA, and HotpotQA. For each dataset, we randomly selected 200 queries to construct validation sets used consistently across all experimental runs. Each query is paired with a groundtruth context and retrieval identifiers, while generation groundtruth answers were obtained using the data generation framework provided by AutoRAG.

Two early-stopping strategies were employed during experimentation. The multi-fidelity algorithm-level early stopping mechanism was applied only in the initial algorithm comparison experiments to identify the most effective Bayesian optimization methods by allocating reduced budgets to unpromising configurations. In subsequent experiments, a component-level threshold-based early stopping strategy was employed to improve runtime efficiency by terminating low-performing configurations early.

As baselines, we adopted grid search for local optimization and random sampling for global optimization. Grid search was conducted over a reduced configuration space, restricted to promising parameter ranges to ensure computational feasibility. In total, our evaluation comprised 63 experimental runs, covering multiple datasets, optimization strategies, and Bayesian optimization methods.

Following the presentation of results for each research question, we discuss how these findings informed the optimization of our RAG framework and what adjustments proved most effective. This analysis not only addresses our initial research questions but also highlights practical insights and opportunities for future improvements in the design of RAG pipelines.

7.1. Comparative Analysis of Bayesian Optimization Algorithms

This section investigates which Bayesian Optimization algorithms are most effective for tuning RAG pipelines. Since RAG involves a large configuration space and multiple interacting components, the choice of optimization algorithm directly affects both the quality of results and the efficiency of the search process. Prior works in related domains have applied methods such as random search, grid search, and Optuna's TPE [68], but it remains unclear which strategies perform best in the context of complex RAG pipelines.

To address this, we compare a set of representative BO approaches that differ in their surrogate models, use of multi-fidelity techniques, and support for multi-objective optimization.

We report results separately for the SciFact, FIQA and HotpotQA datasets. For each dataset, we compare the seven BO strategies against the random baseline. Performance is evaluated

in terms of best score achieved, latency of the best configuration, and total time used.

We explicitly measure latency since it directly contributes to the overall optimization time. Lower latency per configuration reduces the total optimization time and therefore improves efficiency. In addition, latency provides an estimate of the expected processing time when deploying the configuration to handle 200 queries in practice, which is important for real world applications where users must balance performance against runtime.

SciFact Dataset

Performance comparisons on the SciFact dataset appear in Table 7.1. The best performance was achieved by SMAC3 with a Random Forest surrogate, reaching a score of 0.6759 with a latency of 81.18 seconds and a total optimization time of 1h 39m 46s. This outperformed all other methods on both score and stability. The next best results were obtained by Optuna with TPE (0.62878) and Optuna with Gaussian Process (0.62778), both achieving competitive performance with lower latencies of 63.82s and 74.67s respectively. In contrast, multi-fidelity approaches yielded notably lower performance. SMAC3 with Hyperband achieved a score of 0.5339, showing shorter runtime but reduced effectiveness, while RayTune with TPE + Hyperband produced the lowest score overall (0.36365).

These results indicate that full-budget optimization using SMAC3 with Random Forest is most effective on SciFact, while Optuna's surrogate models offer a good trade-off between runtime and performance.

FIQA Dataset

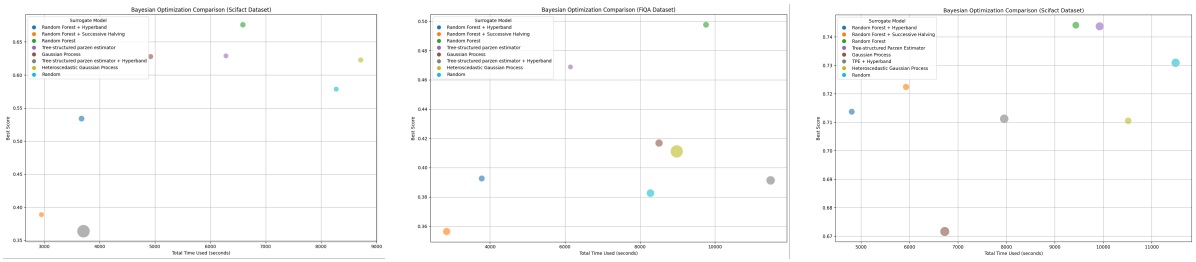
Results from the FIQA dataset experiments are presented in Table 7.1. The best performance was achieved by SMAC3 with a Random Forest surrogate, reaching a score of 0.4976 with a latency of 92.2 seconds and a total optimization time of 2h 42m 42s. The second best method was Optuna with TPE, which achieved a score of 0.4688 with lower latency (64.81s) and shorter total runtime (1h 42m 26s). In contrast, multi-fidelity methods such as SMAC3 with Successive Halving (0.3564) and RayTune with TPE + Hyperband (0.3913) performed considerably worse, suggesting that early stopping terminated promising configurations too early in this case.

HotpotQA Dataset

Results for the HotpotQA dataset are shown in Table 7.1. The best performance was achieved by SMAC3 with a Random Forest surrogate, reaching a score of 0.7441 with a latency of 123.83 seconds and a total optimization time of 2h 37m 13s. Optuna with the Tree-structured Parzen Estimator performed comparably well (0.7437) but required a longer runtime (2h 45m 25s). Unlike the SciFact and FIQA datasets, multi-fidelity methods such as SMAC3 with Successive Halving (0.7224) and SMAC3 with Hyperband (0.7137) achieved relatively stronger results, narrowing the performance gap with full-budget optimization.

Table 7.1.: Comparison of Bayesian Optimization algorithms on SciFact and FIQA datasets. Best scores for each dataset highlighted in bold.

#	Library	Surrogate Model	Best Score	Latency (s)	Total Time Used
SciFact Dataset					
1	SMAC3	Random Forest + Hyperband	0.53388	90.40	1h 01m 15s
2	SMAC3	Random Forest + Successive Halving	0.3998	66.67	49m 11s
3	SMAC3	Random Forest	**0.6759**	81.18	1h 39m 46s
4	Optuna	Tree-structured Parzen Estimator	0.62878	63.82	1h 44m 43s
5	Optuna	Gaussian Process	0.62778	74.67	1h 22m 02s
6	RayTune	TPE + Hyperband	0.36365	407.59	1h 11m 50s
7	HEBO	Heteroscedastic Gaussian Process	0.62255	72.76	2h 25m 16s
8	Optuna	Random	0.57856	68.94	2h 17m 54s
FIQA Dataset					
1	SMAC3	Random Forest + Hyperband	0.39259	97.18	1h 03m
2	SMAC3	Random Forest + Successive Halving	0.3564	141.58	47m 25s
3	SMAC3	Random Forest	**0.4976**	92.2	2h 42m 42s
4	Optuna	Tree-structured Parzen Estimator	0.4688	64.81	1h 42m 26s
5	Optuna	Gaussian Process	0.4168	143.51	2h 21m 49s
6	RayTune	TPE + Hyperband	0.3913	191.75	3h 11m 25s
7	HEBO	Heteroscedastic Gaussian Process	0.4111	391.08	2h 29m 41s
8	Optuna	Random	0.3826	153.06	2h 18m
HotpotQA Dataset					
1	SMAC3	Random Forest + Hyperband	0.7137	96.67	1h 20m 8s
2	SMAC3	Random Forest + Successive Halving	0.7224	105.97	1h 38m 49s
3	SMAC3	Random Forest	**0.7441**	123.83	2h 37m 13s
4	Optuna	Tree-structured Parzen Estimator	0.7437	172.18	2h 45m 25s
5	Optuna	Gaussian Process	0.6716	217.15	1h 52m 8s
6	RayTune	TPE + Hyperband	0.7112	191.75	2h 12m 35s
7	HEBO	Heteroscedastic Gaussian Process	0.7105	115.207	2h 55m 14s
8	Optuna	Random	0.7309	182.81	3h 11m 33s



(a) SciFact dataset (b) FIQA dataset (c) HotpotQA dataset

Figure 7.1.: Bayesian Optimization algorithm comparison across datasets. Different colors represent optimization algorithms. The x-axis shows total optimization time in seconds, the y-axis shows the best score achieved, and bubble size corresponds to latency of the best configuration.

Findings

Across all datasets in our experiments, the best results were consistently achieved by SMAC3 with a Random Forest surrogate without multi-fidelity extensions. This configuration reached the highest scores overall (0.4976 on FIQA, 0.6759 on SciFact, and 0.7441 on HotpotQA) and outperformed all other surrogate model combinations in terms of average metric scores.

Multi-fidelity approaches, including SMAC3 (random forest) with Successive Halving or Hyperband and RayTune with TPE + Hyperband, achieved shorter runtimes but generally lower scores on the FIQA and SciFact datasets. This indicates that early stopping often terminated promising configurations prematurely, making these methods less effective for RAG pipeline optimization under those conditions. A likely reason is that when configurations are evaluated on reduced budgets (e.g., fewer samples or partial epochs), the observed scores may not reflect their true potential. In addition, RAG evaluation metrics are subject to noise, and an underperforming early trial can lead to discarding a configuration that might have yielded strong results if fully evaluated.

However, on the HotpotQA dataset, which involves more complex multi-hop reasoning, multi-fidelity methods performed relatively better and narrowed the gap to full-budget optimization. This suggests that adaptive evaluation strategies may be more beneficial in settings where partial evaluations still provide informative signals about configuration quality.

Optuna with the Tree-structured Parzen Estimator (TPE) provided a competitive and consistent alternative across all datasets. It achieved scores of 0.4688 on FIQA, 0.6288 on SciFact, and 0.7437 on HotpotQA, with relatively low latencies ranging from 63s to 172s. Although it did not surpass SMAC3 with Random Forest in absolute performance, it also offered a strong balance between runtime efficiency and result quality, demonstrating stable behavior across both simpler (FIQA, SciFact) and more complex (HotpotQA) tasks.

The random baseline performed consistently below all Bayesian Optimization methods, with best scores of 0.3826 on FIQA, 0.5786 on SciFact, and 0.7309 on HotpotQA. This confirms that Bayesian Optimization is more sample-efficient and effective in navigating the configuration space than unguided search, achieving higher-quality configurations within the same

budget.

In summary, SMAC3 with Random Forest consistently delivered the best overall results in our RAG pipeline settings, while Optuna with TPE offered a comparably stable and reliable alternative with competitive performance across datasets. Multi-fidelity methods, although appealing for speed, did not yield reliable improvements in this setting.

Thus, to determine which Bayesian Optimization algorithm performs most effectively in our RAG pipeline settings, SMAC3 with a Random Forest surrogate emerges as the optimal choice, consistently delivering the highest scores across datasets. Optuna with TPE also proves to be a competitive alternative, offering a favorable balance between performance and latency. Based on these results, in the subsequent experiments we focus on SMAC3 and Optuna TPE as the representative optimizers for further investigation.

7.2. Scope of Bayesian Optimization in RAG Pipelines

This section investigates the scope at which Bayesian Optimization should be applied in RAG pipelines. Specifically, we compare global optimization, where BO samples full pipeline configurations across all components, against local optimization, where each component is optimized independently and the best configuration from each stage is passed on to the next. For these experiments, we employed an LLM-based evaluator for the compressor component to better assess compression quality beyond simple token matching metrics.

Global optimization has the advantage of capturing composite effects arising from interactions between components, but requires exploring an extremely large configuration space and incurs higher computational cost. In contrast, local optimization is more efficient and allows each module to be tuned individually, but it risks ignoring cross component synergies that may affect end-to-end generation quality.

The goal of this evaluation is to determine whether Bayesian Optimization should be applied globally or locally to most effectively improve generation performance in RAG pipelines. As these experiments employed an LLM-based evaluator for the compressor component, the reported scores are not directly comparable to those from the previous section, which used token matching metrics. The use of the LLM evaluator provides a more semantic measure of compression quality, which in turn influences the overall combined scores reported here.

7.2.1. SciFact

Local Optimization (SAP Models)

As presented in Table 7.2, grid search achieved the highest combined score of 0.6875, but required 18h 43m 34s. Both SMAC3 and Optuna TPE achieved competitive scores (0.6715 and 0.6720) with substantially reduced runtime: SMAC3 completed in 6h 06m (saving 67% of time compared to grid search), while Optuna TPE completed in 5h 54m (saving 69%). This indicates that sampling 20 configurations locally per component can approach the quality of exhaustive grid search while requiring only about one-third of the total runtime.

Table 7.2.: Local optimization results (SAP Models) across datasets. Scores are reported per component, with the final combined score shown in the last column.

Method	Query Expansion + Retriever	Reranker	Filter	Compressor	Prompt + Generator	Combined Score	Total Time Used
SciFact Dataset							
SMAC3	0.6702	0.7545	/	0.8025	0.5404	0.6715	6h 06m 49s
Optuna TPE	0.7643	0.7987	/	0.8497	0.4943	0.6720	5h 53m 49s
Grid Search	0.7747	0.8353	/	0.8702	0.5048	**0.6875**	18h 43m 34s
FIQA Dataset							
SMAC3	0.3822	0.3822	/	0.7125	0.4373	0.5749	11h 38m 00s
Optuna TPE	0.4522	0.4537	/	0.7100	0.4172	0.5636	7h 22m 00s
Grid Search	0.4646	0.4777	/	0.7370	0.4245	**0.5807**	20h 24m 38s
HotpotQA Dataset							
SMAC3	0.7858	0.7858	0.7858	0.7147	0.7055	0.7107	5h 49m 42s
Optuna TPE	0.9050	/	/	/	0.8177	**0.8584**	3h 22m 28s
Grid Search	0.9050	0.9050	0.9050	0.8260	0.6761	0.7510	15h 26m 43s

Table 7.3.: Global optimization results (SAP Models) across datasets.

Method	Combined Score	Total Time Used
SciFact Dataset		
SMAC3	0.6335	12h 46m 55s
Optuna TPE	**0.6663**	13h 42m 36s
FIQA Dataset		
SMAC3	0.5207	17h 48m 08s
Optuna TPE	**0.5482**	17h 02m 12s
HotpotQA Dataset		
SMAC3	0.7128	8h 38m 30s
Optuna TPE	**0.7236**	10h 16m 01s

Table 7.4.: Local optimization results across datasets (Hugging Face models). Scores are reported per component, with the final combined score shown in the last column.

Method	Query Expansion + Retriever	Reranker	Filter	Compressor	Prompt + Generator	Combined Score	Total Time Used
SciFact Dataset							
SMAC3	0.7430	0.8265	/	0.7288	0.4248	0.5768	1h 21m 55s
Optuna TPE	0.7806	0.8175	0.8262	0.7695	0.4133	0.5914	3h 16m 19s
Grid Search	0.7175	0.8137	/	0.7750	0.4568	**0.6159**	8h 35m 58s
FIQA Dataset							
SMAC3	0.1494	0.2236	/	0.4667	0.3757	0.4212	4h 44m 45s
Optuna TPE	0.3381	0.3537	0.3595	0.5647	0.3903	**0.4775**	2h 29m 50s
Grid Search	0.3525	0.3525	/	0.5450	0.3913	0.4682	8h 35m 03s
HotpotQA Dataset							
SMAC3	0.7950	0.7950	0.7950	0.7107	0.6843	**0.6975**	1h 48m 15s
Optuna TPE	0.7583	0.7583	0.7583	0.6570	0.5176	0.5873	1h 08m 42s
Grid Search	0.9200	0.9200	0.9200	0.8210	0.5712	0.6961	5h 04m 59s

Table 7.5.: Global optimization results across datasets (Hugging Face models).

Method	Combined Score	Total Time Used
SciFact Dataset		
SMAC3	**0.6313**	4h 24m 54s
Optuna TPE	0.6265	4h 01m 48s
FIQA Dataset		
SMAC3	0.4464	9h 08m 50s
Optuna TPE	**0.4868**	6h 12m 14s
HotpotQA Dataset		
SMAC3	0.5728	3h 56m 44s
Optuna TPE	**0.5944**	2h 10m 14s

Global Optimization (SAP Models)

When examining global optimization strategies (Table 7.3), Optuna TPE achieved the best score (0.6663) in 13h 43m, while SMAC3 reached 0.6335 in 12h 47m. Although both methods explored only 50 configurations globally, Optuna TPE came close to the performance of grid search (0.6875) while saving 27% of total runtime. SMAC3 was less effective, with a 5.2% lower score than Optuna TPE.

Local Optimization (Hugging Face Models)

The local optimization results with Hugging Face models (Table 7.4) reveal that grid search achieved the highest combined score of 0.6159, but required 8h 36m. Optuna TPE achieved a score of 0.5914 in 3h 16m, while SMAC3 reached 0.5768 in only 1h 22m. Both BO methods therefore achieved scores within 4–6% of grid search while saving between 62–84% of runtime, demonstrating that sampling 20 configurations locally per component is sufficient to approximate exhaustive search with much lower computational cost.

Global Optimization (Hugging Face Models)

In the global optimization setting (Table 7.5), SMAC3 achieved the best score (0.6313) in 4h 25m, while Optuna TPE achieved a slightly lower score (0.6265) in 4h 02m. Both methods explored 50 global configurations and outperformed their local optimization counterparts in terms of score, while still saving around 50% of runtime compared to grid search. Importantly, both global optimization methods surpassed the component-wise grid search baseline, demonstrating that jointly optimizing all pipeline components yields configurations superior to those found through exhaustive local tuning, while remaining substantially more efficient.

Across both SAP and Hugging Face model settings, Bayesian optimization methods achieved performance comparable to or exceeding grid search while requiring substantially less runtime. With limited samples (20 local per component or 50 global), Bayesian optimization was able to approach or surpass grid search quality while saving significant

computational cost. For SAP models, local optimization proved particularly competitive, achieving results comparable to grid search while reducing runtime by more than two-thirds. For Hugging Face models, both global Bayesian optimization methods outperformed the component-wise grid search baseline, achieving higher scores while requiring less than half of the runtime.

These results indicate that the relative effectiveness of local versus global optimization depends on the model set. Local optimization is more efficient and reliable for SAP models, while global optimization provides a better trade-off between performance and efficiency for Hugging Face models.

7.2.2. FIQA

Local Optimization (SAP Models)

Local optimization results (Table 7.2) demonstrate that grid search achieved the highest combined score of 0.5807, requiring 20h 24m 38s. SMAC3 achieved a score of 0.5749 in 11h 38m (saving 43% of time compared to grid search), while Optuna TPE reached 0.5636 in 7h 22m (saving 64%). Both BO methods achieved scores within 1-3% of grid search while significantly reducing runtime. Notably, SMAC3 matched grid search performance closely despite using only 20 samples per component.

Global Optimization (SAP Models)

Turning to global optimization approaches (Table 7.3), Optuna TPE achieved the best score (0.5482) in 17h 02m, while SMAC3 reached 0.5207 in 17h 48m. Optuna TPE's performance was 5.3% better than SMAC3. However, both global optimization approaches underperformed compared to grid search (0.5807), with Optuna TPE achieving 94.4% of grid search performance.

Local Optimization (Hugging Face Models)

For Hugging Face models in the local optimization setting (Table 7.4), Optuna TPE achieved the highest combined score of 0.4775 in just 2h 29m 50s, outperforming grid search (0.4682) while saving 71% of runtime. SMAC3 reached 0.4212 in 4h 44m 45s. This represents a case where Optuna TPE not only matched but exceeded grid search performance with substantially less computational cost.

Global Optimization (Hugging Face Models)

The global optimization experiments with Hugging Face models (Table 7.5) yielded particularly interesting results. Optuna TPE achieved the best score (0.4868) in 6h 12m 14s, while SMAC3 achieved 0.4464 in 9h 08m 50s. Optuna TPE outperformed both SMAC3 and grid search (0.4682), demonstrating that global optimization with 50 samples can identify superior

configurations compared to exhaustive component-wise search. The runtime savings were substantial, with Optuna TPE saving 28% of runtime compared to grid search.

For the FIQA dataset, the relative performance of optimization strategies differed across model sets. With SAP models, grid search achieved the highest overall score, but local Bayesian optimization, particularly SMAC3, closely matched grid search performance while substantially reducing runtime. Global optimization, by contrast, was less effective for this model set.

For Hugging Face models, Optuna TPE demonstrated consistently strong performance in both local and global optimization settings, surpassing the component-wise grid search baseline while requiring significantly less computational time. These findings suggest that the effectiveness of Bayesian optimization strategies depends on both dataset characteristics and underlying model architectures: local optimization is more effective for SAP models, whereas global optimization with Optuna TPE provides superior performance for Hugging Face models.

7.2.3. HotpotQA

Local Optimization (SAP Models)

Analysis of local optimization performance (Table 7.2) reveals that Optuna TPE achieved the highest combined score of 0.8584 in 3h 22m 28s, significantly outperforming grid search (0.7510) while saving 78% of runtime. SMAC3 reached 0.7107 in 5h 49m 42s. Notably, Optuna TPE skipped the reranker, filter, and compressor components entirely, relying solely on retrieval and generation optimization. This omission requires careful interpretation as the compressor evaluation was bypassed, potentially affecting score comparability. Grid search required 15h 26m 43s but achieved a lower score than Optuna TPE.

Global Optimization (SAP Models)

In contrast, global optimization results (Table 7.3) show more modest improvements. Optuna TPE achieved a score of 0.7236 in 10h 16m 01s, while SMAC3 reached 0.7128 in 8h 38m 30s. Both methods performed below grid search (0.7510), with Optuna TPE achieving 96.3% of grid search performance. The runtime for both methods was lower than grid search, with SMAC3 saving 44% and Optuna TPE saving 33% of time.

Local Optimization (Hugging Face Models)

Examining the Hugging Face model results for local optimization (Table 7.4), SMAC3 achieved the highest combined score of 0.6975 in 1h 48m 15s, slightly outperforming grid search (0.6961) while saving 65% of runtime. Optuna TPE reached 0.5873 in 1h 08m 42s, the fastest runtime but with lower performance. Grid search achieved 0.6961 in 5h 04m 59s. All three methods utilized the full pipeline including filter and compressor components.

Global Optimization (Hugging Face Models)

Finally, the global optimization experiments (Table 7.5) produced lower scores overall. Optuna TPE achieved a score of 0.5944 in 2h 10m 14s, while SMAC3 reached 0.5728 in 3h 56m 44s. Both methods underperformed compared to grid search (0.6961), achieving 85.4% and 82.3% of grid search performance respectively. However, both offered substantial runtime savings, with Optuna TPE requiring only 43% of grid search time.

For the HotpotQA dataset, optimization behavior varied notably between model sets. With SAP models, Optuna TPE's local optimization strategy of skipping intermediate components produced the highest score but raises comparability concerns. With Hugging Face models, SMAC3 demonstrated strong local optimization performance, matching grid search quality with significant runtime reduction. Global optimization generally underperformed local approaches on this dataset, suggesting that component-wise optimization may be more effective for multi-hop question answering tasks.

7.2.4. Summary: Local vs. Global Optimization Scope

Our experiments reveal that the optimal scope for applying Bayesian Optimization in RAG pipelines depends on both the model architecture and dataset characteristics. Neither local nor global optimization consistently dominates across all settings.

For SAP models, local optimization generally provides the best balance between performance and efficiency. On SciFact and FIQA datasets, local optimization with 20 samples per component achieved scores within 1-3% of exhaustive grid search while reducing runtime by 43-69%. On HotpotQA, Optuna TPE's local optimization even surpassed grid search performance (0.8584 vs. 0.7510), though this result requires careful interpretation due to component skipping. Global optimization with SAP models typically underperformed local approaches while requiring comparable or longer runtime.

For Hugging Face models, the optimal scope varies by dataset. Global optimization proved more effective on SciFact, achieving scores within 2-3% of grid search. On FIQA, both Optuna TPE's local and global optimization outperformed grid search (0.4775 and 0.4868 vs. 0.4682), suggesting that intelligent sampling can identify superior configurations that exhaustive search misses. However, on HotpotQA, local optimization with SMAC3 matched grid search performance while global optimization fell short.

The runtime savings are substantial across both optimization scopes. Local optimization typically saves 60-75% of grid search time, while global optimization saves 20-50%. The choice between local and global optimization thus depends on specific deployment requirements.

Notably, these experiments show that Optuna TPE often matches or outperforms SMAC3 in both efficiency and final metric scores. This variation from our initial algorithm comparison may stem from several experimental differences. The configuration space expanded from approximately 34,000 to over 50 million possible combinations, the LLM evaluator for compression provided more nuanced quality signals, and threshold-based early stopping changed the optimization dynamics. These modifications create a different optimization landscape that may favor TPE's probabilistic modeling approach in certain scenarios. Despite these

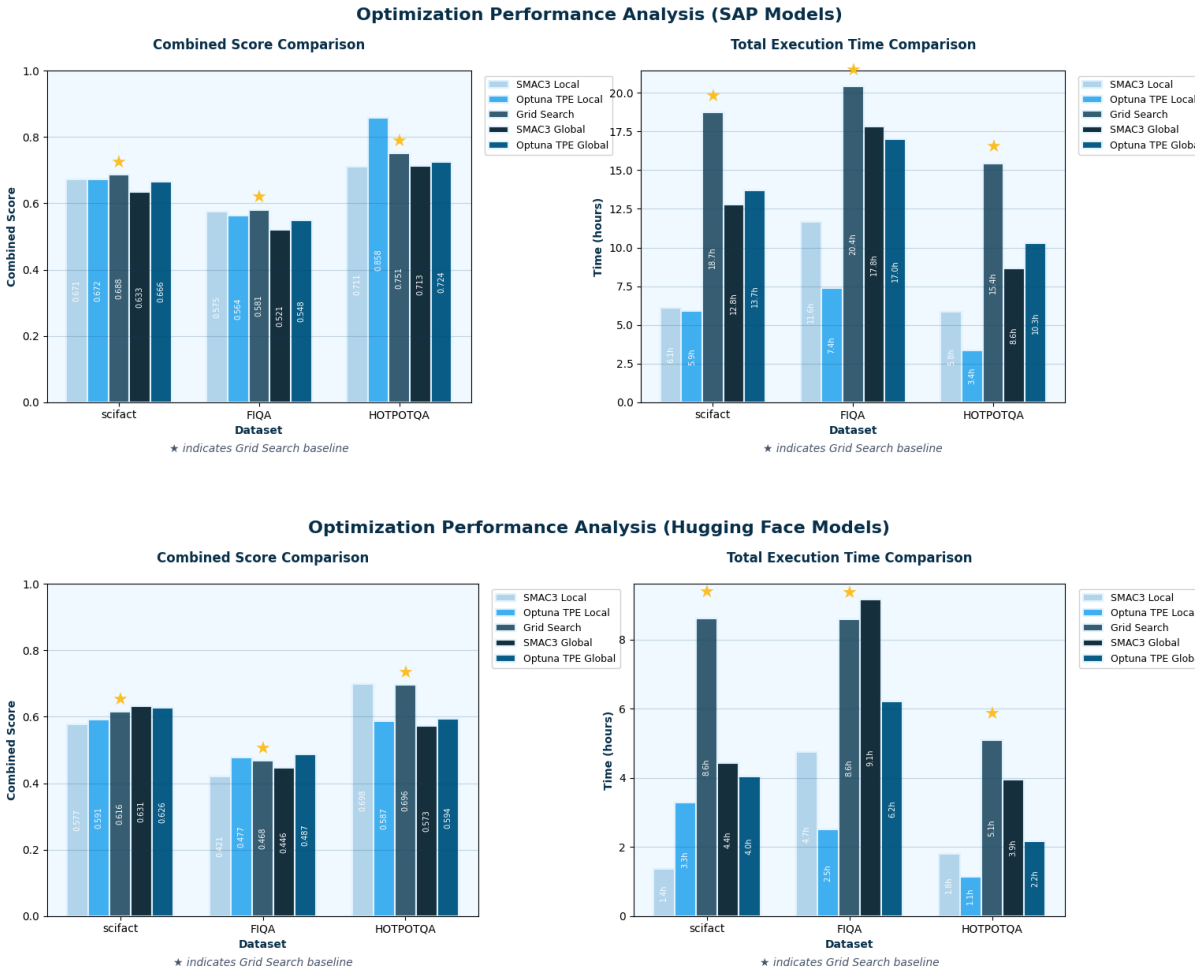


Figure 7.2.: Optimization performance analysis across datasets and model types. Top: SAP models. Bottom: Hugging Face models. Each chart shows combined scores (left panel) and total execution time (right panel) for local and global optimization methods. Grid search baseline is marked with stars.

variations, both SMAC3 and Optuna TPE demonstrate strong performance, confirming that their selection as representative optimizers remains valid.

Our findings indicate that practitioners should consider their specific context when choosing optimization scope. For enterprise deployments with established model sets, local optimization offers reliable performance with significant efficiency gains. For experimental settings with heterogeneous models, global optimization may uncover valuable cross-component synergies despite higher computational costs. The 20-sample local and 50-sample global configurations both prove sufficient to approach or occasionally exceed exhaustive search performance, validating Bayesian Optimization as an effective approach regardless of scope.

7.3. Determining Optimal Sample Size for Configuration Exploration

This section examines whether increasing the number of configurations explored in Bayesian Optimization leads to improved final scores in RAG tuning. While our initial analysis established that 50 samples provide reasonable performance relative to grid search, and we demonstrated that local optimization with 20 samples per component can approximate grid search results, the optimal sample size for global optimization remains unclear. This investigation explores whether expanding the sampling budget yields proportional performance gains or whether diminishing returns emerge at certain thresholds.

Table 7.6.: Results comparing sample sizes in global optimization across datasets. The "Top Configuration Distribution" column shows the distribution of configurations across score ranges.

Method	Best Score (Trial #)	Total Time Used	Top Configuration Distribution	Improved with More Samples?
SciFact Dataset				
Grid Search Local	0.6159	8h 35m 58s	-	-
Random 200	0.6086 (trial 158)	19h 48m 12s	1 config ∼0.58, 2 configs ∼0.57, 2 configs ∼0.56	-
SMAC3 50	0.6313 (trial 48)	4h 25m 04s	2 configs ∼0.62, 5 configs ∼0.61, 5 configs ∼0.60	-
SMAC3 100	**0.6366** (trial 54)	7h 02m 37s	2 configs ∼0.63, 5 configs ∼0.62, 6 configs ∼0.61	Yes (+0.0053)
Optuna TPE 50	0.6265 (trial 32)	4h 01m 48s	2 configs ∼0.61, 2 configs ∼0.60, 2 configs ∼0.59	-
Optuna TPE 100	0.6325 (trial 98)	6h 30m 28s	1 config ∼0.62, 3 configs ∼0.60, 4 configs ∼0.59	Yes (+0.0060)
FIQA Dataset				
Grid Search Local	0.4682	8h 35m 03s	-	-
Random 200	0.4536 (trial 143)	23h 53m 07s	1 config ∼0.45, 2 configs ∼0.43, 4 configs ∼0.42	-
SMAC3 50	0.4464 (trial 20)	9h 08m 00s	5 configs ∼0.44, 3 configs ∼0.43, 4 configs ∼0.42	-
SMAC3 110	0.4744 (trial 108)	13h 14m 00s	6 configs ∼0.47, 5 configs ∼0.46, 7 configs ∼0.45	Yes (+0.0280)
Optuna TPE 50	0.4686 (trial 22)	6h 12m 14s	2 configs ∼0.46, 2 configs ∼0.45, 2 configs ∼0.44	-
Optuna TPE 100	**0.4910** (trial 45)	7h 52m 13s	3 configs ∼0.47, 2 configs ∼0.46, 4 configs ∼0.45	Yes (+0.0224)
HotpotQA Dataset				
Grid Search Local	0.6961	5h 04m 59s	-	-
Random 200	0.6242 (trial 110)	15h 08m 03s	1 config ∼0.58, 2 configs ∼0.57, 2 configs ∼0.56	-
SMAC3 50	0.5727 (trial 48)	3h 56m 44s	2 configs ∼0.54, 1 config ∼0.53, 1 config ∼0.52	-
SMAC3 100	**0.6606** (trial 92)	6h 17m 59s	1 config ∼0.64, 1 config ∼0.61, 1 config ∼0.60	Yes (+0.0879)
Optuna TPE 50	0.5944 (trial 28)	2h 10m 14s	1 config ∼0.57, 2 configs ∼0.56, 1 config ∼0.55	-
Optuna TPE 100	0.6057 (trial 12)	4h 51m 10s	1 config ∼0.58, 1 config ∼0.57, 1 config ∼0.56	Yes (+0.0113)

7.3.1. SciFact

Table 7.6 presents the results for varying sample sizes on the SciFact dataset. Both SMAC3 and Optuna TPE show improvement when doubling the sample size from 50 to 100 configurations. SMAC3 improved from 0.6313 to 0.6366, an increase of 0.0053 points. Optuna TPE showed similar gains, improving from 0.6265 to 0.6325, an increase of 0.0060 points.

The distribution of high-performing configurations reveals interesting patterns. For SMAC3-50, beyond the best score of 0.6313, the top configurations included 2 configs in the 0.62 range, 5 configs around 0.61, and 5 configs around 0.60. SMAC3-100 showed improvement with its best score of 0.6366, plus 2 additional configs in the 0.63 range, 5 configs around 0.62, and 6 configs around 0.61. This indicates that doubling the samples not only improved the best score but also increased the density of high-performing configurations.

For Optuna TPE, the 50-sample run achieved 0.6265 as its best score, with 2 configs each in the 0.61, 0.60, and 0.59 ranges. Optuna TPE-100 improved to 0.6325, with 1 additional config around 0.62, 3 configs around 0.60, and 4 configs around 0.59. While Optuna TPE-100 discovered higher-scoring configurations, it showed less consistency in the high-score regions compared to SMAC3.

The modest improvements from doubling the sample size (0.0053 for SMAC3, 0.0060 for Optuna TPE) suggest potential diminishing returns in the optimization process. These small gains indicate that either 50 samples may already be approaching the practical performance ceiling for these models and datasets, or that substantially more than 100 samples would be required to achieve meaningful further improvements. The fact that both optimizers show similar marginal gains despite different search strategies supports the hypothesis that the optimization may be converging toward an inherent performance limit of the pipeline configuration space.

Random search with 200 samples performed poorly despite exploring four times as many configurations as the 50-sample BO methods. Its best score of 0.6086 occurred late in the search (trial 158), and the top configurations clustered in lower score ranges (0.56-0.58). This demonstrates that intelligent exploration through Bayesian Optimization is more effective than brute-force sampling.

Regarding computational cost, doubling the sample size roughly doubles the runtime. SMAC3-100 required 7h 02m compared to 4h 25m for SMAC3-50, while Optuna TPE-100 needed 6h 30m versus 4h 02m for Optuna TPE-50. Both 100-sample configurations still completed faster than grid search (8h 36m) while achieving superior scores.

The timing of best configurations provides additional insights. SMAC3-50 found its best configuration at trial 48, near the end of its budget. SMAC3-100 found its optimum at trial 54, suggesting continued exploration value beyond 50 samples. Optuna TPE shows even stronger evidence for extended sampling, with its best configuration appearing at trial 98, indicating that the optimizer continued discovering improvements throughout the expanded search.

7.3.2. FIQA

The FIQA dataset presents a challenging optimization landscape characterized by sparse high-performing configurations. Grid search local optimization establishes a baseline score of 0.4682 in 8h 35m, serving as the benchmark for global optimization methods.

Random search with 200 configurations requires 23h 53m to achieve a maximum score of 0.4536, finding only seven configurations above 0.42. This inefficiency highlights the difficulty of the search space, where viable configurations represent a small fraction of the total space.

SMAC3 demonstrates interesting scaling behavior. With 50 configurations, it reaches 0.4464 at trial 20 in 9h 8m. When extended to 110 configurations[1], SMAC3 achieves 0.4744 at trial 108, exceeding the grid search baseline. The additional computational investment of 4 hours yields a meaningful improvement, with 18 configurations scoring above 0.45.

Optuna exhibits superior efficiency in navigating the sparse landscape. With only 50 configurations, Optuna matches the grid search baseline (0.4686) in 6h 12m. Scaling to 100 configurations, Optuna achieves the highest score of 0.4910 at trial 45 in 7h 52m, surpassing all other methods while maintaining computational efficiency.

The distribution of high-scoring configurations reveals the optimization dynamics. While random search struggles to consistently find configurations above 0.43, both SMAC3 and Optuna concentrate their search in productive regions, with SMAC3-110 identifying 18 configurations above 0.45, and Optuna-100 identifying 9 configurations above 0.45. This concentrated exploration pattern confirms that Bayesian optimization effectively learns and exploits the narrow band of viable configurations in FIQA's harsh landscape.

7.3.3. HotpotQA

HotpotQA presents a deceptive optimization landscape where Bayesian optimization methods struggle to consistently outperform grid search local optimization (0.6961). The dataset's multi-hop reasoning requirements create complex component interactions that challenge global optimization approaches.

Random search with 200 configurations achieves 0.6242 after 15 hours, finding only five configurations above 0.56. This sparse distribution of high-performing configurations confirms the challenging nature of the search space, yet differs fundamentally from FIQA's harsh landscape. Here, many configurations achieve acceptable scores, creating a plateau effect.

SMAC3 demonstrates interesting scaling behavior. With 50 configurations, it reaches 0.5727 in under 4 hours. Scaling to 100 configurations yields 0.6606 at trial 92, approaching the grid search baseline while exploring the full 50 million configuration space.

Optuna TPE shows rapid initial convergence, reaching 0.5944 at trial 28 in just 2h 10m with 50 configurations, the fastest result among all methods. The 100-configuration run finds a slightly better score of 0.6057, but notably at trial 12 rather than through extended exploration. This early discovery (trial 12 vs trial 28) suggests the improvement is not a

[1]SMAC3 occasionally explores more configurations than the specified budget of 100, running up to 110 configurations in this experiment. This behavior occurs due to SMAC's internal acquisition function, which may initiate additional evaluations when promising regions are identified.

result of the increased configuration budget but rather reflects the stochastic nature of the optimization process. As detailed in our experimental procedures, evaluation noise from GPU, language models and embedding APIs causes identical configurations to produce different latencies and scores across runs. This variability influences the surrogate model's representation of the search space, leading the optimizer to explore different regions and potentially discover high-performing configurations at different trials. The earlier discovery in the 100-configuration run thus appears to be a consequence of this stochastic exploration pattern rather than the benefit of additional sampling budget.

The configuration distribution patterns reveal the optimization dynamics. While SMAC3-100 identifies one exceptional configuration near 0.64, it finds few other high-scoring alternatives. This suggests the algorithm occasionally discovers promising regions but fails to systematically exploit them. The deceptive plateau of acceptable scores appears to satisfy the acquisition function, reducing exploration pressure toward truly optimal configurations.

Across all three datasets, our experiments reveal a consistent pattern: doubling the configuration budget from 50 to 100 samples yields diminishing returns with increased sampling. The magnitude of improvement varies by dataset characteristics. SciFact shows minimal gains (+0.005-0.006), FIQA exhibits moderate improvement (+0.022-0.028) due to its harsh landscape, while HotpotQA presents algorithm-specific responses with SMAC3 improving substantially (+0.088) but Optuna plateauing early (+0.011). This variation reflects how different optimization landscapes interact with the acquisition functions: harsh landscapes like FIQA benefit from extended exploration to find sparse viable regions, while deceptive plateaus like HotpotQA can trap optimizers in comfortable but suboptimal configurations.

The computational trade-offs further reinforce this diminishing returns pattern. Doubling the sample size consistently doubles runtime across all methods, yet yields only 2-8% score improvements. However, even with limited budgets, Bayesian Optimization demonstrates clear superiority over random search. Both SMAC3-50 and Optuna-50 outperform random-200 despite using 4x fewer samples, confirming that intelligent exploration through surrogate modeling and acquisition functions provides more value than additional random sampling.

These findings suggest that 50 configurations represent a practical sweet spot for global RAG optimization, providing most achievable performance at reasonable computational cost. Extending to 100 configurations may be justified only when the optimization landscape is known to be particularly challenging or when marginal improvements justify doubled runtime. The consistent pattern of diminishing returns indicates that further increases beyond 100 configurations would likely yield even smaller gains, establishing 50-100 samples as the effective range for cost-efficient Bayesian Optimization in RAG pipelines.

7.4. Outcome Robustness Across Datasets

Previous research questions established that Bayesian Optimization's performance varies significantly across datasets, but the underlying causes remain unclear. This analysis examines the Pareto fronts, which represent configurations that optimally balance accuracy and latency, to understand how dataset-specific characteristics influence optimization stability and

effectiveness. By visualizing the distribution and density of Pareto-optimal configurations, we can identify whether domain properties create smooth optimization landscapes conducive to BO or problematic patterns that hinder convergence.

The Pareto front visualizations identify optimal configurations using the principle of Pareto dominance in multi-objective optimization. A configuration appears as a red 'X' marker on the Pareto front when no other configuration simultaneously achieves both higher score and lower latency. Formally, a configuration belongs to the Pareto front if it is non-dominated, meaning that improving either its score or latency would necessarily worsen the other objective. For instance, a configuration with score 0.65 and latency 100 seconds dominates another with score 0.60 and latency 150 seconds since it performs better in both dimensions. However, a configuration with score 0.70 and latency 200 seconds represents a different valid trade-off and would also appear on the Pareto front. The scattered blue points represent all evaluated configurations during optimization, while the red markers highlight only those achieving optimal trade-offs. The density and distribution of these Pareto points reveal how effectively each optimization method navigates the performance-latency trade-off space and whether the dataset characteristics support finding diverse optimal solutions.

7.4.1. Scifact

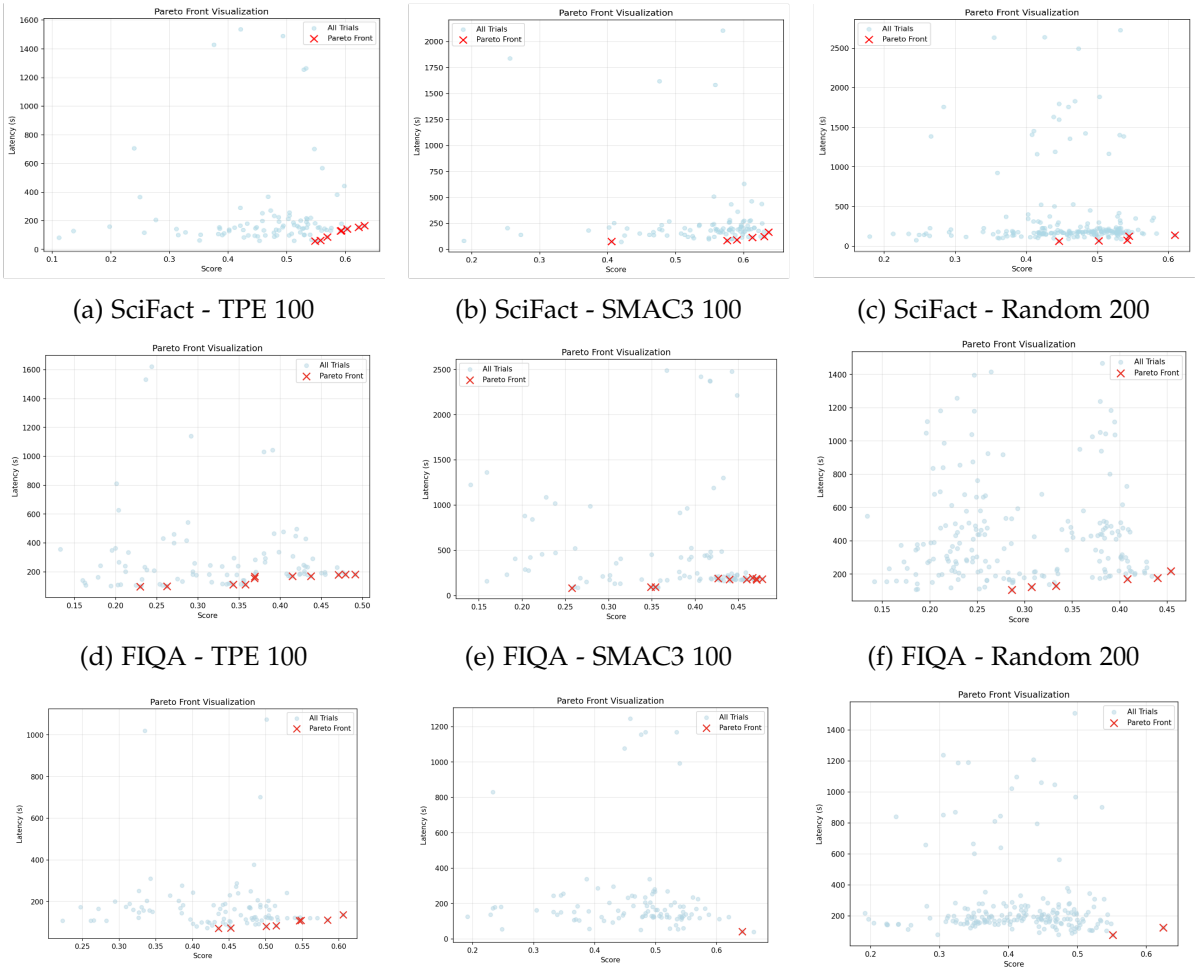
SciFact demonstrates ideal characteristics for Bayesian Optimization, with clear performance gradients that enable efficient navigation of the configuration space. As shown in Figures 7.3a-7.3c, the Pareto fronts reveal distinct optimization behaviors across methods.

TPE-100 and SMAC3-100 exhibit concentrated exploration patterns, with Pareto-optimal points clustering in the 0.55-0.65 score range while maintaining remarkably low latencies (under 200s). Both methods consistently identify high-performing regions. This concentrated pattern indicates that the surrogate models successfully learn the underlying performance landscape and exploit promising regions effectively.

For Random-200, despite exploring 200 configurations over 19h 48m, the results show limited diversity in latency performance. The sparse Pareto front confirms that high-performing configurations are rare in the search space, precisely the scenario where BO's intelligent exploration provides maximum value over random sampling.

The moderate score ceiling (0.55-0.65) combined with clear performance gradients creates an optimization landscape that enables several key advantages. Surrogate models can accurately predict nearby configuration performance due to the smooth landscape structure. The acquisition functions effectively balance exploration and exploitation because the performance signals provide clear guidance. Component interactions remain sufficiently decoupled to avoid performance cliffs that would confuse the optimizer. Additionally, early iterations quickly identify and converge on promising regions since the gradients point toward optimal areas.

This analysis confirms that Scifact's domain characteristics, specifically its structured performance gradients without excessive coupling between components, make it particularly amenable to Bayesian Optimization, resulting in both time efficiency (84% reduction in local optimization) and superior global optimization performance.



(a) SciFact - TPE 100 (b) SciFact - SMAC3 100 (c) SciFact - Random 200

(d) FIQA - TPE 100 (e) FIQA - SMAC3 100 (f) FIQA - Random 200

(g) HotpotQA - TPE 100 (h) HotpotQA - SMAC3 100 (i) HotpotQA - Random 200

Figure 7.3.: Pareto front visualizations across all datasets comparing TPE-100, SMAC3-100, and Random-200 optimization methods. Red crosses indicate Pareto-optimal configurations balancing score and latency. Rows represent different datasets (SciFact, FIQA, HotpotQA) while columns represent different optimization methods (TPE, SMAC3, Random).

7.4.2. FIQA

FIQA presents a challenging optimization landscape characterized by sparse viable configurations. Figures 7.3d-7.3f reveal how different optimization methods navigate this harsh terrain.

TPE-100 and SMAC3-100 demonstrate concentrated exploration patterns with Pareto points clustering between 0.45 and 0.47 scores while maintaining latencies below 200 seconds. This tight clustering indicates that Bayesian Optimization successfully identifies the narrow band of acceptable configurations and exploits this region efficiently. The concentration around this specific performance range suggests these methods learn to avoid the vast regions of poor configurations that dominate the search space.

Random-200 exhibits fundamentally different behavior, with configurations scattered across the entire performance spectrum from 0.15 to 0.45. While the explored configurations show latencies ranging from 200 to 1400 seconds, the few Pareto-optimal points discovered by random search cluster around 200 seconds latency. This sparse Pareto front, containing only a handful of points despite 200 trials, confirms that high-performing configurations with efficient latency are extremely rare in the search space. Random search struggles to consistently find these needle-in-a-haystack configurations, discovering them only occasionally through chance.

The sparse distribution of viable configurations, where most score below 0.30, creates a paradoxical advantage for Bayesian Optimization. Initial samples likely fail significantly, which forces the acquisition function to explore diverse regions rather than settling into local optima. Once the narrow band of viable configurations between 0.45 and 0.50 is discovered, both TPE and SMAC3 concentrate their search in this productive region. This harsh landscape provides clear signals about which regions to avoid, enabling more focused exploration than would occur in a landscape with gradual performance transitions.

The contrast between Bayesian Optimization and random search on FIQA demonstrates how domain characteristics fundamentally shape optimization effectiveness. The harsh landscape that makes random search inefficient becomes an advantage for intelligent exploration methods that can learn from failures and concentrate sampling in the sparse regions of acceptable performance.

7.4.3. HotpotQA

HotpotQA exhibits a deceptive optimization landscape that challenges Bayesian Optimization methods, as shown in Figures 7.3g-7.3i. Grid search local optimization achieves 0.6961, establishing a high baseline that global methods struggle to match.

TPE-100 produces Pareto points scattered across the 0.44 to 0.55 score range with latencies below 200 seconds. This wide distribution suggests that TPE explores various regions but fails to converge on the highest performing areas. The scattered pattern indicates the optimizer gets trapped in local optima, unable to systematically improve beyond the comfortable middle range of scores.

SMAC3-100 displays a unique pattern with one exceptional Pareto point near 0.64 while

other configurations cluster around 0.5. This isolated high performer represents a golden configuration that SMAC3 discovered but failed to exploit systematically. The inability to find similar high-scoring neighbors suggests either the configuration space has isolated peaks or the limited budget prevents thorough exploration of promising regions.

Random-200 identifies only two Pareto points despite 200 trials, with both appearing at different performance extremes. This extreme sparsity confirms that configurations optimally balancing score and latency are rare in HotpotQA's search space. The minimal Pareto front despite extensive sampling highlights the challenge of finding efficient high-performing configurations through random exploration.

The broad distribution of acceptable scores between 0.40 and 0.55 creates a deceptive plateau that misleads Bayesian Optimization. Most configurations achieve reasonable performance in this range, causing the surrogate model to perceive the current region as satisfactory. The acquisition function lacks strong signals to push exploration toward truly optimal regions since the plateau provides consistently moderate rewards. This comfortable middle ground reduces exploration pressure, causing both TPE and SMAC3 to settle for local improvements rather than seeking the global optimum near 0.70.

The multi-hop reasoning requirements of HotpotQA appear to create complex component interactions that global optimization struggles to capture. The fact that local componentwise optimization achieves 0.6961 while global methods reach only 0.64 suggests that sequential optimization better respects the pipeline's inherent dependencies. The deceptive plateau combined with strong component coupling makes HotpotQA particularly challenging for standard Bayesian Optimization approaches.

The analysis of Pareto fronts across three datasets reveals that domain characteristics fundamentally determine Bayesian Optimization success in RAG pipeline tuning. Each dataset presents distinct optimization challenges that interact differently with BO's core mechanisms.

Scifact represents the ideal case for Bayesian Optimization, with clear performance gradients and moderate component coupling. Both TPE and SMAC3 consistently identify and exploit the narrow band of high-performing configurations between 0.55 and 0.65, achieving concentrated Pareto fronts with low latencies. The structured landscape enables surrogate models to accurately predict configuration performance, resulting in efficient convergence and superior results compared to random search.

FIQA's harsh landscape, where most configurations score below 0.30, paradoxically aids Bayesian Optimization. The sparse distribution of viable configurations forces aggressive exploration after initial failures. Once the narrow band of acceptable configurations around 0.45 to 0.47 is discovered, both optimizers concentrate sampling in this productive region. The clear failure signals help BO avoid unproductive areas, making the harsh landscape advantageous for intelligent exploration.

HotpotQA demonstrates how deceptive optimization landscapes can undermine Bayesian Optimization. The plateau of acceptable scores between 0.40 and 0.55 satisfies the acquisition function prematurely, reducing exploration pressure toward the true optimum near 0.70. SMAC3 finds one exceptional configuration but cannot systematically exploit it, while TPE

remains trapped in the comfortable middle range. The multi-hop reasoning requirements create complex component interactions that global optimization fails to capture effectively.

These findings establish that Bayesian Optimization effectiveness depends primarily on landscape topology rather than dataset complexity or baseline performance. Domains with clear gradients or harsh failure modes provide strong signals for optimization, while deceptive plateaus with acceptable but suboptimal configurations mislead surrogate models. The success of BO in RAG pipeline tuning therefore requires careful consideration of domain-specific characteristics beyond simple performance metrics.

8. Discussion

8.1. Interpretation of Results

8.1.1. Which Bayesian Optimization algorithms are most effective for optimizing Retrieval-Augmented Generation pipelines in terms of achieving high quality scores and computational efficiency?

The experimental results provide clear evidence about which optimization algorithms are most effective for RAG pipeline optimization. SMAC3 with a Random Forest surrogate model and Optuna's Tree-structured Parzen Estimator (TPE) emerge as the most competitive approaches, each excelling in different aspects. SMAC3 consistently achieves the highest overall scores across datasets, while Optuna TPE provides comparable results with strong stability and moderate computational cost. Both methods substantially outperform other alternatives, including Gaussian Process models, HEBO's Heteroscedastic Gaussian Process, and RayTune implementations.

The performance advantage of SMAC3 stems from several key characteristics that align well with RAG pipeline optimization challenges. Random Forest surrogate models excel at handling the mixed discrete and continuous parameter spaces typical in RAG configurations. The algorithm demonstrates superior multi-objective optimization capabilities, effectively balancing the dual goals of maximizing evaluation metrics while minimizing execution time. Random Forests can efficiently model non-linear relationships between pipeline components without requiring complex kernel specifications, making them particularly suitable for the heterogeneous parameter types found in RAG systems. This modeling capability often enables SMAC3 to discover higher-scoring configurations, though the exploration process may require additional computational investment.

Optuna's Tree-structured Parzen Estimator represents an attractive alternative when optimization time is critical. TPE consistently maintains lower latency during optimization compared to other surrogate model approaches, making it valuable in scenarios where rapid iteration matters more than absolute maximum performance. The algorithm's probabilistic model separates good and bad configurations more simply than complex Gaussian Process approaches, enabling faster convergence in early optimization stages. While TPE may not always achieve the highest possible scores that SMAC3 can reach, its time efficiency and competitive performance levels make it particularly suitable for resource-constrained environments or when quick optimization cycles are required. The trade-off between SMAC3's potential for higher scores and TPE's faster optimization represents a key decision factor for practitioners selecting optimization strategies.

Multi-fidelity methods using Hyperband and Successive Halving produced unexpectedly

poor results despite their theoretical advantages. RAG pipelines require complete execution through all components to properly assess configuration quality, and early stopping based on incomplete results leads to premature rejection of potentially high-performing configurations. While multi-fidelity methods reduce optimization time significantly across all tested scenarios, the substantial performance degradation makes them unsuitable for RAG pipeline optimization where quality remains the primary objective.

HEBO's Heteroscedastic Gaussian Process approach achieved moderate scores but suffered from high computational overhead. The heteroscedastic modeling that accounts for varying noise levels across the search space theoretically provides advantages but unnecessary for RAG optimization. The additional complexity increased optimization time without delivering corresponding performance improvements over simpler approaches. This finding suggests that the noise characteristics in RAG pipeline evaluation remain relatively uniform across the configuration space, reducing the value of heteroscedastic modeling.

Random search baseline experiments provide important context for interpreting Bayesian optimization performance. Across tested datasets, Bayesian optimization methods consistently achieve substantial improvements over random search baselines while evaluating the same number of configurations. These gains become particularly significant in high-dimensional configuration spaces where random search often performs surprisingly well. The consistent performance advantage demonstrates that intelligent search strategies provide real value through their ability to learn from previous evaluations and focus exploration on promising regions. The efficiency advantage of Bayesian methods becomes more pronounced as the configuration space complexity increases, validating the investment in more sophisticated optimization approaches for production RAG systems.

8.1.2. Should Bayesian Optimization be applied globally across the entire RAG pipeline or locally at individual module levels to achieve optimal end-to-end generation quality?

The experimental comparison between global and local optimization strategies reveals complex trade-offs in RAG pipeline optimization. Local optimization consistently achieves competitive performance while requiring substantially less computational time than global approaches. The key insight is that local optimization provides an efficient path to near-optimal performance, achieving approximately 96 percent of grid search baseline scores while reducing optimization time by over 60 percent.

The advantage of local optimization stems from its decomposition of the optimization problem. With global configuration spaces exceeding 50 million possibilities, even 50 configurations represent negligible coverage. Local optimization allocates 20 configurations per component, providing much denser sampling of each module's parameter space. This focused exploration proves more effective for identifying high-performing configurations within practical time constraints.

Component independence analysis supports the effectiveness of local optimization. Each RAG component operates with distinct optimization objectives: retrievers maximize document relevance, rerankers score passage quality, filters apply threshold-based rules, compressors

reduce tokens based on importance scores, and generators produce coherent text from provided context. These natural boundaries create optimization subspaces that local methods exploit efficiently. The weak interaction effects discovered through global optimization indicate that optimal component configurations remain relatively stable regardless of other pipeline components' settings.

While global optimization occasionally achieves marginally higher absolute scores, the performance gains rarely justify the computational overhead. The diminishing returns become particularly problematic as pipeline complexity increases. Global optimization must evaluate complete pipeline configurations for every trial, accumulating computational costs across all components while exploring an exponentially larger space. This makes global optimization increasingly impractical for rapid iteration or resource-constrained environments.

Scenarios that might justify global optimization remain limited based on the experimental evidence. Pipelines with explicitly designed component interactions, shared parameters across modules, or cascading effects where early components fundamentally alter downstream behavior could benefit from joint optimization. However, standard RAG architectures demonstrate sufficient component independence that local optimization delivers superior efficiency. The practical implications strongly favor adopting local optimization for most RAG deployment scenarios, enabling incremental component improvements, targeted bottleneck resolution, and modular development practices that accelerate iteration cycles.

8.1.3. What is the relationship between the number of configurations explored through Bayesian Optimization and the final performance scores achieved in RAG pipeline tuning?

The experimental analysis of configuration budgets reveals distinct convergence patterns across different datasets, demonstrating how the number of configurations explored directly impacts final performance scores. Both SMAC and TPE with moderate trial budgets achieve superior score-latency trade-offs compared to random search with double the number of trials, establishing that intelligent sampling outweighs raw sample quantity.

Diminishing returns analysis across trial budgets reveals non-monotonic improvement patterns. Larger configuration budgets do not guarantee better maximum scores when evaluation noise is present. Stochastic evaluation from language models and varying GPU latencies cause optimization trajectories to diverge, where small differences in early evaluations shift the acquisition function's focus toward different search space regions. This non-deterministic behavior means identical initial configurations can lead to substantially different final outcomes.

The optimal configuration budget varies significantly by optimization objectives. Datasets with clear gradients typically reach convergence within moderate trial counts (50-100 samples), showing minimal improvement with extended budgets. Harsh landscapes benefit from extended exploration (100+ samples) to escape local optima, while deceptive plateaus may require alternative strategies beyond simply increasing trial counts. These might include forced exploration mechanisms or modified acquisition functions that explicitly seek diversity rather than exploitation.

Sample efficiency improvements from Bayesian optimization become most pronounced when considering the multi-objective nature of RAG optimization. The ability to find configurations that simultaneously optimize for high scores and low latency within limited trials represents a critical advantage over random search, which requires extensive sampling to accidentally discover balanced solutions. The Pareto front visualizations demonstrate that Bayesian methods learn to navigate the performance-latency trade-off space efficiently, gradually concentrating samples in regions offering optimal compromises between competing objectives.

The presence of evaluation noise from language model components introduces additional complexity to convergence analysis. Fixed random seeds produce identical initial configurations, but subsequent trials diverge as variations in scores and latencies accumulate. This non-determinism means convergence patterns should be interpreted as probabilistic trends rather than deterministic trajectories. Organizations should expect variation in optimization outcomes and consider running multiple optimization sessions with different seeds when the stakes justify the additional computational investment.

8.1.4. How do dataset domain characteristics influence the effectiveness and stability of Bayesian Optimization when tuning RAG pipeline configurations across different application contexts?

Dataset domain characteristics fundamentally determine how Bayesian optimization methods navigate the configuration space, with distinct patterns emerging across different application contexts. The optimization landscape topology—whether presenting clear gradients, harsh terrain, or deceptive plateaus—directly influences both the effectiveness and stability of the optimization process.

On datasets presenting ideal optimization landscapes with clear performance gradients (such as scientific fact verification), Bayesian methods concentrate their Pareto-optimal points in high-scoring regions with remarkably low latencies. This concentration indicates rapid convergence to high-performing regions within early trial stages. In contrast, random search finds significantly fewer Pareto-optimal configurations despite extensive runtime, with discovered points clustering at similar latency levels. The sparse Pareto fronts from random search confirm that high-performing configurations are rare but achievable, exactly the scenario where Bayesian optimization's intelligent search excels over random exploration.

Harsh optimization landscapes (characteristic of financial question-answering domains) force extensive exploration patterns. In these cases, Bayesian optimization Pareto points cluster tightly in moderate-scoring regions with controlled latencies, demonstrating the ability to find sweet spots that balance near-optimal scores with minimal computational cost. Random search exhibits widespread scatter across both performance dimensions. This pattern suggests random methods only achieve competitive scores by accidentally encountering computationally expensive configurations.

Deceptive optimization landscapes (found in multi-hop reasoning tasks) reveal an interesting failure mode for Bayesian methods. When datasets present broad distributions of acceptable scores creating performance plateaus, the surrogate model becomes satisfied

without pushing for further exploration. These scenarios show Bayesian optimization finding isolated exceptional configurations but generally remaining trapped in comfortable middle regions. This pattern contradicts the typical assumption that difficult landscapes always benefit from intelligent search, suggesting that deceptively acceptable plateaus can lead to premature convergence.

The stability of optimization outcomes varies dramatically across domain characteristics. Domains with clear gradients exhibit consistent convergence patterns across multiple runs, while harsh and deceptive landscapes show higher variance in final performance. This domain-dependent stability has practical implications: organizations working with well-structured domains can confidently rely on single optimization runs, while those dealing with complex reasoning tasks should budget for multiple optimization attempts to ensure robust configuration selection.

8.2. Limitations and Threats to Validity

8.2.1. Experimental Limitations

The experimental design faces several limitations that may affect the generalizability of findings. The study evaluated optimization strategies on three datasets representing different domains: scientific fact verification (Scifact), financial question answering (FIQA), and multi-hop reasoning (HotpotQA). While these datasets provide diversity in task complexity and domain characteristics, they remain relatively small-scale compared to production RAG deployments. The limited dataset size may not fully capture the optimization challenges present in large-scale enterprise systems processing millions of documents across heterogeneous domains.

A critical limitation involves the use of LLM generated ground truth for evaluation rather than human-written reference texts. The evaluation metrics rely on automated scoring from language models serving as judges, introducing potential biases. Human-written ground truth would provide more robust evaluation but was not feasible within the experimental constraints. This reliance on synthetic evaluation may overestimate performance improvements and miss quality dimensions that matter to human users.

Computational constraints imposed practical boundaries on the experimental scope. Each optimization run required substantial GPU resources, limiting the number of trials, datasets, and configuration variations that could be explored. The experiments used fixed trial budgets of 50 to 200 configurations, which may be insufficient for exploring the massive configuration spaces exceeding 50 million possibilities. Production deployments with larger computational budgets might discover superior configurations or reveal different convergence patterns. Additionally, the experiments focused on single GPU execution, not accounting for distributed computing scenarios where parallelization could fundamentally change optimization dynamics.

The choice of evaluation metrics presents another limitation. The experiments weighted retriever and generator metrics equally at 50 percent each, but optimal weighting likely

varies by application. Some use cases prioritize retrieval precision while others emphasize generation fluency. The fixed weighting scheme may bias results toward configurations that balance both objectives rather than excelling at task specific requirements. Furthermore, the metrics focus on accuracy and latency while neglecting other important dimensions such as cost per query, or robustness to adversarial inputs.

Temporal stability represents an unexplored dimension in the experimental design. The optimization assumes static data distributions and consistent model behavior over time. Production systems face concept drift, evolving document collections, and changing user query patterns. Configurations optimized for current conditions may degrade as these factors shift. The experiments did not evaluate how quickly optimized configurations become stale or whether Bayesian optimization can efficiently adapt to temporal changes without complete re-optimization.

The experimental setup used specific model architectures from both Hugging Face and state-of-the-art commercial LLMs, including GPT-3.5 Turbo, Claude 4 Sonnet, Gemini 2.0 Flash, and Mistral AI Large Instruct. While this selection covers a range of model capabilities and providers, it may not represent the full spectrum of available architectures. Recent advances in retrieval methods, including learned sparse representations and late interaction models, were not comprehensively evaluated. The generator components focused on standard language models without exploring retrieval-augmented training or specialized architectures explicitly designed for RAG systems. Despite testing commercial SOTA models through API calls, the experiments could not modify their internal configurations or fine-tune them for specific RAG tasks, potentially missing optimization opportunities available with fully controllable models. These architectural constraints may limit the applicability of findings to next generation RAG systems employing novel components or custom trained models optimized specifically for retrieval augmented generation.

8.2.2. Generalizability

The generalizability of findings to other RAG architectures remains partially uncertain. The experiments focused on a specific pipeline structure: query expansion, retrieval, reranking, filtering, compression, and generation. While this represents a common RAG architecture, many variations exist in practice. Systems using iterative retrieval, multy-stage reasoning, or hybrid dense-sparse retrieval may exhibit different optimization characteristics. The component independence assumption underlying successful local optimization may not hold for architectures with tighter integration between modules.

Architectural variations in production systems introduce additional complexity not captured in the experiments. Some RAG systems employ ensemble methods combining multiple retrievers or generators, creating intricate interaction patterns that global optimization might better capture. Others use adaptive components that modify their behavior based on query characteristics or retrieved content quality. These dynamic elements could fundamentally alter the optimization landscape, potentially favoring different optimization strategies than those identified in the static pipeline experiments.

Scalability to larger systems presents both opportunities and challenges for the optimization

approaches. Larger document collections may create more distinct performance regions in the configuration space, potentially benefiting Bayesian optimization's intelligent exploration. However, increased scale also amplifies the curse of dimensionality, where even intelligent sampling becomes insufficient for adequate coverage. The experiments with millions of possible configurations already showed coverage challenges; production systems with billions of documents and hundreds of tunable parameters may require fundamentally different optimization approaches.

The transferability of optimized configurations across domains and languages was not evaluated. Organizations often deploy RAG systems across multiple use cases, raising questions about whether optimization must be repeated for each domain or if configurations generalize. Cross-lingual applications introduce additional complexity, as optimal retrieval and generation strategies may vary significantly between languages. The monolingual, domain-specific optimization performed in the experiments may overstate the practical benefits if extensive re-optimization is required for each deployment context.

8.2.3. Future Work

The findings from this study open several promising avenues for future research in RAG pipeline optimization. These directions address current limitations while building on the demonstrated effectiveness of Bayesian optimization approaches.

Human-in-the-Loop Optimization

The reliance on LLM-generated ground truth and automated metrics represents a significant limitation that future work should address through human-in-the-loop optimization frameworks. Incorporating human feedback during the optimization process could capture quality dimensions missed by automated metrics, particularly for nuanced tasks requiring domain expertise. Active learning strategies could efficiently utilize limited human evaluation budgets by focusing human assessment on the most informative configuration comparisons. This approach could also enable preference learning, where the optimizer learns to balance multiple objectives according to stakeholder preferences rather than fixed metric weightings.

Scalability and Distributed Optimization

Addressing the computational constraints that limited experimental scope requires developing distributed Bayesian optimization methods tailored for RAG pipelines. Future work should investigate how to parallelize configuration evaluation across multiple GPUs while maintaining coherent surrogate model updates. Hierarchical optimization strategies could decompose the massive configuration spaces of production systems, using coarse-grained search to identify promising regions before fine-grained local optimization. Additionally, investigating early stopping criteria based on partial evaluation results could reduce the computational cost of exploring poor configurations.

Robustness and Safety Considerations

Production deployments require configurations that are not only high-performing but also robust and safe. Future research should extend optimization objectives to include robustness metrics, ensuring configurations perform well under distribution shift and adversarial inputs. Incorporating safety constraints into the optimization process could prevent selection of configurations that achieve high scores through undesirable behaviors like hallucination or information leakage. Multi-objective optimization frameworks that explicitly model the trade-offs between performance, latency, cost, robustness, and safety would better serve real-world deployment needs.

Architecture-Agnostic Optimization Frameworks

While this study focused on a specific pipeline architecture, production RAG systems employ diverse architectural patterns including iterative retrieval, multi-stage reasoning, and hybrid dense-sparse approaches. Future research should develop optimization methods that automatically adapt to different architectural patterns without manual reconfiguration. This includes investigating how to detect component interaction patterns to determine whether local or global optimization is more appropriate, and developing graph-based representations of RAG architectures that enable automated optimization strategy selection. Meta-learning approaches could leverage optimization experiences across different architectures to identify transferable configuration principles. Such architecture-agnostic frameworks would eliminate the need to develop custom optimization strategies for each RAG variant, enabling practitioners to apply Bayesian optimization effectively regardless of their specific architectural choices.

Adaptive Optimization for Temporal Dynamics

The static optimization used in this study assumes stable data and consistent model behavior. However, real RAG systems operate in changing environments. Future work should design online optimization methods that can detect and adjust to shifts in document content and query patterns. This includes exploring efficient triggers for re-optimization that balance system performance with computational cost, and creating incremental methods that update settings without full re-evaluation. Transfer learning can help retain useful optimization knowledge when data changes, using past results as informed priors instead of starting over. Such time-aware optimization would keep system settings effective throughout deployment, preventing performance decline as conditions change.

Cross-Modal and Multilingual Extensions

Modern RAG systems increasingly handle multiple modalities and languages, requiring optimization strategies that account for these complexities. Future work should investigate how configuration optimization transfers across languages, potentially identifying language-agnostic parameters versus those requiring language-specific tuning. For multimodal RAG

systems combining text, images, and structured data, optimization methods must balance performance across different input types while managing increased configuration complexity. Understanding how optimal configurations vary across modalities could inform the design of adaptive pipelines that adjust their behavior based on input characteristics.

These future directions collectively aim to transform RAG optimization from an empirical art to a principled science, enabling reliable and efficient deployment of high-performing retrieval-augmented generation systems across diverse applications and scales.

9. Conclusion

This research investigated Bayesian optimization strategies for automating RAG pipeline configuration, addressing the challenge of navigating complex parameter spaces with millions of possible combinations. The study compared seven different Bayesian optimization approaches across three diverse datasets to identify which algorithms work best for RAG optimization and where optimization effort should be applied within the pipeline architecture.

The experimental results show that SMAC3 with Random Forest surrogates and Optuna TPE are the most effective and reliable optimizers overall, though their relative advantages vary across datasets and optimization settings. SMAC3 often identifies higher scoring configurations, while Optuna TPE achieves comparable or better results in some cases with strong stability and efficient convergence. The observed differences suggest that optimizer performance is highly context dependent, influenced by dataset characteristics, evaluation metrics, and the underlying configuration space. Both approaches demonstrate variable improvements over random search baselines, with gains ranging from marginal to substantial depending on dataset characteristics and optimization landscape complexity. While some configurations achieve improvements exceeding 20 percent, others show more modest gains or occasionally underperform, particularly when using multi-fidelity methods.

The comparison between global and local optimization strategies revealed that local optimization delivers the best balance of performance and computational efficiency. Local optimization methods achieve 90 to 100 percent of grid search performance in most cases, with occasional instances exceeding grid search scores, while consistently reducing optimization time by 45 to 85 percent. This efficiency stems from the natural decomposition of RAG pipelines where components operate with distinct objectives and minimal interdependencies. Global optimization occasionally identifies marginally better configurations but incurs substantially higher computational cost, limiting its practicality for most deployment scenarios.

Analysis of sample efficiency and convergence patterns in our experiments demonstrated that 50 trials provided sufficient exploration for the tested configurations and datasets. The experimental results show that increasing trial budgets from 50 to 100 configurations yielded only marginal improvements in best scores discovered. This finding suggests that within our experimental setup, 50 configurations represented an effective balance point where most high-performing regions had been identified, and additional sampling exhibited diminishing returns. The Pareto front visualizations revealed that Bayesian methods effectively navigated multi-objective trade-offs between performance and latency within these modest trial budgets, consistently finding better balanced solutions than random search even with 200 trials. However, evaluation noise from language model components introduced non-deterministic behavior that could cause larger trial budgets to occasionally produce worse results than

smaller ones, further supporting the use of conservative configuration budgets rather than extensive exploration in our tested scenarios.

The research contributes practical guidelines for implementing Bayesian optimization in production RAG systems. Organizations should prioritize local optimization with SMAC3 or TPE depending on whether they value maximum performance or rapid iteration. Configuration budgets should be adjusted based on dataset characteristics and available computational resources. The weak component interactions observed suggest that current RAG architectures could benefit from modular optimization approaches that enable incremental improvements without full pipeline reconfiguration.

Several limitations affect the generalizability of findings. The reliance on LLM-generated ground truth and automated evaluation metrics may not fully capture quality dimensions important to human users. The experiments focused on specific pipeline architectures and model selections that may not represent all RAG variations in production. Computational constraints limited the exploration of very large configuration spaces and prevented investigation of temporal stability and cross-domain transferability.

Future work should address these limitations through several research directions. Incorporating human-in-the-loop evaluation frameworks would provide more robust validation of optimization effectiveness, capturing quality dimensions that automated metrics miss while enabling preference learning aligned with stakeholder needs. Developing scalable and distributed optimization methods would overcome computational constraints, allowing exploration of larger configuration spaces through parallelization and hierarchical search strategies. Investigating adaptive optimization strategies that handle temporal dynamics would address the unexplored dimension of configuration stability, ensuring optimized pipelines remain effective as document collections grow and query patterns evolve over time. Creating architecture-agnostic optimization frameworks would extend beyond the specific pipeline structure tested, enabling automatic adaptation to diverse RAG architectures including iterative retrieval and hybrid approaches. Extending optimization frameworks to handle cross-modal and multilingual RAG systems would address the growing complexity of production deployments that process diverse data types and languages. Finally, integrating robustness and safety considerations into the optimization process would ensure configurations not only achieve high performance but also maintain reliability under distribution shifts and prevent undesirable behaviors such as hallucination. These research directions would collectively advance RAG optimization from current empirical approaches toward a more principled and generalizable framework suitable for diverse production environments.

This research demonstrates that Bayesian optimization provides substantial value for RAG pipeline configuration, enabling organizations to achieve near-optimal performance with reasonable computational investment. The findings support adopting local optimization with intelligent search strategies as the pragmatic approach for most RAG deployments, balancing performance requirements with resource constraints while maintaining development agility.

A. LLM Evaluator Prompt for Compressor Component

The following prompt was used for the LLM-based evaluation of compressed contexts. The evaluator uses GPT-4 with temperature 0.0 to ensure consistent scoring across evaluations.

```
Question: {question}
Ground Truth Answer: {ground_truth}
Compressed Context: {context}

Score this compressed context (0.0-1.0) based on:

**1. ATOMIC FACT PRESERVATION (50%)**
- List all atomic facts in the ground truth (specific terms, numbers, methods,
    comparisons, measurements).
- Check if EACH fact exists in the compressed context.
- Missing fact: -0.15
- Missing critical fact (methods, measurements, specific comparator): -0.25
- Replaced with vague/general term: -0.20

**2. COMPLETENESS (15%)**
- Can someone write the EXACT ground truth answer using ONLY this context?
- If NO due to missing specifics: cap total score at 0.5

**3. RELEVANCE & ACCURACY (20%)**
- Irrelevant, off-topic, or wrong facts: -0.15 each
- If >25% of content is unrelated: cap at 0.5
- If unrelated content changes meaning or causes confusion: max 0.3

**4. EFFICIENCY & PRECISION (15%)**
- Brevity bonus (+0.1) if ALL atomic facts are preserved AND context is under 70
    words
- Excessive length with unrelated filler: -0.1 to -0.3 depending on severity

Return only a number between 0.0 and 1.0
```

B. Configuration Specifications

The complete configuration file defining the search space for the optimization experiments is provided below. This specification includes all model options, parameter ranges, and evaluation metrics used by the Bayesian Optimizer. Two configuration variants were employed: the primary configuration shown here uses SAP's AI Core infrastructure for commercial models, while a secondary configuration (not shown) substitutes open-source models deployed via VLLM, including Llama 2/3 variants, Qwen models, and TinyLlama for generation tasks, with corresponding open-source embedding models such as BGE and MPNet. The structure and parameter ranges remain identical across both configurations.

B.1. SAP API Configuration

```yaml
vectordb:
  - name: text-embedding-3-large
    db_type: chroma
    client_type: persistent
    embedding_model: <SAP_EMBEDDING_ENDPOINT_LARGE>
    path: ${PROJECT_DIR}/resources/chroma
  - name: text-embedding-3-small
    db_type: chroma
    client_type: persistent
    embedding_model: <SAP_EMBEDDING_ENDPOINT_SMALL>
    path: ${PROJECT_DIR}/resources/chroma
  - name: text-embedding-ada-002
    db_type: chroma
    client_type: persistent
    embedding_model: <SAP_EMBEDDING_ENDPOINT_ADA>
    path: ${PROJECT_DIR}/resources/chroma
  - name: gemini
    db_type: chroma
    client_type: persistent
    embedding_model: <SAP_GEMINI_EMBEDDING_ENDPOINT>
    path: ${PROJECT_DIR}/resources/chroma
```

Listing B.1: SAP API configuration - Embedding models

```yaml
node_lines:
  - node_line_name: pre_retrieve_node_line
    nodes:
      - node_type: query_expansion
        strategy:
          metrics: [retrieval_f1]
          speed_threshold: 10
          top_k: [2, 6]
          retrieval_modules:
            - module_type: bm25
              bm25_tokenizer: [porter_stemmer, space, gpt2]
            - module_type: vectordb
              vectordb: [text-embedding-3-large, text-embedding-3-small,
                         text-embedding-ada-002, gemini]
              embedding_batch: 256
        modules:
          - module_type: pass_query_expansion
          - module_type: hyde
            generator_module_type: sap_api
            llm: anthropic
            model: claude-4-sonnet
            max_token: [64, 128]
            api_url: <SAP_CLAUDE_ENDPOINT>
          - module_type: query_decompose
            generator_module_type: sap_api
            llm: anthropic
            model: claude-4-sonnet
            api_url: <SAP_CLAUDE_ENDPOINT>
          - module_type: multi_query_expansion
            generator_module_type: sap_api
            llm: anthropic
            model: claude-4-sonnet
            temperature: [0.0, 1.0]
            api_url: <SAP_CLAUDE_ENDPOINT>

  - node_line_name: retrieve_node_line
    nodes:
      - node_type: retrieval
        strategy:
          metrics: [retrieval_f1]
          speed_threshold: 10
        top_k: [2, 6]
        modules:
          - module_type: bm25
            bm25_tokenizer: [porter_stemmer, space, gpt2]
          - module_type: vectordb
            vectordb: [text-embedding-3-large, text-embedding-3-small,
                       text-embedding-ada-002, gemini]
            embedding_batch: 256
```

Listing B.2: SAP API configuration - Query expansion and retrieval

```yaml
- node_type: passage_reranker
  strategy:
    metrics: [retrieval_f1]
    speed_threshold: 10
  top_k: [1, 4]
  modules:
    - module_type: pass_reranker
    - module_type: monot5
      model_name:
        - castorini/monot5-base-msmarco-10k
        - castorini/monot5-large-msmarco-10k
        - unicamp-dl/ptt5-base-en-pt-msmarco-100k-v2
        - unicamp-dl/mt5-base-mmarco-v1
    - module_type: upr
    - module_type: colbert_reranker
    - module_type: sentence_transformer_reranker
      model_name:
        - cross-encoder/ms-marco-MiniLM-L12-v2
        - cross-encoder/ms-marco-TinyBERT-L2-v2
        - cross-encoder/stsb-distilroberta-base
    - module_type: flag_embedding_reranker
      model_name:
        - BAAI/bge-reranker-large
        - BAAI/bge-reranker-base
    - module_type: flag_embedding_llm_reranker
      model_name:
        - BAAI/bge-reranker-v2-m3
        - BAAI/bge-reranker-v2-gemma
    - module_type: flashrank_reranker
      model:
        - ms-marco-MiniLM-L-12-v2
        - ms-marco-MultiBERT-L-12
        - rank-T5-flan
    - module_type: sap_api
      model_name: cohere-rerank-v3.5
      api-url: <SAP_COHERE_RERANK_ENDPOINT>

- node_type: passage_filter
  strategy:
    metrics: [retrieval_f1]
    speed_threshold: 5
  modules:
    - module_type: pass_passage_filter
    - module_type: percentile_cutoff
      percentile: [0.4, 0.9]
    - module_type: similarity_threshold_cutoff
      threshold: [0.45, 0.95]
    - module_type: similarity_percentile_cutoff
      percentile: [0.4, 0.9]
```

Listing B.3: SAP API configuration - Reranking and filtering

```yaml
- node_type: passage_compressor
  strategy:
    metrics: [retrieval_token_f1, retrieval_token_recall,
              retrieval_token_precision]
    speed_threshold: 10
  modules:
    - module_type: pass_compressor
    - module_type: lexrank
      compression_ratio: [0.3, 0.7]
      threshold: [0.05, 0.3]
      damping: [0.75, 0.9]
      max_iterations: [15, 40]
    - module_type: spacy
      compression_ratio: [0.3, 0.5]
      spacy_model: ["en_core_web_sm", "en_core_web_md", "en_core_web_lg", "en_core_web_trf"]
    - module_type: tree_summarize
      generator_module_type: sap_api
      llm: anthropic
      model: claude-4-sonnet
      api_url: <SAP_CLAUDE_ENDPOINT>
    - module_type: refine
      generator_module_type: sap_api
      llm: anthropic
      model: claude-4-sonnet
      api_url: <SAP_CLAUDE_ENDPOINT>

de_line_name: post_retrieve_node_line
des:
- node_type: prompt_maker
  strategy:
    metrics:
      - metric_name: bleu
      - metric_name: meteor
      - metric_name: rouge
      - metric_name: sem_score
        embedding_model: openai
    speed_threshold: 10
  modules:
    - module_type: fstring
      prompt: [<Three prompt templates>]
    - module_type: long_context_reorder
      prompt: [<Three prompt templates>]
    - module_type: window_replacement
      prompt: [<Three prompt templates>]
```

Listing B.4: SAP API configuration - Compression and generation

```yaml
- node_type: generator
  strategy:
    metrics:
      - metric_name: bleu
      - metric_name: meteor
      - metric_name: rouge
      - metric_name: sem_score
        embedding_model: openai
    speed_threshold: 10
  modules:
    - module_type: sap_api
      llm: mistralai
      model: [mistralai-large-instruct]
      temperature: [0.0, 1.0]
      max_token: 512
      api_url: <SAP_MISTRAL_ENDPOINT>
    - module_type: sap_api
      llm: openai
      model: [gpt-3.5-turbo]
      temperature: [0.0, 1.0]
      max_token: 512
      api_url: <SAP_OPENAI_ENDPOINT>
    - module_type: sap_api
      llm: gemini
      model: Gemini-2.0-flash
      temperature: [0.0, 1.0]
      max_token: 512
      api_url: <SAP_GEMINI_ENDPOINT>
    - module_type: sap_api
      llm: anthropic
      model: claude-4-sonnet
      temperature: [0.0, 1.0]
      max_token: 512
      api_url: <SAP_CLAUDE_ENDPOINT>
```

Listing B.5: SAP API configuration - Generator models

Note: The open-source configuration follows an identical structure but substitutes VLLM-deployed models (Llama, Qwen, TinyLlama variants) for API calls and uses Hugging Face embedding models (BGE, MPNet) instead of commercial embeddings.

B.2. Open-Source Model Specifications

The following tables present the comprehensive list of open-source models utilized in our Hugging Face-based configuration. These models serve as alternatives to the commercial APIs described in the previous section, enabling fully self-hosted RAG pipelines without external dependencies.

Table B.1 lists the embedding models used for vector representation, ranging from lightweight models like BGE-Small to multilingual models such as BGE-M3. Table B.2 enumerates the

language models used for response generation, featuring various model sizes from 1.1B to 13B parameters to accommodate different computational constraints and performance requirements.

Table B.1.: Hugging Face Embedding Models

Model	**Checkpoint**
BGE Small	huggingface_baai_bge_small
RuBERT	huggingface_cointegrated_rubert_tiny2
MPNet	huggingface_all_mpnet_base_v2
BGE-M3	huggingface_bge_m3

Table B.2.: Hugging Face Generator Models

Model	**Checkpoint**
Llama 2 7B Chat	meta-llama/Llama-2-7b-chat-hf
Llama 3.2 1B Instruct	meta-llama/Llama-3.2-1B-Instruct
Phi-3 Mini 4K	microsoft/Phi-3-mini-4k-instruct
Qwen 3 4B	Qwen/Qwen3-4B
Qwen 2.5 1.5B Instruct	Qwen/Qwen2.5-1.5B-Instruct
Gemma 2B	google/gemma-2b
Gemma 3 1B IT	google/gemma-3-1b-it
Gemma 2 2B IT	google/gemma-2-2b-it
DeepSeek R1 Distill Qwen 1.5B	deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
Llama 2 7B Chat AWQ	TheBloke/Llama-2-7B-Chat-AWQ
Llama 2 13B Chat AWQ	TheBloke/Llama-2-13B-chat-AWQ
CodeLlama 7B Instruct AWQ	TheBloke/CodeLlama-7B-Instruct-AWQ
TinyLlama 1.1B Chat	TinyLlama/TinyLlama-1.1B-Chat-v1.0

C. Implementation Details

C.1. Hardware Specifications

The experiments were conducted on a high-performance computing cluster equipped with NVIDIA A100-SXM4-80GB GPUs, providing 80GB of GPU memory per device for model loading and inference. This substantial memory capacity proved essential for loading multiple transformer models simultaneously during optimization, particularly when evaluating configurations with large reranking models or processing extensive document collections.

For experiments using SAP API models, GPU acceleration is not strictly required for inference but benefits preprocessing steps including embedding generation and document processing.

C.2. Software Libraries and Versions

C.2.1. Bayesian Optimization Libraries

Library	Version	Primary Use
SMAC3	v2.3.1	Random forest surrogates, multi-fidelity BO
Optuna	v4.3.0	TPE and GP surrogates via BoTorch
Ray Tune	v2.44.1	BOHB implementation
HEBO	v0.3.6	Heteroscedastic GP optimization
HpBandster	v0.7.4	Hyperband-based multi-fidelity

Table C.1.: Bayesian Optimization libraries used in experiments

C.2.2. Core Framework and Dependencies

- **Python**: 3.12
- **AutoRAG**: v0.3.13 - Pipeline architecture, evaluation metrics, data schemas
- **ChromaDB**: v0.5.13 - Vector database for embedding storage
- **VLLM**: Latest stable - Local model serving
- **Weights & Biases**: v0.20.1 - Experiment tracking and visualization

C.2.3. Environment Configuration

All experiments were conducted in isolated Python virtual environments to ensure dependency consistency. The complete dependency tree, including transitive dependencies, is available in the project repository at `requirements.txt`.

Bibliography

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2020. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401.

[2] Q. R. Lauro, S. Shankar, S. Zeighami, and A. Parameswaran. *RAG Without the Lag: Interactive Debugging for Retrieval-Augmented Generation Pipelines*. 2025. arXiv: 2504.13587 [cs.HC]. URL: https://arxiv.org/abs/2504.13587.

[3] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon. *Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020*. 2021. arXiv: 2104.10201 [cs.LG]. URL: https://arxiv.org/abs/2104.10201.

[4] J. Kim, S. Kim, and S. Choi. *Learning to Warm-Start Bayesian Hyperparameter Optimization*. 2018. arXiv: 1710.06219 [stat.ML]. URL: https://arxiv.org/abs/1710.06219.

[5] J. Snoek, H. Larochelle, and R. P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*. 2012. arXiv: 1206.2944 [stat.ML]. URL: https://arxiv.org/abs/1206.2944.

[6] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. "Efficient and robust automated machine learning". In: *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 2755–2763.

[7] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. *Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms*. 2013. arXiv: 1208.3719 [cs.LG]. URL: https://arxiv.org/abs/1208.3719.

[8] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. "Survey of Hallucination in Natural Language Generation". In: *ACM Computing Surveys* 55.12 (Mar. 2023), pp. 1–38. ISSN: 1557-7341. DOI: 10.1145/3571730. URL: http://dx.doi.org/10.1145/3571730.

[9] S. Robertson and H. Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389. ISSN: 1554-0669. DOI: 10.1561/1500000019. URL: http://dx.doi.org/10.1561/1500000019.

[10] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. *Dense Passage Retrieval for Open-Domain Question Answering*. 2020. arXiv: 2004.04906 [cs.CL]. URL: https://arxiv.org/abs/2004.04906.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[12] N. Reimers and I. Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. arXiv: 1908.10084 [cs.CL]. URL: https://arxiv.org/abs/1908.10084.

[13] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave. *Unsupervised Dense Information Retrieval with Contrastive Learning*. 2022. arXiv: 2112.09118 [cs.IR]. URL: https://arxiv.org/abs/2112.09118.

[14] J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira. *Pyserini: An Easy-to-Use Python Toolkit to Support Replicable IR Research with Sparse and Dense Representations*. 2021. arXiv: 2102.10073 [cs.IR]. URL: https://arxiv.org/abs/2102.10073.

[15] C. Team. *Chroma: The AI-Native Open-Source Embedding Database*. https://github.com/chroma-core/chroma. Accessed: 2025-09-23. 2023.

[16] J. Johnson, M. Douze, and H. Jégou. "Billion-scale similarity search with GPUs". In: *arXiv preprint arXiv:1702.08734* (2017).

[17] H. Jégou, M. Douze, and C. Schmid. "Product Quantization for Nearest Neighbor Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (2011), pp. 117–128. DOI: 10.1109/TPAMI.2010.57.

[18] Y. A. Malkov and D. A. Yashunin. *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*. 2018. arXiv: 1603.09320 [cs.DS]. URL: https://arxiv.org/abs/1603.09320.

[19] Pinecone Systems Inc. *Pinecone: Managed Vector Database for Scalable Similarity Search*. https://docs.pinecone.io. Accessed: 2025-10-12. 2023.

[20] Weaviate.io. *Weaviate: Open Source Vector Database for AI Applications*. https://weaviate.io. Accessed: 2025-10-12. 2023.

[21] J. Liu. *LlamaIndex*. Nov. 2022. DOI: 10.5281/zenodo.1234. URL: https://github.com/jerryjliu/llama_index.

[22] L. Gao, X. Ma, J. Lin, and J. Callan. *Precise Zero-Shot Dense Retrieval without Relevance Labels*. 2022. arXiv: 2212.10496 [cs.IR]. URL: https://arxiv.org/abs/2212.10496.

[23] J. Pereira, R. Fidalgo, R. Lotufo, and R. Nogueira. *Visconde: Multi-document QA with GPT-3 and Neural Reranking*. 2022. arXiv: 2212.09656 [cs.CL]. URL: https://arxiv.org/abs/2212.09656.

[24] R. Nogueira and K. Cho. *Passage Re-ranking with BERT*. 2020. arXiv: 1901.04085 [cs.IR]. URL: https://arxiv.org/abs/1901.04085.

[25] R. Nogueira, Z. Jiang, and J. Lin. *Document Ranking with a Pretrained Sequence-to-Sequence Model*. 2020. arXiv: 2003.06713 [cs.IR]. URL: https://arxiv.org/abs/2003.06713.

[26] D. S. Sachan, M. Lewis, M. Joshi, A. Aghajanyan, W.-t. Yih, J. Pineau, and L. Zettlemoyer. *Improving Passage Retrieval with Zero-Shot Question Generation*. 2023. arXiv: 2204.07496 [cs.CL]. URL: https://arxiv.org/abs/2204.07496.

[27] O. Khattab and M. Zaharia. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*. 2020. arXiv: 2004.12832 [cs.IR]. URL: https://arxiv.org/abs/2004.12832.

[28] P. Damodaran. *FlashRank, Lightest and Fastest 2nd Stage Reranker for search pipelines.* Version 1.0.0. Dec. 2023. DOI: 10.5281/zenodo.10426927. URL: https://github.com/PrithivirajDamodaran/FlashRank.

[29] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu. *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. 2023. arXiv: 2309.07597 [cs.CL].

[30] F. Xu, W. Shi, and E. Choi. *RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective Augmentation*. 2023. arXiv: 2310.04408 [cs.CL]. URL: https://arxiv.org/abs/2310.04408.

[31] G. Erkan and D. R. Radev. "LexRank: Graph-based Lexical Centrality as Salience in Text Summarization". In: *Journal of Artificial Intelligence Research* 22 (Dec. 2004), pp. 457–479. ISSN: 1076-9757. DOI: 10.1613/jair.1523. URL: http://dx.doi.org/10.1613/jair.1523.

[32] *Metadata replacement — LlamaIndex API Reference*. https://developers.llamaindex.ai/python/framework-api-reference/postprocessor/metadata_replacement/. Accessed: 2025-10-12.

[33] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: 2307.03172 [cs.CL]. URL: https://arxiv.org/abs/2307.03172.

[34] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008. ISBN: 0521865719.

[35] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. 2023. arXiv: 2306.05685 [cs.CL]. URL: https://arxiv.org/abs/2306.05685.

[36] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu. *G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment*. 2023. arXiv: 2303.16634 [cs.CL]. URL: https://arxiv.org/abs/2303.16634.

[37] ExplodingGradients. *Ragas: Supercharge Your LLM Application Evaluations*. https://github.com/explodinggradients/ragas. 2024.

[38] J. Bergstra and Y. Bengio. "Random search for hyper-parameter optimization". In: 13.null (Feb. 2012), pp. 281–305. ISSN: 1532-4435.

[39] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. *Regularized Evolution for Image Classifier Architecture Search*. 2019. arXiv: 1802.01548 [cs.NE]. URL: https://arxiv.org/abs/1802.01548.

[40] D. Maclaurin, D. Duvenaud, and R. P. Adams. *Gradient-based Hyperparameter Optimization through Reversible Learning*. 2015. arXiv: 1502.03492 [stat.ML]. URL: https://arxiv.org/abs/1502.03492.

[41] K. Jamieson and A. Talwalkar. *Non-stochastic Best Arm Identification and Hyperparameter Optimization*. 2015. arXiv: 1502.07943 [cs.LG]. URL: https://arxiv.org/abs/1502.07943.

[42] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2018. arXiv: 1603.06560 [cs.LG]. URL: https://arxiv.org/abs/1603.06560.

[43] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[44] K. Deb. "Multiobjective Optimization Using Evolutionary Algorithms. Wiley, New York". In: Jan. 2001.

[45] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

[46] J. Snoek, H. Larochelle, and R. P. Adams. "Practical Bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[47] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization*. 2011. URL: https://api.semanticscholar.org/CorpusID:6944647.

[48] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. "Algorithms for hyper-parameter optimization". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., 2011, pp. 2546–2554. ISBN: 9781618395993.

[49] J. Mockus, V. Tiesis, and A. Zilinskas. "The application of Bayesian methods for seeking the extremum". In: *Towards Global Optimization*. Vol. 2. Elsevier, 1978, pp. 117–129.

[50] D. R. Jones, M. Schonlau, and W. J. Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4 (1998), pp. 455–492.

[51] H. J. Kushner. "A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise". In: *Journal of Basic Engineering* 86 (1964), pp. 97–106. URL: https://api.semanticscholar.org/CorpusID:62599010.

[52] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger. "Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting". In: *IEEE Transactions on Information Theory* 58.5 (May 2012), pp. 3250–3265. ISSN: 1557-9654. DOI: 10.1109/tit.2011.2182033. URL: http://dx.doi.org/10.1109/TIT.2011.2182033.

[53] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Poczos. *Multi-fidelity Bayesian Optimisation with Continuous Approximations*. 2017. arXiv: 1703.06240 [stat.ML]. URL: https://arxiv.org/abs/1703.06240.

[54] S. Falkner, A. Klein, and F. Hutter. *BOHB: Robust and Efficient Hyperparameter Optimization at Scale*. 2018. arXiv: 1807.01774 [cs.LG]. URL: https://arxiv.org/abs/1807.01774.

[55] J. Knowles. "ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems". In: *IEEE Transactions on Evolutionary Computation* 10.1 (2006), pp. 50–66. DOI: 10.1109/TEVC.2005.851274.

[56] J. Jin, Y. Zhu, X. Yang, C. Zhang, and Z. Dou. "FlashRAG: A Modular Toolkit for Efficient Retrieval-Augmented Generation Research". In: *Proceedings of the ACM Web Conference 2025*. 2025.

[57] A. Abdallah, B. Piryani, J. Mozafari, M. Ali, and A. Jatowt. *Rankify: A Comprehensive Python Toolkit for Retrieval, Re-Ranking, and Retrieval-Augmented Generation*. 2025. arXiv: 2502.02464 [cs.IR]. URL: https://arxiv.org/abs/2502.02464.

[58] Y. Chen, D. Guo, S. Mei, X. Li, H. Chen, Y. Li, Y. Wang, C. Tang, R. Wang, D. Wu, Y. Yan, Z. Liu, S. Yu, Z. Liu, and M. Sun. *UltraRAG: A Modular and Automated Toolkit for Adaptive Retrieval-Augmented Generation*. 2025. arXiv: 2504.08761 [cs.IR]. URL: https://arxiv.org/abs/2504.08761.

[59] D. Kim, B. Kim, D. Han, and M. Eibich. *AutoRAG: Automated Framework for optimization of Retrieval Augmented Generation Pipeline*. 2024. arXiv: 2410.20878 [cs.CL]. URL: https://arxiv.org/abs/2410.20878.

[60] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. "Towards Automatically-Tuned Neural Networks". In: *Proceedings of the Workshop on Automatic Machine Learning*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 64. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 24 Jun 2016, pp. 58–65.

[61] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: 1907.10902 [cs.LG]. URL: https://arxiv.org/abs/1907.10902.

[62] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. *Tune: A Research Platform for Distributed Model Selection and Training*. 2018. arXiv: 1807.05118 [cs.LG]. URL: https://arxiv.org/abs/1807.05118.

[63] A. I. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. R. Griffiths, A. M. Maraval, H. Jianye, J. Wang, J. Peters, and H. B. Ammar. *HEBO Pushing The Limits of Sample-Efficient Hyperparameter Optimisation*. 2022. arXiv: 2012.03826 [cs.LG]. URL: https://arxiv.org/abs/2012.03826.

[64] LlamaIndex. *Hyperparameter Optimization for RAG - LlamaIndex*. https://docs.llamaindex.ai/en/stable/examples/param_optimizer/. Accessed: 2025-09-23. 2024.

[65] J. Fu, X. Qin, F. Yang, L. Wang, J. Zhang, Q. Lin, Y. Chen, D. Zhang, S. Rajmohan, and Q. Zhang. *AutoRAG-HP: Automatic Online Hyper-Parameter Tuning for Retrieval-Augmented Generation*. 2024. arXiv: 2406.19251 [cs.CL]. URL: https://arxiv.org/abs/2406.19251.

[66] M. Barker, A. Bell, E. Thomas, J. Carr, T. Andrews, and U. Bhatt. *Faster, Cheaper, Better: Multi-Objective Hyperparameter Optimization for LLM and RAG Systems*. 2025. arXiv: 2502.18635 [cs.LG]. URL: https://arxiv.org/abs/2502.18635.

[67] A. Aravind. *RAGBuilder: Open Source Tool Kit for RAG Hyperparameter Tuning*. https://github.com/ragbuilder/ragbuilder. Accessed: 2025-09-23. 2024.

[68] A. Conway, D. Dey, S. Hackmann, M. Hausknecht, M. Schmidt, M. Steadman, and N. Volynets. *syftr: Pareto-Optimal Generative AI*. 2025. arXiv: 2505.20266 [cs.AI]. URL: https://arxiv.org/abs/2505.20266.

[69] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych. "BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021. URL: https://openreview.net/forum?id=wCu6T5xFjeJ.

[70] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhopf, R. Sass, and F. Hutter. *SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization*. 2022. arXiv: 2109.09831 [cs.LG]. URL: https://arxiv.org/abs/2109.09831.

[71] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. "Language Models are Unsupervised Multitask Learners". In: 2019. URL: https://api.semanticscholar.org/CorpusID:160025533.

[72] K. Järvelin and J. Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Trans. Inf. Syst.* 20.4 (Oct. 2002), pp. 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418. URL: https://doi.org/10.1145/582415.582418.

[73] E. Voorhees and D. Tice. "The TREC-8 Question Answering Track Evaluation". In: *Proceedings of the 8th Text Retrieval Conference* (Nov. 2000).

[74] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[75] C.-Y. Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/.

[76] S. Banerjee and A. Lavie. "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments". In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.

[77] A. Aynetdinov and A. Akbik. *SemScore: Automated Evaluation of Instruction-Tuned LLMs based on Semantic Textual Similarity*. 2024. arXiv: 2401.17072 [cs.CL]. URL: https://arxiv.org/abs/2401.17072.