JOIN (Jurnal Online Informatika)
p-ISSN: 2528-1682, e-ISSN: 2527-9165
Volume 11 Number 1 | June 2026: 243-255
DOI: 10.15575/join.v11i1.1733

Evaluating RAG Performance on Small Language Models for Low-Resource Devices through Chunking and Retrieval Methods

Amelia Dewi Agustiani[1], Salsabila Maharani Putri[2], Jonner Hutahaean[3], Muhammad Rizqi Sholahuddin[4], Muhammad Riza Alifi[5], Ade Hodijah[6]
[1, 2, 3, 4, 5, 6]Politeknik Negeri Bandung, Indonesia

Article Info

Article history:

Received July 4, 2025
Revised December 8, 2025
Accepted December 8, 2025
Published May 15, 2026

Keywords:

Chunking Technique
Low-Resource Device
Retrieval-Augmented Generation
Retrieval Approaches
Small Language Model

ABSTRACT

Retrieval-Augmented Generation (RAG) combines generative capabilities of language models with external document retrieval to answer questions grounded in reference texts. However, deploying RAG on low-resource devices like Android smartphones is challenging because SLMs have limited computational capacity and depend heavily on efficient chunking and retrieval. Although interest in on-device processing is growing, research on RAG configurations for SLMs under strict resource constraints especially for domain-specific tasks remains limited. This study therefore investigates which combinations of chunking technique, chunk size, overlap, and retrieval strategy best balance accuracy and speed on low-resource devices. The evaluation uses 148 Indonesian questions sourced from an official Hajj guidebook. The study consists of two phases retrieval and generation. Retrieval is evaluated using BLEU, ROUGE-L, MRR, MAP, and Hit@k, while answer quality is measured with BERTScore. The experiments compare different chunking methods (fixed-size or semantic), chunk sizes (128 or 256 tokens), overlaps (25, 50 and 100 tokens), and retrieval methods (dense, sparse, or hybrid). Results show that sparse retrieval with 256-token chunks and 100-token overlap yields the best answer quality (F1 = 0.726). However, 128-token chunks with the same overlap provide the fastest generation time (69.737 seconds). The main contribution of this study is a systematic evaluation of RAG configurations for fully on-device SLMs using a domain-specific Hajj and Umrah dataset not explored in prior research. The findings provide practical guidance for designing efficient and accurate RAG-based question-answering systems on low-resource devices.

Corresponding Author:

Jonner Hutahaean,
Informatics Engineering Department, Politeknik Negeri Bandung
Jl. Gegerkalong Hilir, Ds. Ciwaruga Kotak Pos 1234 Bandung, 40513
Email: jonnerh@jtk.polban.ac.id

1. INTRODUCTION

Retrieval-Augmented Generation (RAG) is a method that uses pretrained language models and external retrieval components to make answers that are based on documents. This strategy makes answers better without changing the model parameters, if there are relevant documents [1]. The chunking mechanism, which divides materials into smaller sections to provide more efficient retrieval and contextual grounding, is one of the major elements affecting RAG performance [2].

Variations in chunk size and chunk overlap have been shown to have a substantial impact on the relevancy of information supplied to the language model as well as the quality of retrieved content.

p-ISSN: 2528-1682
e-ISSN: 2527-9165

Also, the choice of retrieval approach has a big effect on how accurate the final response is, especially when answering complicated or analytical questions [3]. However, a lot of these studies use Large Language Models (LLMs), which are difficult to implement on limited devices like smartphones and demand a lot of processing power, compared to deep learning or transformers [4]. In order to overcome this constraint, SLMs have been created to facilitate effective language processing on devices with limited resources [5]. In on-device contexts, these models enable more private and real-time applications [6].

Due to their limited capacity, SLMs rely heavily on effective retrieval techniques and high-quality document inputs. Despite this, little study has been done on how chunking configurations and retrieval techniques interact to affect the precision and effectiveness of RAG systems based on SLMs [7]. A deeper understanding of this interaction is crucial for optimizing performance in low-resource environments.

The purpose of this study is to assess how retrieval and chunking tactics affect a RAG system's accuracy and response time using a small language model. The case study uses an Android device to administer a document-based question-answering job based on a Hajj advice paper. It is anticipated that the findings of this study will help the creation of effective and private document-based information access systems and offer insights into the best RAG-SLM settings for responsive, lightweight applications. This study offers a structured experimental comparison of chunking configurations and retrieval techniques for fully on-device RAG using an Indonesian Hajj dataset, which has not been assessed in previous studies. The results provide useful recommendations for enhancing performance when resources are limited.

2. METHOD

This section elaborates on the methodology adopted in this study, which focuses on the implementation and evaluation of a Retrieval-Augmented Generation (RAG) system using a methodological structure based on the RAG architecture that Lewis et al. [8] proposed in Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. It has two parts: a retriever and a generator. Figure 1 illustrates the overall system pipeline.

The design of this methodology is influenced by the modular framework introduced in AutoRAG: Automated Framework for Optimization of Retrieval-Augmented Generation Pipeline [9], which divides the RAG process into pre-retrieval, retrieval, and post-retrieval phases. AutoRAG offers a comprehensive automated optimization framework that assesses various RAG components in an end-to-end manner; conversely, our methodology involves the customization and manual evaluation of each phase, emphasizing the identification of the most efficient combination of chunking strategy, retrieval technique, and model configuration for resource-limited settings. Also, the experimental setup is designed to work well on mid-range Android smartphones with curated Hajj-related QA datasets. This puts our approach apart from AutoRAG, which aims for large-scale automated optimization.

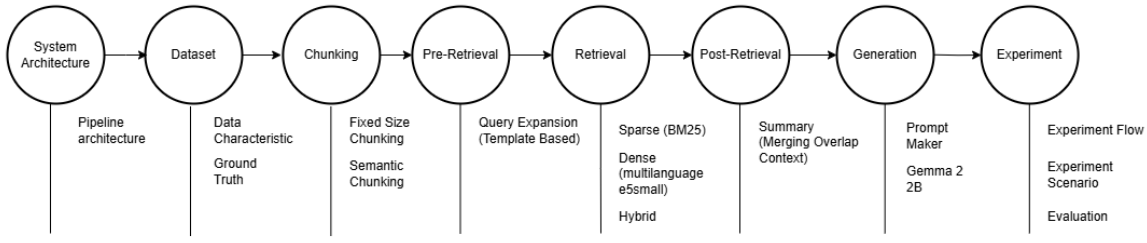

Figure 1. Methods

There are four main steps to this research. The first step is to define the problem and prepare the dataset. This means finding the right Hajj and Umrah documents, cleaning up the content, and making question-answer pairs that are specific to the domain. The second step is to design the system. This is where the RAG architecture is put into action by setting up the chunking process, building the

vector store, and connecting the retriever to the Small Language Model. The third phase is experimental evaluation, where different parameter settings are used to test different configurations and see how well they work at retrieving information and giving correct answers. The last step is analysis and validation, where the results are compared, analyzed, and evaluated based on accuracy, latency, and resource efficiency to find the best setup for mid-range devices.

2.1. System Architecture

The architecture of the proposed RAG system for Android is shown in Figure 2. During the development phase, guide documents are divided into smaller parts, embedded with the e5-multilingual-small model, and saved in an ObjectBox database on the device.

During production, user searches are expanded using templates and then processed using dense, sparse, or hybrid retrieval. After that, the results are merged. The retrieved pieces are then utilized to make a prompt, which is sent to a local language model to get an answer.

The implementation was made by combining and changing parts from two open-source projects: IRIS Android and Android Document QA ([10], [11]).

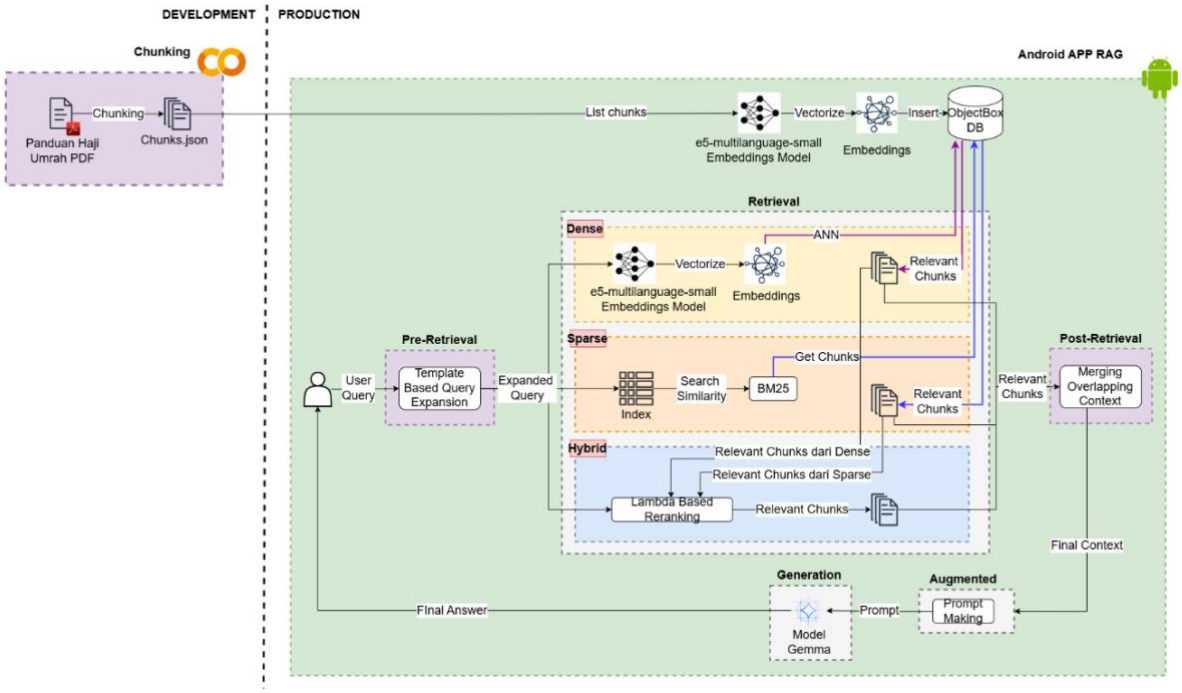

Figure 2. Architecture Pipeline

2.2. Dataset

2.2.1. Data Characteristics

The dataset utilized in this research is sourced from the *Tuntunan Manasik Haji dan Umrah* handbook, released by the Ministry of Religious Affairs of the Republic of Indonesia in 2024. The PDF file comprises 379 pages and is 6.10 MB in size. It gives complete information about the steps of Hajj and Umrah, including what you need to do, the most important rituals, and helpful tips. The information in this document is an outside source of knowledge that the system gets and combines with a Small Language Model (SLM) in a Retrieval-Augmented Generation (RAG) framework. This method does not fine-tune the model. Instead, it uses the information it gets from the document to answer queries about Hajj practices.

2.2.2. Ground Truth

Due to the lack of publicly accessible, domain-specific question-answer datasets on Hajj, this study developed an alternative method for constructing ground truth by employing a specialized Q&A

section, *"Tanya Jawab Manasik Haji dan Umrah,"* from the official guidebook issued by the Ministry of Religious Affairs of the Republic of Indonesia. The inquiries functioned as test inputs, while the associated responses acted as ground truth labels. To avert data loss and maintain experimental integrity, the Q&A part was omitted from the retrieval corpus. Manual validation of all 148 questions revealed that 78 had contextually or semantically corresponding answers found in other portions of the paper. The questions were subsequently categorized into four distinct types of reasoning. 125 factual questions, 6 analytical questions, 3 comparison questions, and 14 tutorial questions [3].

2.3. Chunking

Chunking is the practice of breaking a document into smaller parts to make it easier to find and to get around the limits on input length in language models. This technique can be used at many levels of detail, such as tokens, phrases, and semantic units [2]. This study concentrates on fixed-size chunking and semantic chunking approaches, despite the examination of many chunking strategies in prior research.

2.3.1. Fixed Size Chunking

With fixed-size chunking, text is split into parts that are of the same size, either by tokens or sentences, depending on what you need to do. This method is commonly used. However, it can provide chunks with too little or too much information, depending on how the document is structured and how hard the query is. To strike a compromise between contextual coverage and information relevancy, it is important to choose the right chunk size [12]. This study employs a Retrieval-Augmented Generation (RAG) framework utilizing a tiny language model on a low-resource Android device, incorporating chunk sizes of 128 and 256 tokens, each with overlaps of 25, 50, and 100 tokens [3]. Prior research has examined greater chunk sizes, including 512 tokens; however, they are excluded from this study due to latency and memory limitations characteristic of low-resource settings. The results show that there is a trade-off between chunk size and generation time, which shows how important it is to optimize for both accuracy and speed. This also makes me want to compare it to semantic chunking, which doesn't have defined token bounds.

2.3.2. Semantic Chunking

Semantic chunking is a way to break up a document into smaller parts depending on how similar the meanings of the phrases are to each other [13]. This method initiates by dividing the document into discrete sentences and thereafter producing embeddings for clusters of contiguous sentences. These embeddings facilitate the computation of semantic distances across groups, hence allowing for the identification of chunk borders in a contextually significant manner. This study employs semantic chunking in two configurations: with overlap and without overlap. In the overlapping variation, a single-sentence overlap is employed, whereby the final sentence of the preceding chunk is reiterated at the commencement of the following chunk to maintain contextual continuity.

2.4. Pre-Retrieval

In Retrieval-Augmented Generation (RAG) systems, pre-retrieval query expansion is essential for improving the relevance of recovered documents, particularly for concise, natural-language inquiries in areas such as Hajj and Umrah. The inquiry *"Apa yang dimaksud dengan ihram?"* can be simplified to merely *"ihram"* post-preprocessing, which diminishes contextual relevance for efficient retrieval.

Traditional synonym-based expansion (e.g., replacing ihram with suci or murni) often yields off-topic results. Therefore, this study adopts a template-based expansion strategy tailored to question types (e.g., factual, comparative), generating variants like *"ihram adalah"*, *"ihram berarti"*, or *"ihram merupakan".* This increases the likelihood of matching semantically relevant passages.

This approach is supported by Ma et al. [14], who emphasize the importance of adapting query reformulations to the type of question-factual, reasoning-based, or multi-hop, highlighting how different structures can significantly impact retrieval and generation performance in RAG systems. Similarly, Lin

et al. [15] demonstrate in their work on Multi-Stage Conversational Passage Retrieval that explicit and standalone question reformulations using neural query rewriting and term importance estimation improve retrieval quality. Although their method is model-driven, the principle aligns with template-based expansion: retaining the core meaning while reformulating the query into clearer and semantically rich alternatives.

2.5. Retrieval

2.5.1. Sparse

Sparse retrieval is a classic way to get information that uses algorithms like TF-IDF or BM25 to match terms exactly. This method uses word frequency to make both searches and documents into sparse vectors. It then finds similarities between the terms that are shared. The word "sparse" means that most of the term weights in the vectors are zero because each document only uses a small part of the vocabulary [1]. BM25 and other methods use exact term matching to achieve sparse retrieval. In these methods, queries and documents are shown as sparse term-frequency vectors. This study uses Apache Lucene to index preprocessed chunks (lowercased, tokenized, and stopword-filtered) that are kept in a local database. For performance, indexing employs an in-memory ByteBuffersDirectory, and Lucene's QueryParser parses requests. Using BM25 with the default settings (k1 = 1.2, b = 0.75), the ranking is done, and the results are given as a list of scored chunks organized by how relevant they are.

2.5.2. Dense

Dense retrieval is a neural-based retrieval technique that encodes questions and documents into dense vector embeddings utilizing transformer models like BERT. It lets semantic matching happen by comparing vectors with cosine similarity or the dot product [1]. This work employs a dense retrieval methodology utilizing the intfloat/e5-multilingual-small sentence embedding model [16], which accommodates several languages and is particularly effective for semantically intricate questions in Bahasa Indonesia. ONNX converts the model to int8 format so that it may be used quickly and with less capacity on mobile devices.

The model turns both queries and document fragments into 384-dimensional vector embeddings. These rich representations go beyond just matching keywords to capture semantic meaning, which makes them good for dealing with lexical variances and contextual complexities. Using Approximate Nearest Neighbor (ANN) search, the resulting embeddings are saved and compared. This makes it easy and quick to find the most relevant chunks based on vector similarity.

2.5.3. Hybrid

Hybrid retrieval uses both sparse and dense retrieval methods to make the most of their capabilities. BM25 and other sparse retrieval algorithms rely on finding exact term matches. On the other hand, dense retrieval strategies use embeddings to create semantic vector representations. Hybrid retrieval seeks to integrate both literal and contextual relevance, hence enhancing performance in information retrieval (IR) tasks [16].

This study employs a weighted score combination method to facilitate hybrid retrieval. As Mandikal and Mooney explain it, the final relevance score between a document D and a query Q is calculated as follows:

$$S_{hybrid}(D,Q) = \lambda \cdot Sim\big(z_{dense}(D), z_{dense}(Q)\big) + (1-\lambda) \cdot Sim(z_{sparse}(D), z_{sparse}(D)) \quad (1)$$

Where z_{dense} and z_{sparse} denote the dense and sparse embedding functions, respectively, and Sim is the cosine similarity measuring the angle between the corresponding vector embeddings. The hyperparameter $\lambda \in [0,1]$ controls the contribution of each component: higher values emphasize semantic (dense) similarity, while lower values prioritize lexical (sparse) matching.

However, they must be normalized before being combined because sparse and dense retrieval models frequently generate scores on various numerical scales. Without normalization, the final hybrid score may be biased toward the retrieval method with inherently larger or narrower score distributions.

To address this, we adopt Min-Max Normalization, as also applied in prior hybrid retrieval studies [9], where dense and sparse scores are normalized separately before being fused. These studies

p-ISSN: 2528-1682
e-ISSN: 2527-9165

highlight that without normalization, one modality (e.g., dense or sparse) may dominate the combined sscore due to differing value ranges This technique makes sure that each retrieval signal adds up to the final hybrid relevance score by scaling both score distributions into the same range.

The Min-Max Normalization technique transforms each score v_i into a normalized value v_{norm_i} using the formula provided in [17].

$$v_{norm_i} = (\frac{v_i - v_{min}}{v_{max} - v_{min}}) \quad (2)$$

v_{norm_i} represents the normalized score at index i, calculated from the original score v_i. The normalization adjusts the value by subtracting the minimum score v_{norm_i} in the dataset and dividing it by the difference between the maximum and minimum scores $v_{max} - v_{min}$. This puts all scores into a consistent [0, 1] range, which makes it possible to compare and combine scores from different retrieval methods.

2.6. Post-Retrieval

After we get the relevant pieces, we do a post-retrieval merging procedure to make them more coherent and cut down on redundancy. To do this, you need to find sequences of at least five words that are the same between the end of one chunk and the start of another. To merge chunks with the longest overlaps, you keep deleting duplicate words until you can't merge any more. This method helps make passages shorter and easier to read, which is especially helpful when working on devices with limited space and compact models like Gemma 2B, where input length and processing speed are quite important.

2.7. Generation

The generation stage in the RAG pipeline is responsible for producing final answers based on the retrieved chunks. This study employs Gemma-2B-it, an instruction-tuned language model, deployed locally on Android devices via the GGUF format using llama.cpp. This section addresses prompt engineering techniques, local deployment efficiency, and the reasoning behind model selection.

2.7.1. Model Generation

The Gemma-2B-it model was chosen for its optimal performance in response quality and computational efficiency, particularly in resource-limited settings. In comparison to alternatives like LLaMA 3.2-1B-Instruct and Qwen 1.5B, Gemma delivered more comprehensive and cohesive results with minimum redundancy. Despite exhibiting a minor delay during the initial phase of token creation, the model reliably produces contextually relevant outputs. The model is quantized to int8 GGUF format, facilitating efficient execution on local devices while preserving adequate semantic comprehension for document-based answer generation.

2.7.2. Prompt Maker

In the local RAG system, prompting plays an essential role in directing the language model to produce relevant, precise, and context-grounded responses. The prompt is constructed by integrating task instructions, retrieved document segments, the user's query, and a generation signal for the model. The structure follows the format suggested by Bartowski [18],using special tokens such as <bos><start_of_turn>user and <start_of_turn>model to indicate dialogue turns.

This approach is further inspired by the structured prompting methodology introduced by Park et al. [19], who emulate Retrieval-Augmented Generation (RAG) pipelines through prompt engineering. Similar to their template, which consists of an instruction header, a === CONTEXT === delimiter, a question section, and an output segment, the proposed format provides explicit guidance for the model. As shown in Table 1, the retrieved document chunks are inserted under the === Konteks === section,

followed by the user's question. The model is then prompted to generate an answer within the <start_of_turn>model block and terminate the response using the </s> token.

The prompt template is written in Bahasa Indonesia because the target users and document sources in this study are Indonesian. Therefore, the use of formal, instruction-oriented Indonesian language is intentional to ensure that the generated responses align with the linguistic context of the application and the model's instruction-following behaviour. This configuration promotes consistent generative behaviour, ensures that the output remains grounded in the retrieved evidence, and supports clear, concise responses suitable for on-device inference.

Table 1. Prompting Format

Prompting Format
<bos><start_of_turn>user Tugas Anda: - Pilih bagian konteks yang relevan dengan pertanyaan. - Gunakan kalimat dari konteks secara langsung tanpa mengubah makna. - Jelaskan secara wajar agar jawaban mudah dipahami, tetap berdasarkan konteks. - Jangan menambahkan informasi atau opini di luar konteks. - Jawaban harus jelas, relevan, dan cukup untuk menjawab pertanyaan. - Gunakan kalimat yang rapi, perhatikan spasi, tanda baca, dan baris baru jika perlu. - Hindari pengulangan yang tidak perlu. - Tidak perlu membuat kesimpulan atau rekomendasi kecuali ada di konteks. - Jawaban harus diakhiri dengan token </s> tanpa tambahan karakter lain. === Konteks === <retrieve context> ========== Pertanyaan: <user query> Jawaban: <start_of_turn>model

2.8. Experiment

2.8.1. Experiment Flow

Figure 3 shows the order in which each experiment is done. The document is processed once to create all of the chunking settings, such as fixed-size and semantic. A certain configuration is chosen for each case, and then a retrieval method (dense, sparse, or hybrid) is chosen. The system subsequently executes question answering using the test dataset, extracting pertinent segments and transmitting them to the small language model (SLM) for response generation. Results are gathered and assessed according to precision and response duration. This process guarantees uniform and replicable performance in all experimental conditions.

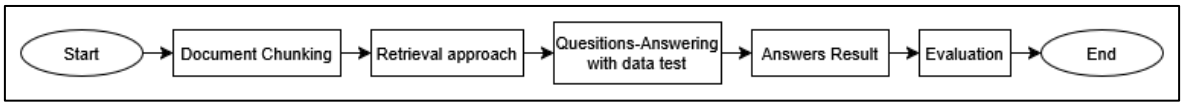

Figure 3. Experiment Execution Flow

2.8.2. Experiment Scenario

This study assesses 24 experimental scenarios, each created by integrating various retrieval methodologies and chunking settings. The retrieval methodologies encompass dense, sparse, and hybrid approaches. Two strategies for chunking are employed: fixed-size and semantic. Fixed-size chunking is evaluated using chunk lengths of 128 and 256 tokens, each associated with overlaps of 25, 50, and 100 tokens. In these arrangements, the Top-K value for retrieval is established at 3, enabling the system to select the three most pertinent chunks for transmission to the generator.

In semantic chunking, the chunk units are generally extended (according to sentence borders), and tests are performed with and without one-sentence overlap. The Top-K number for semantic chunking is established at 2 to ensure computational efficiency and context size balance. This modification accounts for the increased average size of semantically segmented segments, ensuring that

the overall input stays within permissible limitations for on-device production while still offering enough rich context.

2.8.3. Evaluation

To assess the performance of the proposed Retrieval-Augmented Generation (RAG) system, we utilize metrics at both the retrieval and generation levels. Retrieval metrics evaluate the system's efficacy in identifying pertinent segments, whereas generation measurements gauge the quality of the ultimate response.

To assess the efficacy of the retrieval module inside the Retrieval-Augmented Generation (RAG) architecture, we employ a blend of conventional information retrieval metrics and content-level similarity assessments. The selected metrics are derived from the previous research conducted by Yu et al. [20] and Setty et al. [12], and are utilized to assess both ranking accuracy and semantic congruence with the anticipated responses. The assessment includes the subsequent essential metrics:

MRR quantifies the position of the first pertinent document in the ordered retrieval outcomes. The calculation is based on the mean of the reciprocal rank of the initial pertinent result for each query:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \quad (3)$$

$|Q|$ represents the total number of questions, and $rank_i$ denotes the position of the first pertinent document for query iii. An elevated MRR signifies that pertinent results are obtained more promptly.

MAP quantifies the mean precision across all pertinent documents and queries. It signifies both pertinence and hierarchical arrangement:

$$MAP = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{\sum_{k=1}^{n} (P(k) \times rel(k))}{|relevant\ documents_q|} \quad (4)$$

$P(k)$ denotes the accuracy at rank k, $rel(k)$ signifies relevance, and $|relevant\ documents_q|$ represents the total count of relevant documents for query q. A high MAP score signifies continuously precise and well-ordered retrieval.

Hit@K assesses the proportion of searches for which at least one pertinent document is included within the top-K results. It is calculated as:

$$HIT@K = \frac{1}{N} \sum_{i=1}^{N} hit_i \quad (5)$$

$hit_i = 1$ if a relevant document is discovered in the top-K results for query i, and 0 otherwise. A high Hit@K means that the top-ranked retrieval is likely to be accurate.

BLEU evaluates the correspondence of n-grams between the retrieved text and the reference response. It is very adept at determining if the retrieved output includes the key phrases included in the anticipated response. The BLEU score is computed as follows:

$$BLEU = BP \cdot \exp(\sum_{n=1}^{N} w_n \log p_n) \quad (6)$$

In this context, p_n signifies the adjusted precision for n-grams of length n, w_n represents the weight assigned to each n-gram, and BP is the shortness penalty, which imposes a penalty on excessively brief outputs. A superior BLEU score signifies enhanced surface-level resemblance between the retrieved segment and the reference.

ROUGE-L assesses structural and content similarity by calculating the length of the Longest Common Subsequence (LCS) between the retrieved and reference texts. The ROUGE-L score is defined as:

$$ROUGE - L_{recall} = \frac{LCS\ (candidate,reference)}{\#words\ in\ reference} \quad (6)$$

The accuracy component, conversely, quantifies the relevance of the retrieved content in relation to the reference and is computed as:

$$ROUGE - L_{precission} = \frac{LCS\ (candidate,reference)}{\#words\ in\ reference} \quad (6)$$

The F1-Score offers a balanced assessment by harmoniously integrating recall and precision, resulting in the final ROUGE-L score:

$$ROUGE - L_{F1-Score} = 2\ x \frac{recall \cdot precission}{recall+precission} \quad (6)$$

In these formulas, *LCS(candidate,reference)* is the number of words in the longest common subsequence that the candidate and reference texts have in common. The F1 variation of ROUGE-L is especially significant for assessing retrieval-augmented generation (RAG) systems, because recovered text segments should strongly correspond with reference answers in terms of both content and coherence. ROUGE-L functions as a valuable metric for assessing the structural and semantic informativeness of retrieved segments, particularly in intensive retrieval contexts.

3. RESULT AND DISCUSSION

This section evaluates 24 experimental scenarios by examining the effects of retrieval methods, chunking tactics, chunk sizes, and overlaps on the overall performance of the question-answering system. The investigation examines the impact of each configuration on answer quality and processing efficiency in an end-to-end context.

Table 2. Experiment Result

Retrieval Approach	Chunking Configuration	Retrieval						Generation	
		ROUGE-L	BLEU	MRR	MAP	HIT@K	Retrieve Time (ms)	BERT-Score	Generate Time (s)
Dense	Fixed, 128 tokens, 25 overlaps	0,243	0,138	0,415	0,291	0,513	2.044,716	0,708	78,232
	Fixed, 128 tokens, 50 overlaps	0,246	0,144	0,380	0,243	0,538	213,791	0,712	84,967
	Fixed, 128 tokens, 100 overlaps	0,279	0,152	0,402	0,134	0,487	296,419	0,714	81,338
	Fixed, 256 tokens, 25 overlaps	0,173	0,099	0,449	0,342	0,577	2.187,980	0,701	140,649
	Fixed, 256 tokens, 50 overlaps	0,176	0,100	0,453	0,337	0,551	364,642	0,706	153,800
	Fixed, 256 tokens, 100 overlaps	0,173	0,100	0,415	0,278	0,564	409,108	0,714	148,328
	Semantic, No Overlap	0,148	0,085	0,481	0,387	0,551	639,162	0,704	170,564
	Semantic, With Overlap	0,165	0,094	0,551	0,456	0,615	2.291,574	0,708	170,953
Sparse	Fixed, 128 tokens, 25 overlaps	0,265	0,152	0,468	0,350	0,577	2.058,419	0,709	84,160
	Fixed, 128 tokens, 50 overlaps	0,264	0,154	0,479	0,310	0,564	200,554	0,713	85,055
	Fixed, 128 tokens, 100 overlaps	0,312	0,179	0,415	0,180	0,487	**185,514**	0,714	**69,737**
	Fixed, 256 tokens, 25 overlaps	0,196	0,116	0,521	0,414	0,667	2.051,378	0,714	136,547
	Fixed, 256 tokens, 50 overlaps	0,202	0,122	0,573	0,454	0,705	386,500	0,720	134,446
	Fixed, 256 tokens, 100 overlaps	0,222	0,133	0,564	0,385	0,692	377,561	**0,726**	144,054
	Semantic, No Overlap	0,168	0,094	0,590	0,511	0,679	624,230	0,717	176,322
	Semantic, With Overlap	0,180	0,100	0,583	0,506	0,667	2.137,507	0,715	171,531

p-ISSN: 2528-1682
e-ISSN: 2527-9165

Retrieval Approach	Chunking Configuration	Retrieval						Generation	
		ROUGE-L	BLEU	MRR	MAP	HIT@K	Retrieve Time (ms)	BERT-Score	Generate Time (s)
Hybrid	Fixed, 128 tokens, 25 overlaps	0,275	0,162	0,487	0,355	0,577	1.498,068	0,712	88,546
	Fixed, 128 tokens, 50 overlaps	0,288	0,179	0,472	0,321	0,577	1.256,736	0,708	87,792
	Fixed, 128 tokens, 100 overlap	**0,324**	**0,184**	0,391	0,171	0,474	1.007,642	0,718	72,049
	Fixed, 256 tokens, 25 overlaps	0,194	0,116	0,553	0,442	0,692	1.521,216	0,714	142,381
	Fixed, 256 tokens, 50 overlaps	0,204	0,122	0,596	0,464	0,679	1.599,946	0,715	180,995
	Fixed, 256 tokens, 100 overlaps	0,218	0,135	0,562	0,391	**0,705**	1.056,682	0,724	150,489
	Semantic, No Overlap	0,178	0,104	0,615	0,522	**0,705**	1.507,392	0,717	168,633
	Semantic, With Overlap	0,192	0,112	**0,622**	**0,541**	0,692	3.295,095	0,718	184,114

3.1. Performance Overview

This study assessed 24 system configurations using a dataset of 148 questions to investigate the effects of retrieval approaches and chunking strategies on system performance. As shown in Table 2, the evaluated configurations combine three retrieval approaches, namely dense, sparse, and hybrid retrieval, with several chunking settings, including fixed-size chunks of 128 and 256 tokens as well as semantic chunking with and without overlap. The table reports both answer-quality metrics, including ROUGE-L, BLEU, and BERTScore-F1, and retrieval-quality metrics, including MRR, MAP, HIT@K, and retrieval time.

The results show that retrieval and chunking configurations affect both response quality and computational efficiency. BERTScore-F1 ranged from 0.701 to 0.726, while generation time ranged from approximately 70 to more than 180 seconds. Retrieval time varied from around 185 ms to more than 3,000 ms. However, compared with generation time, retrieval latency contributed relatively little to the overall processing time; therefore, the analysis places greater emphasis on answer quality and generation efficiency.

Overall, sparse and hybrid retrieval methods generally outperformed dense retrieval. The highest BERTScore-F1 was achieved by sparse retrieval with fixed 256-token chunks and a 100-token overlap, obtaining a score of 0.726. This indicates that the configuration produced the strongest semantic similarity between the generated answers and the reference answers. Meanwhile, the most computationally efficient generation setting was sparse retrieval with fixed 128-token chunks and a 100-token overlap, which achieved the shortest generation time of 69.737 seconds while maintaining a competitive BERTScore-F1 of 0.714. This suggests that smaller chunks can reduce generation latency with only a modest reduction in semantic answer quality.

The table also shows that the strongest retrieval-ranking performance was obtained by hybrid retrieval with semantic chunking and overlap, which achieved the highest MRR of 0.622 and MAP of 0.541. However, this configuration required the longest retrieval time, reaching 3,295.095 ms, and its generation time was also relatively high at 184.114 seconds. Therefore, although semantic chunking with hybrid retrieval improves ranking quality, it may not be the most efficient option for practical deployment in a local RAG system.

These findings indicate that no single configuration dominates across all evaluation metrics. Larger chunks and semantic chunking tend to improve retrieval coverage and ranking quality, but they increase generation time. In contrast, smaller chunks, particularly 128-token chunks with high overlap, offer better computational efficiency but may risk incomplete contextual coverage when the answer requires information distributed across longer document segments. Therefore, configuration tuning is

essential to balance answer quality, retrieval effectiveness, and processing efficiency in end-to-end question-answering performance.

3.2. Impact of Retrieval Approach

The hybrid retrieval method exhibited superior ranking efficacy, but sparse retrieval provided same answer quality with significantly enhanced efficiency. Conversely, dense retrieval repeatedly exhibited the worse performance across parameters. The results, as outlined in Table 3, indicate that while hybrid retrieval demonstrates superior accuracy, sparse retrieval achieves a more favorable equilibrium between performance and computational expense, rendering it more appropriate for practical implementation.

Table 3. Experiment Result in Retrieval Approach

Chunking Configuration	Retrieval						Generation	
	ROUGE-L	BLEU	MRR	MAP	HIT@K	Retrieve Time (ms)	BERT-Score	Generate Time (s)
Dense	0,201	0,114	0,443	0,309	0,550	128,604	0,708	1.055,924
Sparse	0,226	0,131	0,524	0,389	0,630	**125,231**	**0,716**	**1.002,708**
Hybrid	**0,234**	**0,139**	**0,537**	**0,401**	**0,638**	134,375	**0,716**	1.592,847

3.3. Impact of Chunking Technique

Table 4 shows that semantic chunking produced superior retrieval ranking performance, although fixed-size chunking excelled in surface-level similarity and processing speed. Although it attained comparable generation quality, fixed-size chunking demonstrated greater efficiency, rendering it advantageous for situations with constrained computational resources.

Table 4. Experiment Result in Chunking Technique

Chunking Configuration	Retrieval						Generation	
	ROUGE-L	BLEU	MRR	MAP	HIT@K	Retrieve Time (ms)	BERT-Score	Generate Time (s)
Fixed-Size Chunking	**0,236**	**0,138**	0,477	0,326	0,590	**114,642**	**0,713**	**1.039,826**
Semantic Chunking	0,172	0,098	**0,574**	**0,487**	**0,652**	173,686	**0,713**	1.749,160

3.4. Impact of Chunk Size in Fixed-Size Chunking

Table 5 illustrates that extending the chunk size from 128 to 256 tokens enhanced retrieval ranking and answer completeness, whereas smaller chunks provided superior surface-level similarity and expedited processing. Despite both configurations attaining similar generation quality, 256-token chunks delivered a more extensive context at the expense of increased runtime, underscoring a trade-off between efficiency and information comprehensiveness.

Table 5. Experiment Result in Chunk Size

Chunking Configuration	Retrieval						Generation	
	ROUGE-L	BLEU	MRR	MAP	HIT@K	Retrieve Time (ms)	BERT-Score	Generate Time (s)
128	**0,277**	**0,160**	0,434	0,262	0,533	**81,320**	0,712	**973,540**
256	0,196	0,116	**0,521**	**0,390**	**0,648**	147,965	**0,715**	1.106,113

3.5. Impact of Chunk with Overlap

Table 6 illustrates that chunk overlap augmented contextual continuity and elevated generation quality, especially in fixed-size chunking. Larger overlaps resulted in elevated F1-scores, but also prolonged processing time. The 128-token segment with a 100-token overlap provided a nice balance between quality and efficiency. In semantic chunking, overlap yielded marginal performance improvements but resulted in significant latency, indicating that overlap should be utilized selectively based on system priorities.

Table 6. Experiment Result in Chunk Overlap

Chunk Size	Overlap	Retrieval						Generation	
		ROUGE-L	BLEU	MRR	MAP	HIT@K	Retrieve Time (ms)	BERT-Score	Generate Time (s)
128	25	0,261	0,150	0,457	0,332	0,556	83,646	0,710	1.867,068
	50	0,266	0,159	0,444	0,291	0,560	85,938	0,711	557,027
	100	**0,305**	**0,172**	0,402	0,162	0,483	**74,375**	0,715	**496,525**

p-ISSN: 2528-1682
e-ISSN: 2527-9165

Chunk Size	Overlap	Retrieval						Generation	
		ROUGE-L	BLEU	MRR	MAP	HIT@K	Retrieve Time (ms)	BERT-Score	Generate Time (s)
256	25	0,188	0,111	0,508	0,399	0,645	139,859	0,709	1.920,191
	50	0,194	0,115	0,541	0,418	0,645	156,413	0,714	783,696
	100	0,205	0,123	0,514	0,351	0,654	147,624	**0,721**	614,450
Semantic	No	0,165	0,094	0,562	0,473	0,645	171,839	0,713	923,595
	With	0,179	0,102	**0,585**	**0,501**	**0,658**	175,533	0,714	2.574,725

3.6. Discussion

This study addresses the challenge of implementing RAG systems on low-resource devices like mobile phones, where constrained computational power limits both the size of the model and the effectiveness of retrieval. Small Language Models present a viable option compared to larger architectures; however, their effectiveness is significantly reliant on well-optimized chunking and retrieval strategies to produce precise responses. The experimental findings indicate that meticulous adjustment of configurations significantly influences the relation between the quality of answers and the duration of processing. The integration of sparse retrieval with a suitable chunk size and overlap presents a practical approach that enhances overall performance from start to finish. This illustrates that the performance constraints of Small Language Models on local hardware can be alleviated through RAG optimization, instead of depending exclusively on external cloud infrastructure. This study presents a viable approach for implementing domain-specific question answering systems directly on devices, ensuring minimal resource demands.

4. CONCLUSION

This study assessed how various retrieval methods, chunking strategies, chunk sizes, and overlaps influence the efficacy of a Retrieval-Augmented Generation (RAG) system utilizing a Small Language Model (SLM) for answering questions related to Hajj on devices with limited resources. The analysis, which involved 24 configurations and 148 carefully selected question–answer pairs, demonstrates that system parameters have a substantial impact on both the quality of answers and the efficiency of processing. This study addresses the fundamental issue of implementing language models in low-resource settings, where constrained computational power limits reasoning quality and context coverage. It illustrates that an optimized RAG configuration can greatly enhance performance without dependence on cloud infrastructure.

Increased chunk sizes with broader overlaps enhanced answer comprehensiveness by providing additional context, although necessitated extended production time. Although semantic chunking enhanced retrieval ranking, fixed-size chunking resulted in superior generation quality and runtime efficiency. Sparse retrieval provided the optimal equilibrium between precision and velocity among retrieval strategies. The sparse–256–overlap 100 configurations attained the best accuracy, whereas the sparse–128–overlap 100 provided quicker replies with comparable outcomes.

Various evaluation metrics demonstrated trade-offs: MRR, MAP, and Hit@K assessed semantic relevance, but ROUGE and BLEU more accurately represented answer quality from the user's viewpoint. This underscores the necessity of aligning metric selection with system objectives.

Generation performance was influenced by systematic prompting. Prompts featuring explicit instructions, contextual retrieval, and formal Bahasa Indonesia queries enhanced the coherence and relevance of the responses generated by the Gemma-2B-it model. Nonetheless, its restricted context window and reasoning ability continued to pose a limitation. This highlights that both prompt design and the generative model's capabilities are essential to overall system performance. The results indicate that model limitations cannot be mitigated alone through retrieval; efficient system design necessitates both optimum retrieval configuration and meticulous, quick development.

The main contribution of this research is a systematic experimental comparison of RAG setups for complete on-device deployment utilizing an Indonesian Hajj dataset, which has not been previously investigated. The findings offer practical direction for the creation of domain-specific question-answering systems in resource-limited circumstances. Further research may extend the assessment to

multi-turn interactions and a wider range of Hajj and Umrah paperwork to evaluate system robustness in authentic user scenarios.

ACKNOWLEDGEMENTS

This study was funded by Politeknik Negeri Bandung through the Biaya Penelitian Tugas Akhir Mahasiswa Program Sarjana Terapan in 2025, under contract number 317/PL1/HK.02/2025. We extend our gratitude to our colleagues from Politeknik Negeri Bandung for their valuable insights and expertise that significantly contributed to this research, notwithstanding any potential disagreements with the interpretations or conclusions presented in this paper.

REFERENCES

[1] Y. Gao *et al.*, 'Retrieval-Augmented Generation for Large Language Models: A Survey', Dec. 2023, [Online]. Available: http://arxiv.org/abs/2312.10997

[2] X. Wang *et al.*, 'Searching for Best Practices in Retrieval-Augmented Generation', Jul. 2024, [Online]. Available: http://arxiv.org/abs/2407.01219

[3] J. Liu, R. Ding, L. Zhang, P. Xie, and F. Huang, 'CoFE-RAG: A Comprehensive Full-chain Evaluation Framework for Retrieval-Augmented Generation with Enhanced Data Diversity', Oct. 2024, [Online]. Available: http://arxiv.org/abs/2410.12248

[4] R. F. Reza, Muhmmad Thoriq, and Rd. Imam Saepul Millah, 'Sentiment Analysis of Marketplace Review with Islamic Perspective using Fine-Tuning DistilBERT', *Khazanah Journal of Religion and Technology*, vol. 2, no. 2, pp. 45–54, Jan. 2025, doi: 10.15575/kjrt.v2i2.1118.

[5] C. Van Nguyen *et al.*, 'A Survey of Small Language Models', Oct. 2024, [Online]. Available: http://arxiv.org/abs/2410.20011

[6] T. Fan, J. Wang, X. Ren, and C. Huang, 'MiniRAG: Towards Extremely Simple Retrieval-Augmented Generation', Jan. 2025, [Online]. Available: http://arxiv.org/abs/2501.06713

[7] Z. Lu *et al.*, 'Small Language Models: Survey, Measurements, and Insights', Sep. 2024, [Online]. Available: http://arxiv.org/abs/2409.15790

[8] P. Lewis *et al.*, 'Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks', Apr. 2021, [Online]. Available: http://arxiv.org/abs/2005.11401

[9] D. Kim, B. Kim, D. Han, and M. Eibich, 'AutoRAG: Automated Framework for optimization of Retrieval Augmented Generation Pipeline', Oct. 2024, [Online]. Available: http://arxiv.org/abs/2410.20878

[10] Nerve Sparks, 'nerve-sparks/iris_android', 2025, *India*.

[11] Shubham Panchal, 'shubham0204/Android-Document-QA', 2025, *India*.

[12] S. Setty, H. Thakkar, A. Lee, E. Chung, and N. Vidra, 'Improving Retrieval for RAG based Question Answering Models on Financial Documents', Mar. 2024, [Online]. Available: http://arxiv.org/abs/2404.07221

[13] R. Qu, R. Tu, and F. Bao, 'Is Semantic Chunking Worth the Computational Cost?', Oct. 2024, [Online]. Available: http://arxiv.org/abs/2410.13070

[14] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan, 'Query Rewriting for Retrieval-Augmented Large Language Models', 2023. [Online]. Available: https://github.com/xbmxb/RAG-query-rewriting

[15] S.-C. Lin, J.-H. Yang, R. Nogueira, M.-F. Tsai, C.-J. Wang, and J. Lin, 'Multi-Stage Conversational Passage Retrieval: An Approach to Fusing Term Importance Estimation and Neural Query Rewriting', Mar. 2021, [Online]. Available: http://arxiv.org/abs/2005.02230

[16] P. Mandikal and R. Mooney, 'Sparse Meets Dense: A Hybrid Approach to Enhance Scientific Document Retrieval', Jan. 2024, [Online]. Available: http://arxiv.org/abs/2401.04055

[17] B. G. Chepino, R. R. Yacoub, A. Aula, M. Saleh, and B. W. Sanjaya, 'EFFECT OF MINMAX NORMALIZATION ON ORB DATA FOR IMPROVED ANN ACCURACY', *Journal of Electrical Engineering, Energy, and Information Technology (J3EIT)*, vol. 11, no. 2, p. 29, Aug. 2023, doi: 10.26418/j3eit.v11i2.68689.

[18] Bartowski, 'bartowski/gemma-2-2b-it-GGUF', Hugging Face.

[19] J. Park, K. Atarashi, K. Takeuchi, and H. Kashima, 'Emulating Retrieval Augmented Generation via Prompt Engineering for Enhanced Long Context Comprehension in LLMs', Feb. 2025, [Online]. Available: http://arxiv.org/abs/2502.12462

[20] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, and Z. Liu, 'Evaluation of Retrieval-Augmented Generation: A Survey', May 2024, doi: 10.1007/978-981-96-1024-2_8.