(2026), 4 (1): 246-259 **Scientica** 3021-8209

Jurnal Ilmiah Sain dan Teknologi

IMPLEMENTASI SISTEM *RETRIEVAL-AUGMENTED GENERATION* (RAG) STUDI KASUS: *QUESTION ANSWERING* DARI DOKUMEN PDF

Apbenk Fathur Rahman [1)], Dony Novaliendry [2)], Yeka Hendriyani [3)], Khairi Budayawan [4)]

Program Studi Informatika, Universitas Negeri Padang

Alamat e-mail: apbenkfathur@gmail.com

Abstract (English)

The increasing need for information management and analysis systems has become more evident with the rapid growth of digital data. Digital documents such as PDF files present challenges in obtaining specific information quickly and accurately due to their static nature and reliance on manual search. This research aims to develop a question answering system based on Retrieval-Augmented Generation (RAG) that is capable of extracting and generating answers directly from PDF documents. The system is implemented as a web-based application using the Flask framework and developed through the Waterfall method, which consists of requirements analysis, system design, implementation, testing, and maintenance stages. The RAG pipeline integrates document segmentation, semantic embedding, and vector-based indexing to support semantic similarity retrieval, which is then combined with a Large Language Model (LLM) for answer generation. The results demonstrate that the proposed system is able to generate relevant and accurate answers that are consistent with the content of the source documents, thereby improving information retrieval efficiency and reducing the risk of information hallucination. In addition, session management and password hashing are applied to ensure data security and the confidentiality of user documents.

Article History

Submitted: 27 Januari 2026

Accepted: 30 Januari 2026

Published: 31 Januari 2026

Key Words

Digital Documents (PDF), *Question Answering*, *Retrieval-Augmented Generation* (RAG), *Large Language Model* (LLM).

Abstrak (Indonesia)

Peningkatan kebutuhan akan sistem pengelolaan dan analisis informasi semakin nyata seiring dengan pertumbuhan data digital yang pesat. Dokumen digital seperti PDF menimbulkan tantangan dalam memperoleh informasi spesifik secara cepat dan akurat akibat sifatnya yang statis serta ketergantungan pada pencarian manual. Penelitian ini bertujuan untuk mengembangkan sistem *question answering* berbasis Retrieval-Augmented Generation (RAG) yang mampu mengekstraksi dan menghasilkan jawaban secara langsung dari dokumen PDF. Sistem diimplementasikan sebagai aplikasi web berbasis framework Flask dan dikembangkan menggunakan metode Waterfall yang meliputi tahapan analisis kebutuhan, perancangan sistem, implementasi, pengujian, dan pemeliharaan. Pipeline RAG mengintegrasikan proses segmentasi dokumen, pembentukan *semantic embedding*, serta pengindeksan berbasis vektor untuk mendukung pencarian berdasarkan kesamaan makna, yang kemudian dipadukan dengan *Large Language Model* (LLM) dalam proses generasi jawaban. Hasil penelitian menunjukkan bahwa sistem mampu menghasilkan jawaban yang relevan dan akurat sesuai dengan isi dokumen sumber, meningkatkan efisiensi pencarian informasi, serta meminimalkan risiko terjadinya halusinasi informasi. Selain itu, penerapan manajemen sesi dan mekanisme *password hashing* memastikan keamanan serta kerahasiaan dokumen pengguna.

Sejarah Artikel

Submitted: 27 Januari 2026

Accepted: 30 Januari 2026

Published: 31 Januari 2026

Kata Kunci

Dokumen digital (PDF), *Question Answering*, *Retrieval-Augmented Generation* (RAG), *Large Language Model* (LLM).

Pendahuluan

Pada era digital saat ini, dokumen digital seperti PDF, laporan akademik, kontrak, maupun arsip organisasi menjadi salah satu sumber informasi yang paling banyak digunakan. Namun, semakin bertambahnya jumlah dokumen tersebut, dapat menimbulkan tantangan baru yaitu bagaimana pengguna dapat dengan cepat menemukan dan memperoleh jawaban spesifik

dari isi dokumen tanpa harus membaca keseluruhan teks. Tantangan ini mendorong perlunya sistem cerdas yang tidak hanya dapat menyimpan dokumen, tetapi juga mampu melakukan *question answering* secara langsung berdasarkan isi dokumen digital. Permasalahan ini semakin mendesak terutama dalam lingkungan akademik dan organisasi, di mana pengguna dituntut untuk memperoleh informasi secara cepat, akurat, dan berbasis sumber yang jelas.

Peningkatan kebutuhan akan sistem pengelolaan dan analisis informasi semakin nyata seiring dengan pertumbuhan data digital yang sangat besar. Menurut Danopoulos et al. (2019), era *big* data ditandai dengan pemrosesan data dalam jumlah masif yang dapat mencapai triliunan hingga kuintiliun *byte* setiap harinya. PDF merupakan salah satu format dokumen digital yang paling luas digunakan di era *big* data, baik dibidang akademik maupun organisasi. Dokumen digital, termasuk PDF, merupakan bagian penting dari data tersebut karena menyimpan informasi bernilai tinggi dalam bentuk teks. Oleh karena itu, dibutuhkan metode yang tidak hanya mampu menyimpan dokumen sebagai arsip, tetapi juga dapat mengekstraksi pengetahuan yang terkandung di dalamnya untuk dimanfaatkan secara efektif.

Salah satu pendekatan modern yang berkembang untuk menjawab tantangan tersebut adalah *Retrieval-Augmented Generation* (RAG). RAG menggabungkan sistem pencarian informasi (*retrieval*) dengan kemampuan generatif dari *Large Language Model* (LLM). Pendekatan ini tidak hanya sekadar melakukan pencarian berbasis kata kunci, melainkan juga memungkinkan interaksi yang lebih kaya seperti *question answering*, eksplorasi konteks, hingga pemahaman mendalam terhadap isi dokumen (Fast et al., 2006). Dengan cara ini, pengguna dapat memperoleh jawaban yang relevan dan kontekstual langsung dari dokumen digital yang dianalisis.

Retrieval-Augmented Generation (RAG) memungkinkan sistem untuk menjawab pertanyaan pengguna berdasarkan isi dokumen yang disediakan tanpa perlu melatih ulang model bahasa besar. Dalam konteks dokumen PDF, pendekatan ini bekerja dengan cara membagi teks dokumen ke dalam potongan kecil (*chunking*), lalu merepresentasikannya dalam bentuk vektor melalui proses *semantic embedding*. Selanjutnya, vektor-vektor tersebut disimpan dalam indeks pencarian seperti FAISS, sehingga sistem dapat dengan cepat menemukan bagian dokumen yang paling relevan dengan pertanyaan. Informasi hasil *retrieval* kemudian dipadukan dengan kemampuan generatif LLM untuk menghasilkan jawaban yang lebih akurat, kontekstual, dan sesuai dengan isi dokumen (Lewis et al., 2020; Karpukhin et al., 2020; Johnson et al., 2017; Reimers & Gurevych, 2019).

Kemampuan *Large Language Model* (LLM) dalam menjawab pertanyaan, baik umum maupun spesifik, telah terbukti pada berbagai penelitian sebelumnya (Jeong, 2023; Beam et al., 2023; Kamalloo et al., 2023). Namun, penggunaan pengetahuan internal LLM saja sering kali tidak cukup untuk menjamin kesesuaian jawaban dengan konteks eksternal. Huang et al. (2025) menunjukkan bahwa konflik dapat muncul ketika model mengandalkan memori internalnya, sehingga jawaban yang dihasilkan kurang akurat atau bahkan menyimpang dari isi dokumen yang menjadi acuan. Oleh karena itu, integrasi mekanisme *retrieval* melalui pendekatan RAG menjadi penting agar LLM dapat mengakses informasi relevan dari dokumen yang telah di indeks, sehingga jawaban lebih berbasis fakta dan sesuai konteks.

Dalam praktiknya, banyak pengguna memanfaatkan LLM secara langsung melalui *prompt* umum untuk menanyakan isi dokumen PDF. Pendekatan berbasis *prompt* ini bersifat manual, tidak terstruktur, dan sangat bergantung pada kemampuan pengguna dalam merancang pertanyaan yang tepat. Selain itu, pendekatan tersebut memiliki keterbatasan konteks dan tidak menjamin bahwa jawaban yang dihasilkan benar-benar bersumber dari dokumen yang dimaksud. Kondisi ini menimbulkan urgensi akan pengembangan sistem yang mampu mengelola dokumen secara terstruktur serta menyediakan mekanisme *question answering* yang konsisten, akurat, dan dapat ditelusuri sumbernya. Oleh karena itu, penelitian ini menjadi penting untuk membedakan pendekatan sistematis berbasis *Retrieval-Augmented Generation*

(RAG) dari penggunaan LLM berbasis *prompt* semata, khususnya dalam konteks pengelolaan dan pemanfaatan dokumen PDF.

Sasaran dari penelitian ini adalah pengguna di lingkungan akademik dan organisasi yang secara rutin berinteraksi dengan dokumen PDF, seperti mahasiswa, dosen, peneliti, serta staf administrasi. Kelompok pengguna ini membutuhkan akses informasi yang cepat dan akurat dari dokumen digital tanpa harus membaca keseluruhan isi dokumen secara manual.

Berdasarkan latar belakang tersebut, penulis tertarik untuk membangun sistem sederhana berbasis *Retrieval-Augmented Generation* (RAG) dengan studi kasus *question answering* dari dokumen PDF. Sistem ini dirancang untuk mengekstraksi informasi penting dari dokumen melalui mekanisme *chunking*, *semantic embedding*, dan indeks vektor, lalu memanfaatkan kemampuan generatif LLM untuk menyajikan jawaban yang relevan. Dengan pendekatan ini, sistem tidak hanya mengandalkan kecerdasan bahasa dari LLM, tetapi juga memanfaatkan proses *retrieval* agar informasi yang ditampilkan sesuai dengan isi dokumen. Penelitian ini diharapkan dapat menjadi prototipe awal menuju solusi dokumen cerdas berbasis AI yang mampu memberikan jawaban akurat dan kontekstual dari dokumen digital.

Metode Penelitian

Metode pengembangan sistem yang digunakan dalam pembuatan sistem *Retrieval-Augmented Generation* (RAG) untuk *question answering* dari dokumen PDF adalah *Waterfall. Model Waterfall* merupakan salah satu pendekatan dalam *System Development Life Cycle* (SDLC) yang memandang proses pengembangan perangkat lunak sebagai suatu alur yang sistematis, berurutan, dan saling bergantung antar tahapan. Karakteristik utama dari metode ini adalah setiap tahapan harus diselesaikan secara tuntas sebelum dapat melanjutkan ke tahapan berikutnya, sehingga tidak terdapat pekerjaan paralel maupun proses kembali ke tahap sebelumnya. Urutan tahapan dalam model *Waterfall* terdiri dari analisis kebutuhan, perancangan sistem, implementasi, pengujian, serta pemeliharaan. Dengan alur yang jelas dan terstruktur, model ini memudahkan tim pengembang dalam mengidentifikasi kebutuhan sejak awal, merancang solusi teknis secara detail, serta memastikan hasil akhir sesuai dengan rancangan awal. Model ini memberikan alur kerja yang terstruktur dan mudah dipantau dalam pengembangan sistem.

Penerapan metode *Waterfall* pada pengembangan sistem RAG memungkinkan setiap tahapannya difokuskan secara maksimal untuk mencapai tujuan yang diharapkan. Pada tahap analisis kebutuhan, sistem didefinisikan untuk mampu membaca dan memproses dokumen PDF, kemudian mengekstraksi informasi penting yang dapat digunakan untuk menjawab pertanyaan pengguna. Tahap perancangan dilakukan dengan mendefinisikan arsitektur sistem, alur proses *retrieval, embedding*, hingga *generation* yang menjadi inti dari RAG. Selanjutnya tahap implementasi merealisasikan rancangan tersebut ke dalam bentuk perangkat lunak dengan memanfaatkan bahasa pemrograman, *framework*, serta model *machine learning* untuk *embedding* dan LLM sebagai generator. Tahap pengujian dilakukan untuk memastikan sistem mampu memberikan jawaban yang akurat dan relevan berdasarkan isi dokumen, serta berjalan stabil sesuai spesifikasi. Terakhir, tahap pemeliharaan dilakukan untuk memperbaiki *bug*, meningkatkan performa, dan menyesuaikan sistem dengan kebutuhan baru. Dengan karakteristik sekuensialnya, model *Waterfall* mendukung pengembangan sistem RAG agar berjalan lebih terstruktur dan terkontrol.

Hasil dan Pembahasan

A. Hasil Implementasi Autentikasi

Sistem menerapkan mekanisme keamanan untuk melindungi akses pengguna dan memisahkan data antar pengguna. Proses registrasi bertujuan untuk mendaftarkan identitas

pengguna baru ke dalam basis data. Fitur ini dibangun menggunakan kombinasi *library werkzeug.security* untuk kriptografi dan *flask* untuk manajemen sesi.

```python
def register():
    data = request.json
    username = data.get("username")
    password = data.get("password")

    if not username or not password:
        return jsonify({"error": "Data tidak lengkap"}), 400

    hashed_pw = generate_password_hash(password)
```

Figure 1. Kode implementasi hashing password pada registrasi

Pada proses registrasi yang terlihat pada figure 1 di atas, sistem tidak menyimpan kata sandi pengguna dalam bentuk teks asli (*plain text*). Library *werkzeug security* digunakan melalui fungsi *generate_password_hash* untuk mengubah kata sandi menjadi format *hash* yang terenkripsi sebelum disimpan ke basis data *MySQL*. Hal ini menjamin keamanan data kredensial jika terjadi kebocoran data. Antarmuka halaman registrasi ini dapat dilihat seperti gambar 2 di bawah ini.

Figure 2. Antarmuka dari proses registrasi

Figure 2 menunjukkan antarmuka tempat pengguna membuat *username* dan *password* untuk pertama kali, selain itu pengguna juga harus memasukan nama lengkap dan juga alamat email sebagai kelengkapan data pengguna. Setelah itu pengguna bisa masuk melalui halaman *login*.

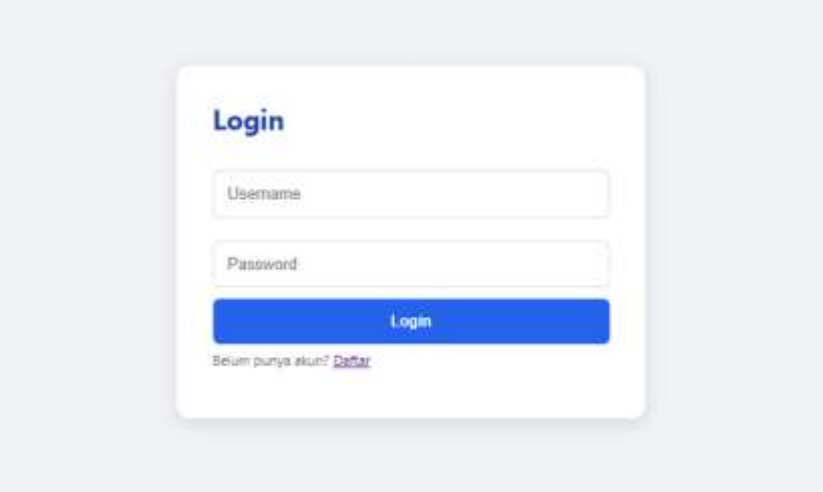

Figure 3. Antarmuka proses login

Selanjutnya pada proses *login*, sistem memverifikasi identitas pengguna menggunakan fungsi *check_password_hash*. Fungsi ini membandingkan kata sandi yang dimasukkan saat login dengan hash yang tersimpan di database. Jika cocok, sistem memanfaatkan objek session dari *framework Flask* untuk menyimpan user_id. Sesi ini berfungsi sebagai token otorisasi sementara yang memungkinkan pengguna mengakses Implementasi kode untuk proses login dapat dilihat pada gambar 4 di bawah.

```
def login():
    data = request.json
    username = data.get("username")
    password = data.get("password")

    db = get_db_connection()
    cur = db.cursor(dictionary=True)
    cur.execute("SELECT * FROM users WHERE username = %s", (username,))
    user = cur.fetchone()
    cur.close()
    db.close()

    if user and check_password_hash(user['password'], password):
        session['user_id'] = user['id']
        session['username'] = user['username']
        return jsonify({"message": "Login berhasil"})

    return jsonify({"error": "Username atau password salah"}), 401
```

Figure 4. Kode implementasi login

Pada halaman Login, pengguna dapat masuk ke dalam sistem menggunakan username dan password yang telah didaftarkan sebelumnya. Apabila proses autentikasi berhasil, sistem akan mengarahkan pengguna ke halaman dashboard sistem *RAG (Retrieval-Augmented Generation).*

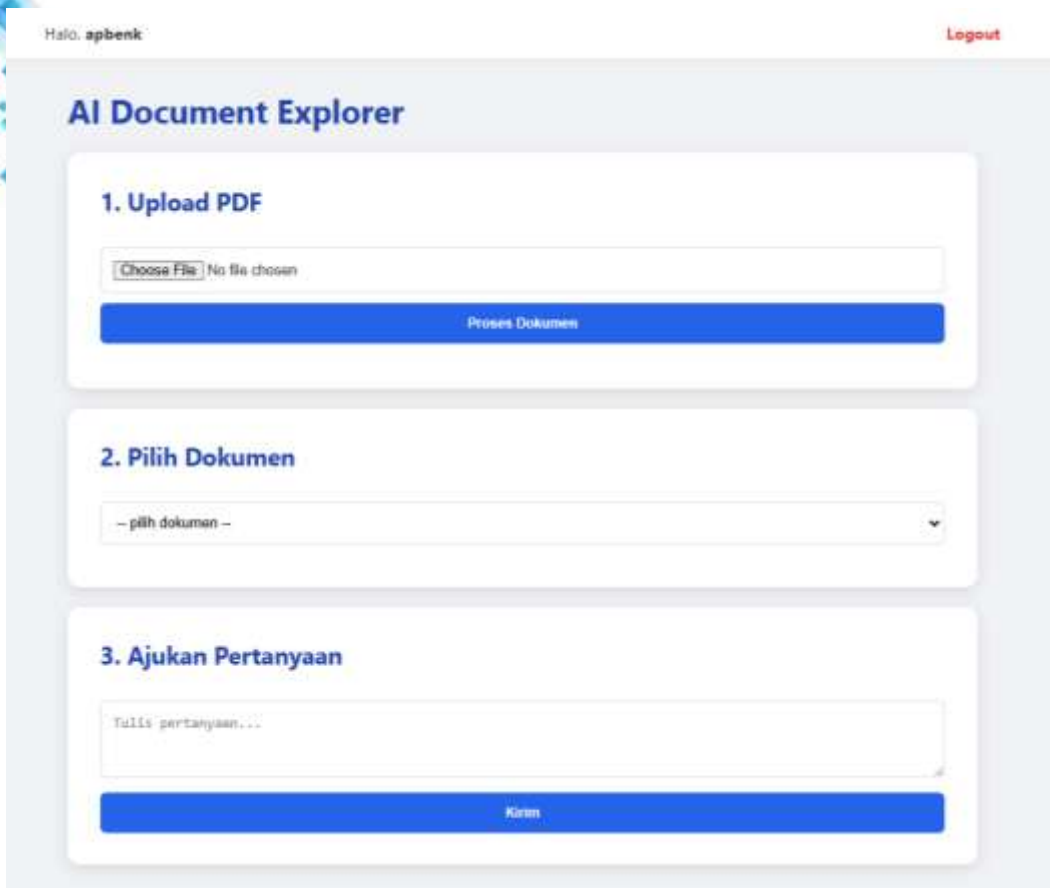

Figure 5. Halaman dashboard

Halaman *dashboard* ini merupakan pusat aktivitas pengguna, di mana pengguna dapat mengunggah dokumen baru, memilih dokumen yang telah diunggah sebelumnya, serta mengakses riwayat pertanyaan dan jawaban yang pernah dilakukan, riwayat akan otomatis muncul ketika user memilih dokumen yang sudah pernah dilakukan tanya jawab sebelumnya. Seluruh fitur tersebut hanya dapat diakses selama sesi login masih aktif, sehingga keamanan dan pemisahan data antar pengguna tetap terjaga.

B. Hasil Implementasi Pengolahan Dokumen

Tahap ini merupakan fondasi dari sistem RAG, di mana dokumen PDF mentah diubah menjadi basis pengetahuan yang dapat dicari. Proses ini melibatkan tiga *library* utama: *fitz* (PyMuPDF), *sentence transformers, dan faiss*.

1. Proses Ekstraksi PDF dan *Chunking*

Langkah pertama adalah Ekstraksi Teks. Ketika pengguna mengunggah *file* PDF, sistem menggunakan *fitz* untuk membuka dokumen dan mengekstrak teks dari seluruh halaman. *Library fitz* dipilih karena performanya yang tinggi dalam membaca struktur PDF. Setelah teks diekstrak, dilakukan proses *chunking* (pemecahan teks) menjadi potongan-potongan kecil berukuran 500 karakter. Pemecahan ini krusial agar konteks yang diproses tidak melebihi batas memori model AI.

```
doc = fitz.open(pdf_path)
text = " ".join([page.get_text() for page in doc])

# 3. Chunking & Embedding
chunk_size = 500
overlap = 100
step = chunk_size - overlap
chunks = [text[i : i + chunk_size] for i in range(0, len(text), step)]
```

Figure 6. Proses ektraksi teks dan chunking

Pada figure 6 di atas sistem menggunakan PyMuPDF (fitz) untuk membaca dokumen. Untuk menjaga keutuhan konteks, teks dipecah menjadi chunks berukuran 500 karakter dengan teknik overlapping sebesar 100 karakter. Strategi overlap ini sangat penting agar informasi yang berada di perbatasan pemotongan tidak hilang.

2. Proses Embedding dan FAISS

```
# Inisialisasi model embedding lokal
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
embeddings = embedding_model.encode(chunks)

# Membuat indeks FAISS
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(np.array(embeddings))
faiss.write_index(index, faiss_path)
```

Figure 7. Kode implementasi proses embedding dan FAISS

Potongan kode pada figure 7 di atas menunjukkan proses transformasi teks menjadi vektor numerik menggunakan model all-MiniLM-L6-v2. Vektor tersebut kemudian disimpan ke dalam Vector Database FAISS. Sebagai hasil akhir dari rangkaian proses tersebut, sistem menghasilkan dan mengelola tiga komponen output utama secara terstruktur. *Pertama*, file PDF asli disimpan ke dalam direktori penyimpanan sistem, di mana lokasi penyimpanannya (file path) dicatat ke dalam database SQL. *Kedua*, hasil *chunking* teks diekspor menjadi file terpisah dan disimpan dalam sistem dalam format .txt, path menuju file ini kemudian diperbarui ke dalam kolom dokumen pada database SQL. *Ketiga*, *indeks* vektor yang telah terbentuk disimpan sebagai file FAISS berekstensi .index pada sistem lokal, yang nantinya akan dipanggil kembali saat proses pencarian (*retrieval*) dilakukan. Implementasi kode untuk menyimpan hasil proses ini dapat dilihat pada gambar di bawah ini.

```python
# 1. Simpan File PDF
filename = file.filename
pdf_path = os.path.join(STORAGE["pdf"], filename)
file.save(pdf_path)

# 2. Simpan Chunks Teks ke File
chunk_file = os.path.join(STORAGE["chunks"], f"{filename}.txt")
with open(chunk_file, "w", encoding="utf-8") as f:
    f.write("\n---CHUNK_SEP---\n".join(chunks))

# 3. Simpan Vector Index (FAISS)
faiss_path = os.path.join(STORAGE["faiss"], f"{filename}.index")
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(np.array(embeddings))
faiss.write_index(index, faiss_path)

# 4. Simpan Metadata ke Database
db = get_db_connection()
cur = db.cursor()
cur.execute(
    "INSERT INTO documents (filename, pdf_path, faiss_path, user_id) VALUES (%s, %s, %s, %s)",
    (filename, pdf_path, faiss_path, session['user_id'])
)
db.commit()
cur.close()
db.close()
```

Figure 8. Kode implementasi penyimpanan hasil read, chunk dan FAISS

Proses pada figure 8 di atas dieksekusi secara otomatis ketika pengguna mengunggah dokumen melalui *dashboard*. Alur ini mencakup pembacaan dokumen, konversi teks menjadi vektor *embedding* (disimpan dengan ekstensi *index*), hingga akhirnya sistem menyimpan seluruh lokasi (*file path*) dari hasil tersebut ke dalam *database*.

a. Hasil Implementasi Sistem Tanya Jawab

Fitur utama sistem adalah kemampuan menjawab pertanyaan berdasarkan dokumen (RAG). Proses ini menggabungkan pencarian semantik (*Retrieval*) dan pembuatan teks generatif (*Generation*).

1. Retrieval

Pada tahap *Retrieval*, sistem menerima pertanyaan pengguna dan mengubahnya menjadi vektor menggunakan model *sentence_transformers* yang sama dengan proses pengolahan dokumen. Menggunakan pustaka *faiss* dan bantuan *numpy*, sistem mencari 3 potongan teks *(chunks)* dari dokumen yang memiliki jarak vektor terdekat atau yang paling relevan dengan pertanyaan tersebut.

```
question = request.form.get("question")
doc_id = request.form.get("document_id")

# 1. Ambil path file dari Database
db = get_db_connection()
cur = db.cursor(dictionary=True)
cur.execute("SELECT * FROM documents WHERE id = %s AND user_id = %s", (doc_id, session['user_id']))
doc_data = cur.fetchone()

if not doc_data:
    return jsonify({"error": "Dokumen tidak ditemukan"}), 404

# 2. Load FAISS Index & Text Chunks
index = faiss.read_index(doc_data['faiss_path'])

chunk_file = os.path.join(STORAGE["chunks"], f"{doc_data['filename']}.txt")
with open(chunk_file, "r", encoding="utf-8") as f:
    all_chunks = f.read().split("\n---CHUNK_SEP---\n")

# 3. Cari Konteks Relevan (Search)
q_embed = embedding_model.encode([question])
# Cari 3 chunk paling mirip
I = index.search(np.array(q_embed), k=3)
context = "\n".join([all_chunks[i] for i in I[0] if i < len(all_chunks)])
```

Figure 9. Kode implementasi proses *retrieval*

Pada figure 9 di atas, alur dimulai ketika *backend* menerima data *question* dan *document_id* dari *request form*. Langkah pertama adalah mengambil jalur *(path)* file FAISS dan teks dari database SQL. Setelah file indeks FAISS dibaca kembali oleh sistem, pertanyaan pengguna diproses melalui model *embedding (all-MiniLM-L6-v2).* Hasil vektor pertanyaan tersebut kemudian dicocokkan dengan indeks untuk mendapatkan tiga potongan teks *(chunks)* dengan skor kemiripan tertinggi. Potongan teks terpilih ini kemudian digabungkan menjadi satu kesatuan konteks yang siap dikirimkan ke LLM. Berikut ini adalah contoh konteks yang berhasil di *retrieve* ketika pengguna mengajukan pertanyaan.

Figure 10. Contoh hasil konteks pada tahap *retrieval*

Terlihat pada figure 10 konteks dari sebuah dokumen pdf yang berbentuk sebuah artikel yang berjudul “Pengembangan Chatbot Berbasis Pdf Menggunakan Local Retrieval-Augmented Generation (Rag) Dan Ollama” dengan pertanyaan “Jenis database apa yang

digunakan?" sistem RAG menghasilkan konteks yang terlihat seperti potongan potongan kalimat acak, di sini lah proses selanjutnya dibutuhkan, yaitu LLM untuk membuat potongan kalimat acak ini menjadi lebih *natural*.

2. Generation

Langkah terakhir adalah mengirimkan konteks dan pertanyaan ke *Large Language Model* (LLM) untuk mendapatkan jawaban yang *natural*.

```
context = "\n".join([all_chunks[i] for i in I[0] if i < len(all_chunks)])

# 4. Kirim ke AI (LLM)
prompt = f"Gunakan konteks berikut untuk menjawab.\nKonteks:\n{context}\n\nPertanyaan: {question}"

response = ai_client.chat.completions.create(
    model="gemini-2.5-flash",
    messages=[{"role": "user", "content": prompt}]
)
answer = response.choices[0].message.content
```

Figure 11. Kode implementasi proses *Generation*

Pada tahap *generation* yang terlihat pada figure 11 di atas, pada tahap ini konteks yang ditemukan digabungkan dengan pertanyaan asli ke dalam sebuah *prompt*. *Prompt* disusun sedemikian rupa agar model *Gemini* hanya menjawab berdasarkan fakta di dalam dokumen, sehingga meminimalisasi kesalahan informasi atau halusinasi. Sistem menggunakan pustaka open Ai sebagai klien untuk berkomunikasi dengan *API Google Gemini (gemini-2.5-flash)*.

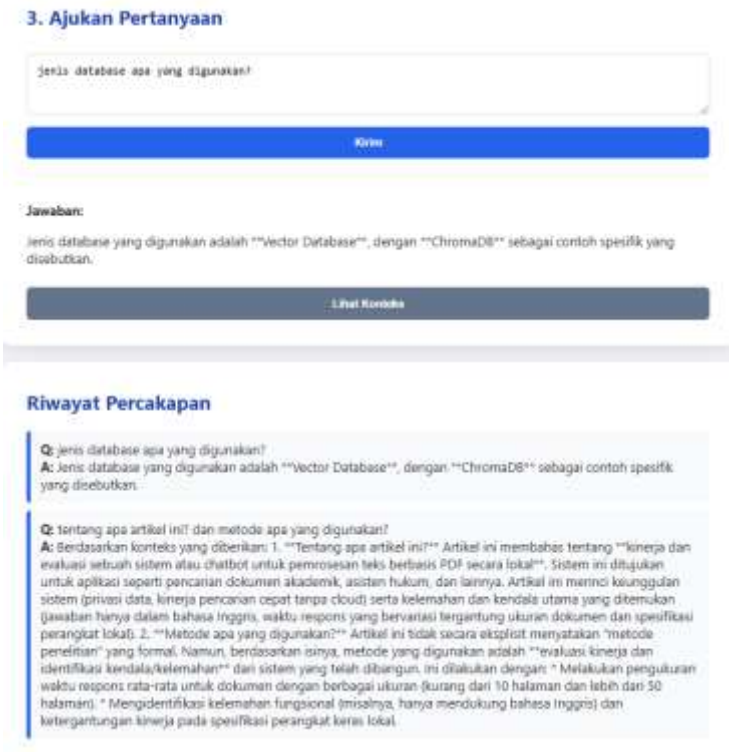

Figure 12. Hasil dari tahap generation

Terlihat pada figure 12 di atas, setelah pengguna mengunggah dokumen dan memilih dokumen yang akan digunakan, pengguna dapat mengajukan pertanyaan melalui antarmuka web. Pertanyaan pengguna dan dokumen akan diproses oleh sistem RAG sehingga mendapatkan informasi yang faktual yang akan dikirimkan ke model LLM (Figure 12). Model LLM kemudian menghasilkan jawaban berdasarkan konteks tersebut, dan hasil pertanyaan beserta jawaban yang dihasilkan akan ditampilkan pada bagian riwayat percakapan pada

dashboard dan disimpan ke dalam basis data SQL pada tabel *qa history* sebagai riwayat interaksi.

b. Hasil Perancangan dan Relasi Basis Data

Pada sistem *Retrieval-Augmented Generation* (RAG) yang dibangun, basis data dirancang untuk mendukung pengelolaan pengguna, dokumen, serta riwayat tanya jawab secara terstruktur dan terisolasi antar pengguna. Basis data terdiri dari tiga tabel utama, yaitu *users*, *documents*, dan *qa history*, yang saling terhubung melalui relasi kunci asing (*foreign key*).

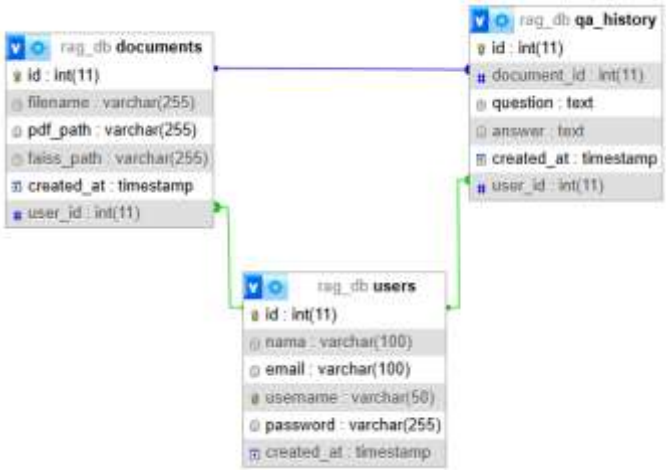

Figure 13. Relasi tabel database

Tabel *users* berfungsi untuk menyimpan data pengguna yang telah melakukan registrasi, seperti nama, *email*, *username*, *password* dalam bentuk *hash*, serta waktu pembuatan akun. Tabel *documents* digunakan untuk menyimpan metadata dokumen PDF yang diunggah oleh pengguna. Informasi yang disimpan meliputi nama file, lokasi penyimpanan file PDF, lokasi file indeks FAISS, waktu unggah, serta *user_id* sebagai *foreign key* yang merujuk ke tabel *users*. Relasi ini menunjukkan bahwa satu pengguna dapat memiliki banyak dokumen (*one-to-many*), namun setiap dokumen hanya dimiliki oleh satu pengguna.

Selanjutnya, tabel *qa_history* menyimpan riwayat pertanyaan dan jawaban yang dihasilkan oleh sistem RAG. Tabel ini menyimpan pertanyaan pengguna, jawaban dari model LLM, waktu interaksi, serta *user_id* dan *document_id* sebagai *foreign key*. Relasi ini memungkinkan sistem untuk mengaitkan setiap pertanyaan tidak hanya dengan pengguna yang mengajukan, tetapi juga dengan dokumen yang digunakan sebagai sumber jawaban.

c. Analisis Hasil (*Blackbox Testing*)

Table 1. Blackbox Testing

No	Fitur yang diuji	Skenario pengujian	Output yang diharapkan	Hasil pengujian
1	Register	User daftar melalui web	User terdaftar ke dalam sistem	Valid
2	*Login*	User *login* melalui web	User bisa masuk ke *dashboard* menggunakan *username* dan password terdaftar	Valid
3	*Upload* Dokumen	User mengunggah dokumen	User dapat mengunggah dokumen berfromat pdf	Valid

4	Pilih Dokumen	User memilih dokumen	User dapat memilih dokumen yang ingin di gunakan dalam proses Q&A	Valid
5	Ajukan pertanyaan	User mengajukan pertanyaan	User dapat mengajukan pertanyaan dan sistem dapat memberikan jawaban berdasarkan dokumen	Valid
6	Melihat riwayat	User Melihat riwayat Q&A sebelumnya	User dapat melihat Riwayat Q&A sebelumnya	Valid

Berdasarkan hasil implementasi dan pengujian *black-box* yang telah dilakukan, sistem *Retrieval Augmented Generation (RAG)* berhasil diimplementasikan secara menyeluruh, mencakup autentikasi pengguna, pengolahan dokumen PDF, pencarian semantik, hingga penyajian jawaban berbasis dokumen. Integrasi *library fitz (PyMuPDF), sentence_transformers*, dan FAISS terbukti efektif dalam membangun pemrosesan dokumen yang efisien, di mana proses ekstraksi teks dan *chunking* berjalan stabil serta mampu menjaga konteks informasi. Penggunaan *embedding* lokal meningkatkan efisiensi biaya dan kontrol data, sementara FAISS memberikan performa pencarian yang cepat dan akurat sehingga jawaban yang dihasilkan lebih relevan dan minim halusinasi. Pada tahap *generation*, integrasi klien OpenAI dengan model *Gemini* memungkinkan sistem menghasilkan jawaban bahasa alami yang kontekstual berdasarkan dokumen sumber. Selain itu, penerapan autentikasi menggunakan *werkzeug.security* dan manajemen sesi *Flask* memastikan isolasi data antar pengguna, sehingga sistem aman digunakan dalam lingkungan multi-pengguna.

Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian yang telah dilakukan pada sistem Retrieval-Augmented Generation (RAG), dapat disimpulkan bahwa sistem berbasis RAG berhasil dibangun dan diimplementasikan dalam bentuk aplikasi web menggunakan framework Flask. Sistem ini mampu membaca dokumen PDF dan menjawab pertanyaan pengguna secara otomatis, sehingga dapat mengatasi permasalahan sulitnya memperoleh informasi secara cepat dari dokumen PDF yang bersifat statis apabila hanya mengandalkan pencarian manual.

Pipeline RAG pada sistem telah terintegrasi secara menyeluruh, mulai dari proses pemotongan teks dokumen menjadi potongan berukuran 500 karakter, pembentukan embedding semantik menggunakan model *all-MiniLM-L6-v2*, hingga proses pengindeksan vektor menggunakan library FAISS. Integrasi komponen-komponen tersebut memungkinkan sistem melakukan pencarian informasi berdasarkan kesamaan makna (semantic similarity), bukan hanya berdasarkan kecocokan kata kunci semata.

Berdasarkan hasil pengujian, penggunaan konteks hasil pencarian FAISS dengan pendekatan *top-k* sebanyak tiga potongan teks yang diproses oleh model generatif melalui Gemini API terbukti mampu menghasilkan jawaban yang relevan dan akurat. Jawaban yang dihasilkan tetap berlandaskan pada isi dokumen yang diunggah oleh pengguna, sehingga risiko terjadinya halusinasi informasi dapat diminimalkan.

Selain itu, aspek keamanan data pengguna juga telah diperhatikan melalui penerapan manajemen sesi serta mekanisme *password hashing*. Dengan implementasi tersebut, dokumen yang diunggah bersifat privat dan hanya dapat diakses serta diproses oleh pengguna yang memiliki hak akses, sehingga keamanan dan kerahasiaan data pengguna dapat terjaga dengan baik.

Referensi

Albert, G. D., & Voutama, A. (2025). Pengembangan chatbot berbasis PDF menggunakan local retrieval-augmented generation (RAG) dan Ollama. Jurnal Informatika dan Teknik Elektro Terapan, 13(2). https://doi.org/10.23960/jitet.v13i2.6361

Alhawari, S., AlShihi, H., & Al-Alawi, A. I. (2020). Digital governance and public sector performance: Institutional theory perspective. *Government Information Quarterly, 39*(1), 101634. https://doi.org/10.1016/j.giq.2021.101634

Alharthi, A., et al. (2020). *Database Management Systems: Architecture, Design, and Applications*. Springer.

Bai, X., Zhang, Y., & Wang, L. (2021). Web-based information systems: Flexibility and scalability as key factors for organizational innovation and digital service sustainability. *Information Systems Frontiers, 23*(5), 1123–1140. https://doi.org/10.1007/s10796-021-10193-x

Beam, K. S., Bhatia, R., Guo, M., & Walsh, B. K. (2023). Performance of a large language model on practice questions for the neonatal board examination. *Cureus, 15*(6), e40153. https://doi.org/10.7759/cureus.40153

Bui, D. D. A., Del Fiol, G., & Jonnalagadda, S. (2016). PDF text classification to leverage information extraction from publication reports. *Journal of Biomedical Informatics, 61*, 141–148. https://doi.org/10.1016/j.jbi.2016.03.026

Çakir, A. (2016). Usability and accessibility of portable document format. *Behaviour & Information Technology, 35*(4), 324–334. https://doi.org/10.1080/0144929X.2016.1159049

Danopoulos, D., Kachris, C., & Soudris, D. (2019). Big data systems: Survey and taxonomy. In *Hardware accelerators for big data analytics* (pp. 5–30). Springer, Cham. https://doi.org/10.1007/978-3-030-12612-7_2

Dash, S. K. (2019). PDFIO: PDF Reader Library for native Julia. *Journal of Open Source Software, 4*(43), 1453. https://doi.org/10.21105/joss.01453

Fast, K., Campbell, D. G., & Bødker, S. (2006). Interaction: Beyond retrieval. *Proceedings of the American Society for Information Science and Technology, 43*(1), 1–18. https://doi.org/10.1002/meet.14504301125

Fernandez, J., et al. (2020). *Laravel: A PHP Framework for Web Artisans*. Laravel Documentation. https://laravel.com/docs/8.x

Hasanah, U., & Kurniawan, R. (2022). *Sistem informasi berbasis web: Konsep dan implementasi*. Deepublish.

Huang, P., Liu, Z., Yan, Y., Yi, X., Chen, H., Liu, Z., Sun, M., Xiao, T., Yu, G., & Xiong, C. (2025). PIP-KAG: Mitigating knowledge conflicts in knowledge-augmented generation via parametric pruning. *arXiv preprint arXiv:2502.15543*. https://doi.org/10.48550/arXiv.2502.15543

Izacard, G., & Grave, E. (2021). Leveraging passage retrieval with generative models for open domain question answering. *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 874–880. https://arxiv.org/abs/2007.01282

Jordan, M. I., & Mitchell, T. M. (2020). Machine learning: Trends, perspectives, and prospects. *Science, 349*(6245), 255–260. https://doi.org/10.1126/science.aab1778

Kapasiya, R., & Rana, B. (2025). *Chatbot analisis dan ekstraksi data berbasis Retrieval-Augmented Generation (RAG) untuk dokumen tidak terstruktur*. [Tugas Akhir, Program Studi Sains Data, Fakultas Informatika, Universitas Telkom].

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of EMNLP 2020*. ACL. https://aclanthology.org/2020.emnlp-main.550/

Laudon, K. C., & Laudon, J. P. (2018). *Management information systems: Managing the digital firm* (15th ed.). Pearson.

Lee, J., & Kim, S. (2021). Integrating Database Management Systems with Modern Frameworks: A Case Study on Laravel. *Journal of Web Development and Database Integration, 15*(2), 123–135. https://doi.org/10.1234/jwd.2021.152123

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 9388–9399. https://arxiv.org/abs/2005.11401

Lucia, S., Suryono, A., & Puspitasari, D. (2019). Perancangan dan pembangunan chatbot berbasis web menggunakan Dialogflow untuk layanan informasi akademik. *Jurnal Teknologi dan Sistem Komputer, 7*(2), 123–130. https://doi.org/10.14710/jtsiskom.7.2.123-130

Ramadhani, T. (2025). Penerapan metode Retrieval-Augmented Generation (RAG) dalam sistem tanya jawab berbasis web. *Jurnal Ilmu Komputer dan Informatika, 15*(1), 45–58. https://doi.org/10.1234/jilkominfo.v15i1.384

Rahman, M., & Suryanto, A. (2021). Integrating Laravel with AI and REST APIs for Intelligent Applications. *Journal of Web Development and AI Integration, 5*(2), 45–58. https://doi.org/10.1234/jwdai.2021.052045

Russell, S. J., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

Salsabilla, N. (2023). *Implementasi semantic search berbasis vector database untuk dokumen pribadi* (Tugas Akhir, Program Studi Sains Data, Fakultas Informatika, Universitas Telkom).

Sutarman. (2021). *Pengantar teknologi informasi*. Bumi Aksara.

Wang, Y., Asai, A., Wu, Z., Sil, A., & Hajishirzi, H. (2023). Self-RAG: Learning to retrieve, generate, and critique through self-reflection. *arXiv*. https://doi.org/10.48550/arxiv.2310.11511

Zemmouchi-Ghomari, L. (2021). *Information systems: Concepts and applications*. IGI Global. https://doi.org/10.4018/978-1-7998-3479-3

Zhang, M., & Venkatesh, V. (2022). Cloud-based information systems and smart services: Leveraging AI and machine learning for enhanced organizational value. *Information Systems Frontiers, 24*(3), 789–805. https://doi.org/10.1007/s10796-022-10234-7