QUEROVA

AI-Powered Document Intelligence Platform

A Comprehensive RAG-Based Question-Answering System

Course: SE 383 01 - Python Programming

Instructor: Assoc. Prof. Dr. Önder Tombuş

Project Team:

Furkan Aksoy – 210706029

Emre Sarı – 220706304

İrem Özbilgin – 220706015

Maltepe University

Faculty of Engineering and Natural Sciences

Software Engineering Department

January 2026

Table Of Contents

1. Executive Summary

Querova represents a significant advancement in document intelligence technology, leveraging state-of-the-art artificial intelligence to transform how users interact with and extract information from complex documents. This project addresses the growing need for intelligent document processing systems in academic, business, and research environments where efficient information retrieval from large document collections is critical.

The platform implements Retrieval-Augmented Generation (RAG) technology, combining the power of large language models with semantic search capabilities to provide accurate, contextualized answers with verifiable source citations. Unlike traditional keyword-based search systems or standalone language models, Querova grounds its responses in actual document content, ensuring factual accuracy and enabling users to verify claims through direct source references.

Our implementation supports multiple document formats including PDF, DOCX, and plain text files, with intelligent chunking strategies that preserve document structure and context. The system processes documents through a sophisticated pipeline involving text extraction, semantic embedding generation, vector storage, and retrieval mechanisms optimized for accuracy and performance. The platform supports four distinct question types: open-ended inquiries for detailed explanations, multiple-choice questions for structured assessment, true/false statements for fact verification, and short-answer questions for concise information retrieval.

A key innovation of Querova is its advanced source verification system, which implements three-tier confidence scoring: exact matches for verbatim quotes, paraphrase detection for semantic equivalence, and inference marking for derived conclusions. This transparency enables users to assess answer reliability and trace information back to original sources, a critical requirement for academic research, legal review, and compliance auditing.

The technical architecture demonstrates sophisticated integration of modern technologies including FastAPI for high-performance backend services, ChromaDB for vector storage and similarity search, Google Gemini for language understanding and generation, and React with advanced UI frameworks for responsive user experience. The system achieves processing times under 25 seconds per query while maintaining high accuracy through careful prompt engineering and retrieval optimization.

This report documents the complete development lifecycle from initial conception through final deployment, detailing architectural decisions, implementation challenges, performance optimizations, and lessons learned. Our results demonstrate the viability of RAG-based systems for production environments, with clear paths identified for future enhancements including multi-language support, advanced analytics, and enterprise-scale deployment capabilities.

2. Introduction

2.1 Background and Motivation

The exponential growth of digital documentation in academic institutions, corporate environments, and government agencies has created an urgent need for intelligent document processing systems. Traditional methods of information extraction, relying on manual searching or simple keyword matching, have become increasingly inadequate for handling the volume and complexity of modern document collections. Researchers, students, and professionals routinely face the challenge of synthesizing information from hundreds or thousands of pages across multiple documents, a task that is both time-consuming and prone to missing critical details.

Recent advances in artificial intelligence, particularly in natural language processing and large language models, have opened new possibilities for automated document understanding. However, pure language model approaches suffer from hallucination problems, where models generate plausible-sounding but factually incorrect information. Additionally, these models lack transparency in their reasoning process, making it difficult for users to verify the accuracy of provided information or trace answers back to source material.

Retrieval-Augmented Generation emerged as a solution to these limitations by combining information retrieval with language generation. This hybrid approach retrieves relevant information from a knowledge base before generating responses, grounding the AI's output in factual document content while maintaining the natural language understanding capabilities of modern language models. The RAG paradigm has demonstrated significant improvements in factual accuracy and verifiability compared to standalone language models.

2.2 Project Objectives

The primary objective of Querova is to develop a production-ready document intelligence platform that combines the benefits of semantic search, natural language processing, and user-friendly interface design. The system must handle real-world document formats encountered in academic and professional settings, process documents efficiently regardless of size or complexity, and provide accurate answers with transparent source attribution.

Specific technical objectives include:

- Implement a robust document processing pipeline supporting PDF, DOCX, and TXT formats with intelligent text extraction that preserves document structure and metadata
- Design and implement a semantic chunking strategy that balances context preservation with retrieval accuracy, optimizing chunk size and overlap parameters for various document types
- Integrate state-of-the-art language models for both embedding generation and answer synthesis, with careful prompt engineering to maximize answer quality and minimize hallucination
- Develop a source verification system that categorizes evidence quality and provides confidence scoring for generated answers, enabling users to assess reliability

- Create an intuitive user interface that supports multiple interaction patterns including manual question input and batch processing via JSON uploads
- Achieve query response times under 25 seconds while maintaining high accuracy and comprehensive source citation

2.3 Scope and Limitations

This implementation focuses on English and Turkish language documents, with architecture designed to support additional languages in future iterations. The system targets document sizes up to 10MB and supports common business document formats. While the platform handles complex documents including multi-column layouts, the current text extraction pipeline processes tabular data primarily as sequential text. Consequently, queries requiring precise row-column relationship understanding in complex tables may yield lower accuracy compared to unstructured text. Furthermore, highly specialized formats such as mathematical notation or complex chemical diagrams require additional preprocessing steps not included in this iteration.

The current version implements a single-user deployment model suitable for individual researchers or small teams. Future enhancements will address multi-user scenarios, document access control, and collaborative features. The system assumes documents contain primarily textual content; extensive image-based content requires optical character recognition preprocessing not included in the current implementation.

3. Literature Review

Recent progress in natural language processing has been largely driven by Transformer-based architectures, which replaced recurrence with attention mechanisms and enabled efficient scaling to very large models (Vaswani et al., 2017). Building on this foundation, large language models (LLMs) trained on broad web-scale corpora have demonstrated strong zero-shot and few-shot capabilities across many tasks, including question answering and reasoning-like generation (Brown et al., 2020). Despite these advances, a widely documented limitation is that LLMs can produce fluent but ungrounded or incorrect statements—often referred to as hallucinations—especially when the required information is not reliably represented in the model's parameters (Ji et al., 2022). This challenge motivates system designs that explicitly connect generation with external evidence and verifiable sources.

A central line of work addressing this reliability gap is Retrieval-Augmented Generation (RAG), which combines a retriever that selects relevant passages from a knowledge corpus with a generator that produces an answer conditioned on those retrieved contexts (Lewis et al., 2020). In this paradigm, the system does not rely solely on the parametric memory of the LLM; instead, it "grounds" responses in retrieved documents that can be shown to the user as supporting evidence. RAG has been demonstrated as an effective approach for knowledge-intensive tasks, improving factual accuracy while also enabling transparency via evidence-based answering (Lewis et al., 2020). The broader open-domain question answering literature similarly emphasizes the importance of retrieval as a first stage for selecting candidate contexts before performing answer generation (Karpukhin et al., 2020).

Retrieval quality is strongly influenced by how documents are represented. Classic information retrieval systems rely on sparse lexical matching, where ranking is driven by

term overlap between queries and documents. One of the best-known probabilistic sparse ranking functions is BM25, which remains a robust baseline across domains due to its efficiency and strong performance under distribution shifts (Robertson & Zaragoza, 2009). However, lexical methods struggle when relevant passages are semantically aligned but use different vocabulary. This limitation has led to widespread adoption of dense retrieval, where both queries and passages are mapped into a shared embedding space and similarity is computed using vector distance metrics (Karpukhin et al., 2020). Dense Passage Retrieval (DPR), for example, uses a dual-encoder architecture to learn dense representations and significantly improves retrieval accuracy on open-domain QA benchmarks compared to sparse baselines in many settings (Karpukhin et al., 2020).

Dense retrieval pipelines depend on strong sentence and passage embedding models. Contextual encoders based on Transformer pretraining—such as BERT—provide high-quality representations for downstream tasks (Devlin et al., 2019). Sentence-BERT (SBERT) adapts this approach to produce semantically meaningful sentence embeddings suitable for similarity search and clustering, enabling efficient retrieval in embedding space (Reimers & Gurevych, 2019). More recent contrastive learning approaches, such as SimCSE, further improve embedding quality by optimizing objectives that bring semantically related texts closer while pushing unrelated texts apart, achieving strong performance on semantic similarity benchmarks (Gao, Yao, & Chen, 2021). For practical RAG systems, these embedding methods support semantic search over heterogeneous document corpora, which is essential when users phrase questions differently than the source material.

Because no single retrieval strategy is universally optimal, a large body of work investigates hybrid retrieval and multi-stage ranking pipelines. Benchmarking studies highlight that BM25 is often a strong baseline, while neural retrievers and late-interaction models can improve results but may be more sensitive to domain shift and computational constraints (Thakur et al., 2021). In many modern systems, a first-stage retriever (sparse or dense) is followed by a more expensive re-ranking step that refines the candidate list. This two-stage architecture is particularly common in question answering, where retrieving a broad candidate set improves recall and re-ranking improves precision.

Neural re-ranking commonly uses cross-encoder architectures that jointly encode the query and passage, enabling fine-grained token-level interactions at the cost of higher computation. A representative example is BERT-based passage re-ranking, which demonstrated strong gains on standard ranking benchmarks by scoring candidate passages with a cross-encoder (Nogueira & Cho, 2019). Late-interaction approaches, such as ColBERT, offer a middle ground by retaining token-level matching while improving efficiency compared to full cross-encoders (Khattab & Zaharia, 2020). In a RAG setting, re-ranking is especially valuable because the generator's output quality is highly dependent on the relevance and faithfulness of the retrieved context; improving the top-k evidence set can directly reduce unsupported claims and improve answer completeness.

At the infrastructure level, dense retrieval requires efficient indexing and similarity search over high-dimensional vectors. Approximate nearest neighbor (ANN) methods are widely used to make retrieval scalable for large corpora. FAISS is a prominent library that enables billion-scale similarity search through optimized GPU and approximate indexing strategies (Johnson, Douze, & Jégou, 2017). Among ANN graph-based methods, Hierarchical Navigable Small World (HNSW) graphs are well known for providing strong recall-efficiency trade-offs and robustness in practical nearest neighbor search, and they have been formalized and evaluated in depth in the literature (Malkov & Yashunin, 2020). Modern

vector databases frequently employ such ANN structures to support low-latency semantic search and filtering operations, which are essential for interactive question-answering systems.

Another important consideration is that simply expanding the LLM context window does not fully solve grounding or evidence utilization. Studies on long-context behavior show that models can underutilize relevant information when it appears in the middle of lengthy contexts, a phenomenon commonly described as “lost in the middle,” which can degrade performance even when supporting evidence is present in the prompt (Liu et al., 2024). This finding supports the design choice of retrieval plus careful context construction (e.g., chunking, top-k selection, and re-ranking) rather than relying on unstructured long prompts. It also motivates chunking strategies that balance semantic coherence with retrieval granularity, so the system can surface the most relevant evidence in a compact form.

Because hallucination remains a core risk in generative systems, recent work focuses not only on grounding but also on citation and verifiability. Surveys of hallucination emphasize both detection and mitigation techniques and highlight retrieval grounding as a key practical method for reducing unfaithful generation (Ji et al., 2022). Beyond simply retrieving context, research has proposed benchmarks and methods to evaluate whether generated statements are supported by citations. For example, ALCE formalizes evaluation of citation quality and supports systematic comparison of citation-aware generation strategies (Gao, Yen, Yu, & Chen, 2023). Related work on quote- or evidence-supported answering, such as systems trained to provide verified supporting spans, further demonstrates that evidence presentation can improve user trust and enable better human assessment of correctness (Menick et al., 2022).

Finally, robust evaluation methodologies are critical for RAG systems because both retrieval and generation contribute to end-to-end quality. Traditional evaluation often relies on reference answers, but many realistic use cases lack gold references. RAGAs proposes a reference-free evaluation framework that estimates key aspects of RAG performance, including context relevance and faithfulness, enabling faster iteration and more structured debugging of retrieval and generation components (Es et al., 2024). In practice, these evaluation perspectives align with engineering needs: improving retrieval recall and ranking quality, ensuring the generator uses evidence faithfully, and surfacing citations that allow users to verify claims. Overall, the literature supports a pipeline-oriented approach—document processing and chunking, embedding-based retrieval, optional hybrid ranking and re-ranking, evidence-aware generation, and systematic evaluation—as a principled foundation for systems like Querova.

4. System Architecture

4.1 Overall Architecture Design

Querova employs a modern microservices-inspired architecture with clear separation between frontend presentation, backend logic, and data persistence layers. This design enables independent scaling of components, facilitates testing and debugging, and allows for future enhancement without requiring system-wide modifications. The architecture follows RESTful API principles, ensuring clean interfaces between components and enabling potential integration with external systems.

The frontend layer implements a single-page application using React 18, providing responsive user interfaces with real-time feedback during document processing and query execution. The backend exposes a comprehensive REST API through FastAPI, handling document ingestion, vector storage operations, and query processing. Data persistence utilizes ChromaDB for vector storage and local filesystem for document uploads, with all components designed for containerization and cloud deployment.

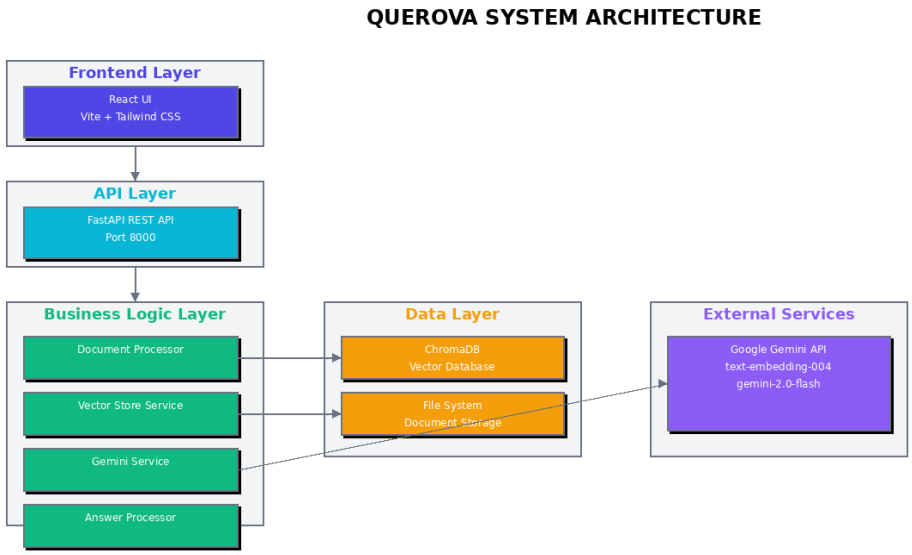

Figure 1: Querova System Architecture
Layered architecture showing component interactions

Figure 1: *Querova System Architecture*

Figure 1 illustrates the layered architecture of Querova, organized into four primary tiers. The Frontend Layer implements the user interface using React with Vite and Tailwind CSS, providing responsive interactions and real-time feedback. The API Layer, built with FastAPI, exposes RESTful endpoints for document management and question processing. The Business Logic Layer contains core services including the Document Processor for text extraction and chunking, Vector Store Service for ChromaDB integration, Gemini Service for AI model interaction, and Answer Processor for source verification. The Data Layer manages persistence through ChromaDB for vector storage and the local filesystem for document storage. Additionally, the architecture integrates with external services, specifically the Google Gemini API for embeddings and text generation. This separation of concerns enables maintainability, testability, and scalability of the system.

4.2 Component Overview

Document Processing Service: This service handles the ingestion pipeline, accepting documents through multipart upload, performing format-specific text extraction, and implementing intelligent chunking strategies. The processor preserves document metadata including page numbers, section identifiers, and structural information that enables precise

source citation. Error handling includes validation of file sizes, format verification, and graceful degradation when encountering corrupted or unsupported content.

Vector Store Service: ChromaDB provides the foundation for semantic search, storing document chunk embeddings along with associated metadata. The service implements optimized query methods supporting top-k retrieval with configurable relevance thresholds. The vector store maintains document-chunk relationships, enabling efficient document deletion and supporting future features like incremental updates and version tracking.

Language Model Integration: The Gemini service abstracts interactions with Google's language models, handling both embedding generation for document chunks and query processing for answer synthesis. The service implements retry logic for API reliability, manages rate limiting, and provides safety filtering to ensure appropriate content generation. Prompt templates are maintained separately, allowing refinement without code changes.

Answer Processing Service: This component implements the source verification pipeline, analyzing generated answers to extract and validate source citations. The processor employs fuzzy matching algorithms to align generated text with source chunks, categorizes matches as exact quotes, paraphrases, or inferences, and calculates confidence scores based on match quality and retrieval relevance scores.

4.3 Data Flow and Processing Pipeline

The document ingestion pipeline begins when users upload files through the web interface. The frontend performs initial validation of file size and format before transmitting data to the backend API endpoint. The document processor saves uploaded files, extracts text content using format-specific libraries, and segments the text into overlapping chunks optimized for retrieval. Each chunk is enriched with metadata including source document ID, page number, chunk index, and positional information.

The embedding generation phase transforms text chunks into dense vector representations capturing semantic meaning. The Gemini service batches embedding requests for efficiency, handling API rate limits and retrying failures. Generated embeddings are stored in ChromaDB alongside chunk text and metadata, with indexing optimized for cosine similarity search.

Query processing follows a multi-stage pipeline beginning with question embedding generation. The vector store retrieves the top-k most relevant chunks based on embedding similarity, applying configurable relevance thresholds to filter low-quality matches. Retrieved chunks are formatted into a context prompt along with the original question, type-specific instructions, and examples guiding the language model's response format.

The language model generates an answer grounded in the provided context, attempting to cite sources using the [Kaynak X] format as instructed. The answer processing service analyzes the generated text, extracting citation markers, matching them to source chunks, and validating quote accuracy. The final response package includes the generated answer, verified sources with confidence scores, processing time metrics, and overall confidence assessment.

4.4 Data Model and Database Schema

The Querova data model is documented as a logical (conceptual) normalized relational schema designed to support efficient document storage, vector-based retrieval, and comprehensive query tracking. In the current implementation, this logical schema is realized through a vector store (ChromaDB) with structured metadata and application-level records; an optional relational database can be introduced for operational logging and analytics without changing the retrieval layer. The storage architecture balances normalization for data integrity with pragmatic denormalization for query performance, particularly in high-frequency retrieval operations.

The schema centers around the document-chunk-embedding triad, which forms the foundation of the retrieval-augmented generation pipeline. Documents are decomposed into chunks for processing, with each chunk receiving a corresponding embedding vector that enables semantic similarity search. This design allows for granular source attribution while maintaining document-level organization and metadata.

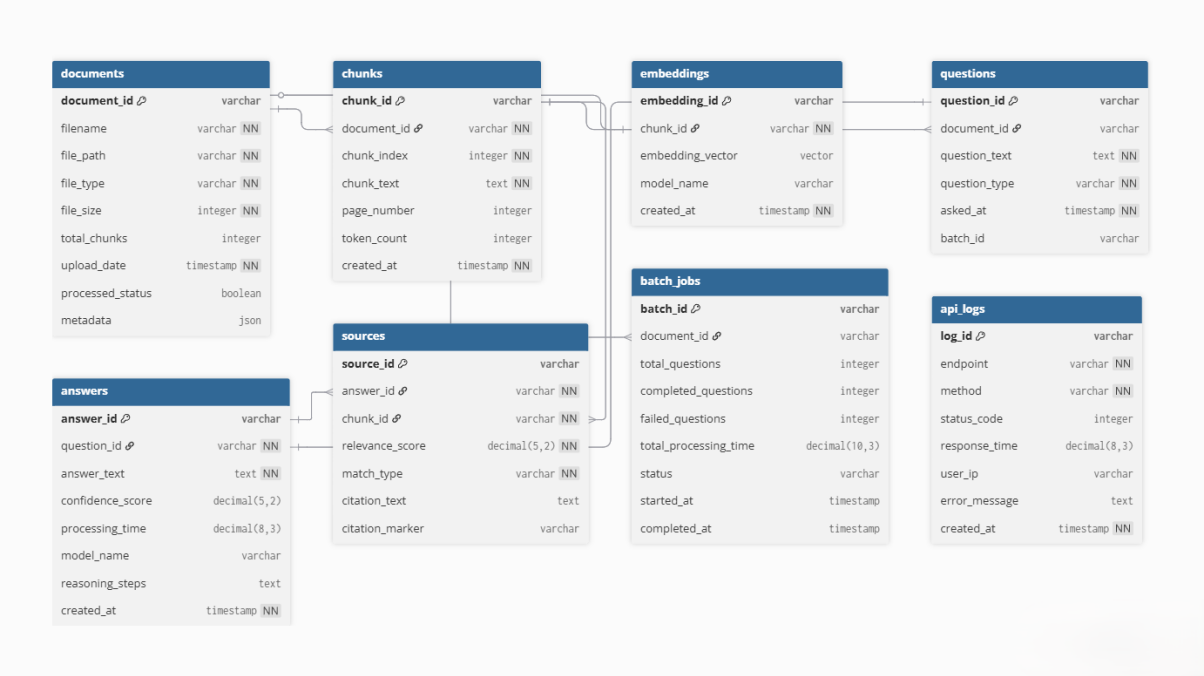

Figure 2: *ERD shows logical entities; physically realized via ChromaDB collections + metadata*

Figure 2 presents the complete entity-relationship model for Querova's logical storage schema. The diagram illustrates eight primary entities and their relationships. The Documents entity serves as the root, storing metadata including filename, file path, file type (PDF, DOCX, or TXT), file size, and processing status. Each document maintains a one-to-many relationship with Chunks, which represent the segmented portions of document text optimized for retrieval. The chunk entity includes the actual text content, page number for citation purposes, token count for prompt management, and an index indicating position within the source document.

The Embeddings entity maintains a one-to-one relationship with Chunks, storing the vector representations generated by Google's text-embedding-004 model. In practice, these vectors are persisted within the vector database alongside chunk identifiers and metadata, while keeping the Embeddings entity conceptually separate supports re-embedding with different models without modifying chunk text content.

The Questions entity records user queries with associated metadata including question type classification (open-ended, multiple-choice, true-false, or short-answer), timestamp, and optional batch identifier for grouped processing. Each question links to exactly one Answer through a one-to-one relationship, with answers storing the generated response text, confidence scores, processing time metrics, and a structured explanation (when enabled) derived from the model output.

The Sources entity implements a many-to-many relationship between answers and chunks, representing the retrieved context that supported answer generation. Each source record includes relevance scores derived from vector similarity calculations, match type classification (exact match, paraphrase, or inference), citation text extracted from the chunk, and citation markers (e.g., [Kaynak 1]) used in the answer. This granular source tracking enables the transparency features that distinguish Querova from simpler question-answering systems. Operationally, these source links can be stored as answer-linked metadata records or in a lightweight relational table used for traceability and reporting.

Supporting entities include Batch_Jobs for managing bulk question processing with status tracking and performance metrics, and API_Logs for comprehensive request logging enabling monitoring, debugging, and usage analytics. The schema employs appropriate indexes on frequently queried fields such as document filenames, question timestamps, and batch identifiers, optimizing query performance for interactive use cases. Foreign key constraints maintain referential integrity, with cascade delete rules ensuring that removing a document properly cleans up associated chunks, embeddings, questions, and answers.

This design supports current functionality while providing extensibility for planned features. The normalized structure enables efficient updates and maintains data consistency, while strategic denormalization in high-read entities like chunks and embeddings ensures query performance remains acceptable. The schema accommodates future enhancements including multi-user support through user entity addition, document versioning through temporal tables, and advanced analytics through aggregation-optimized views.

5. System Use Cases

The Querova system provides comprehensive functionality for document-based question answering. The primary actor, the User (researcher or student), interacts with the system through multiple use cases designed to support various workflows from document management to answer verification.

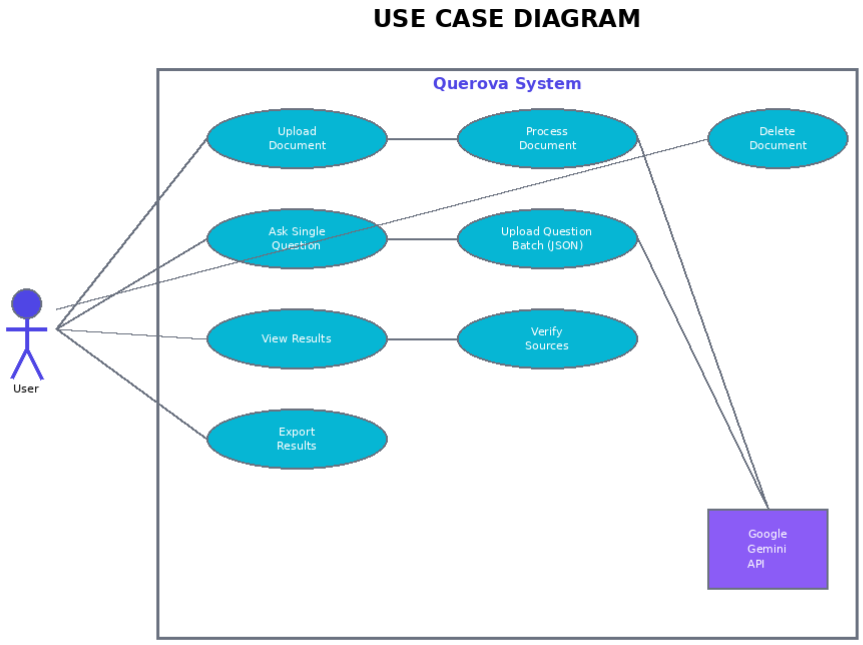

Figure 2: Use Case Diagram
Showing user interactions with the system

Figure 3: *Use Case Diagram*

Figure 3 presents the complete use case model for Querova. The diagram shows eight primary use cases within the system boundary. The Upload Document use case initiates the Process Document use case, which interacts with the external Google Gemini API for embedding generation. Users can Ask Single Question or Upload Question Batch in JSON format, both of which leverage the Gemini API for natural language processing. The View Results use case connects to Verify Sources, enabling users to trace answer origins. The Delete Document use case allows users to manage their document collection. The Export Results use case facilitates sharing and archival of query results. This use case model demonstrates the system's flexibility in supporting both individual and batch processing workflows while maintaining transparency through source verification. The integration with the Google Gemini API as an external system highlights the dependence on cloud-based AI services for core functionality.

6. Question Answering Process Flow

The question answering process in Querova follows a sophisticated multi-phase pipeline designed to ensure accurate, well-sourced responses. Understanding this flow is essential to appreciating how the system maintains high quality while providing rapid response times.

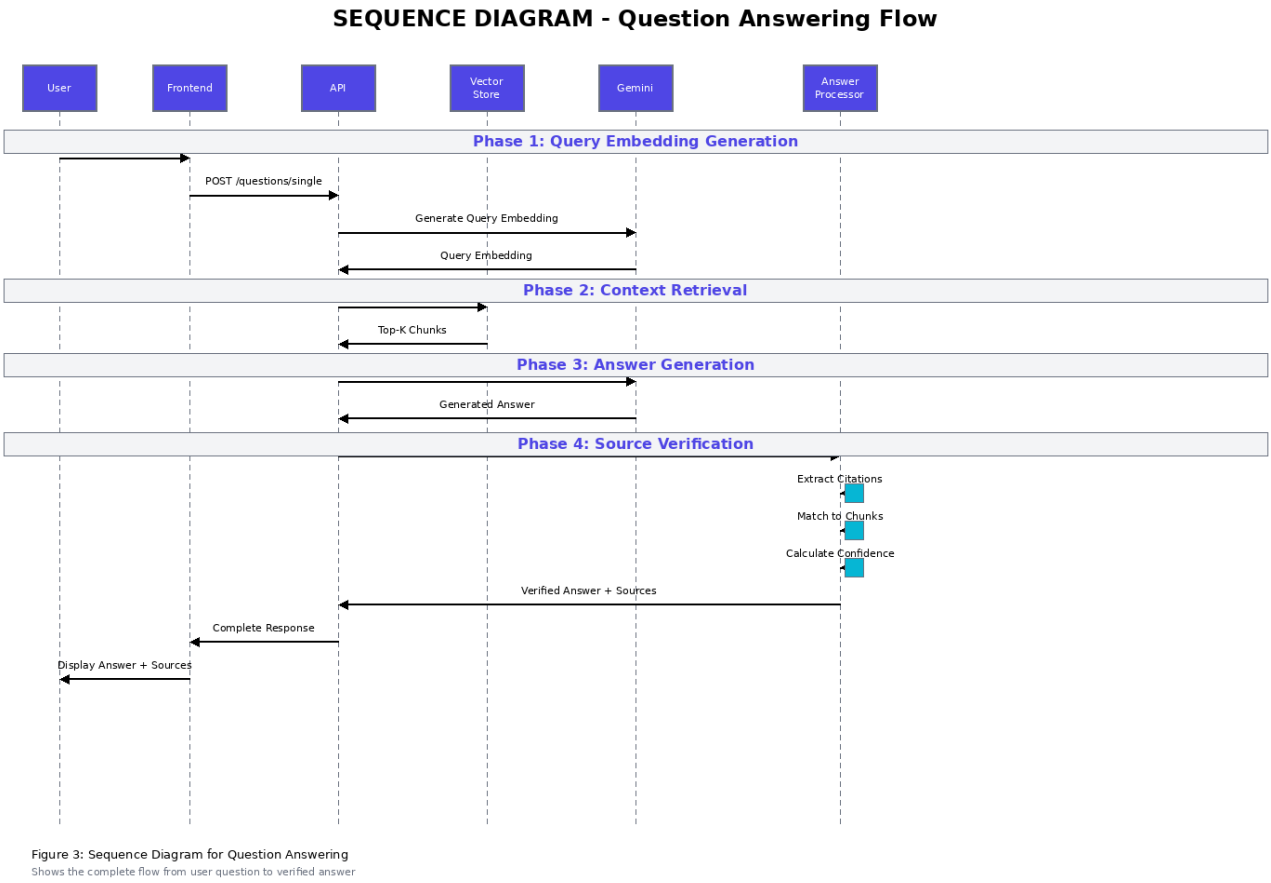

Figure 4: *Sequence Diagram for Question Answering*

Figure 4 illustrates the complete interaction flow when a user submits a question. The process begins when the User submits a question through the Frontend interface. The Frontend forwards this request to the API layer via a POST request to the questions endpoint. The sequence then proceeds through four distinct phases, each shown with clear separation in the diagram.

Phase 1, Query Embedding Generation, involves the API requesting the Gemini service to generate a vector embedding representing the semantic meaning of the user's question. This embedding enables semantic search rather than simple keyword matching. Phase 2, Context Retrieval, uses the generated embedding to search the Vector Store for the most relevant document chunks. The Vector Store returns the top-K chunks ranked by similarity. Phase 3, Answer Generation, combines the original question with the retrieved context chunks and sends them to the Gemini service, which generates a comprehensive answer grounded in the provided context.

Phase 4, Source Verification, represents a key innovation where the Answer Processor performs three critical sub-tasks: extracting citation markers from the generated text, matching these citations to the retrieved source chunks, and calculating confidence scores based on match quality. Finally, the verified answer with source attributions is returned through the API to the Frontend, which displays the results to the User with clear source references and confidence indicators. This sequence ensures that every answer is not only accurate but also traceable to its source material, maintaining the transparency essential for academic and professional use.

7. Technical Implementation

7.1 Backend Implementation

The backend infrastructure is built on FastAPI, a modern Python web framework chosen for its exceptional performance, automatic API documentation generation, and native support for asynchronous operations. FastAPI's Pydantic integration ensures robust request validation and type safety throughout the application. The framework's automatic OpenAPI schema generation provides interactive documentation accessible at the /docs endpoint, facilitating development and integration testing.

Document processing leverages specialized libraries for each supported format. PyMuPDF handles PDF extraction, providing access to text content, page information, and document metadata. The library efficiently processes complex PDFs including multi-column layouts and documents with embedded fonts. DOCX processing utilizes python-docx, extracting both paragraph text and table content while preserving document structure. Plain text files are handled with encoding detection and fallback strategies ensuring reliable processing of diverse character sets.

The chunking strategy currently implements a sliding window approach with configurable chunk size (default 500 tokens) and overlap (50 tokens). While this baseline strategy ensures consistent retrieval units, it lays the groundwork for future implementations of semantic or recursive chunking strategies that would dynamically adjust boundaries based on paragraph structure and semantic completeness.

Example chunking configuration:

```
CHUNK_SIZE = 500  # tokens per chunk

CHUNK_OVERLAP = 50  # token overlap between chunks

MAX_FILE_SIZE = 10485760  # 10MB limit
```

7.2 Vector Database Integration

ChromaDB serves as the vector database, selected for its simplicity, performance, and Python-native design. The database stores document embeddings as high-dimensional vectors, typically 768 dimensions for the text-embedding-004 model. ChromaDB implements HNSW (Hierarchical Navigable Small World) indexing for efficient approximate nearest neighbor search, achieving sub-linear query time complexity even for large document collections.

The integration layer abstracts ChromaDB operations, providing methods for chunk addition, similarity search, document deletion, and collection statistics. The search implementation supports filtering by document ID, enabling targeted retrieval from specific documents, and accepts relevance score thresholds for quality control. Distance metrics are normalized to similarity scores on a 0-1 scale, providing intuitive confidence values for retrieved chunks.

Metadata storage within ChromaDB enables rich filtering and attribution. Each vector entry includes the chunk text for display, document ID for relationship tracking, page number for citation, chunk index for ordering, and custom metadata fields for extensibility. This design

supports future enhancements like temporal filtering, document category filtering, and access control at the chunk level.

7.3 Language Model Integration

Google Gemini provides both embedding generation and text generation capabilities. The text-embedding-004 model generates dense vector representations capturing semantic meaning beyond keyword matching. These embeddings enable the system to match questions with relevant content even when exact terminology differs, a critical capability for natural language queries.

Answer generation employs Gemini 2.0 Flash, a model balancing performance with cost efficiency. The service constructs type-specific prompts that guide the model to produce appropriate answer formats for each question type. For open-ended questions, prompts emphasize comprehensive explanations with source citations. Multiple-choice prompts request explicit answer selection followed by justification. True/false prompts demand binary classification with supporting evidence. Short-answer prompts enforce concise responses focusing on essential information.

Safety configuration prevents inappropriate content generation while minimizing false positives. The implementation sets permissive safety thresholds for categories like harassment and dangerous content, necessary because technical and medical documents often trigger overly conservative filters. However, the system maintains appropriate safeguards for truly harmful content through careful prompt design and output validation.

Retry logic handles transient API failures and rate limiting. The service implements exponential backoff with jitter, attempting up to three retries for failed requests. Logging captures detailed error information including API response codes, retry attempts, and final outcomes, facilitating debugging and monitoring in production environments.

7.4 Frontend Implementation

The user interface leverages React 18's concurrent rendering features for responsive interactions even during expensive operations like large document uploads. The application employs a component-based architecture with clear separation of concerns: presentation components handle visual rendering, container components manage state and business logic, and service modules abstract API communication.

Tailwind CSS provides utility-first styling with custom theme extensions implementing the glassmorphism aesthetic. The design system defines consistent spacing, typography, and color palettes applied throughout the interface. Responsive breakpoints ensure optimal layouts across device sizes from mobile phones to desktop monitors.

Animation and micro-interactions enhance user experience through Framer Motion integration. Document uploads display progress indicators with smooth transitions. Question processing shows loading animations while awaiting backend responses. Result displays animate in with staggered timing for visual polish. These enhancements provide feedback reassuring users that the system is processing their requests.

State management utilizes React hooks for local component state and context for shared application state. The implementation avoids heavy state management libraries, keeping the

codebase simple and maintainable. API communication abstracts fetch operations behind service functions that handle request formatting, error handling, and response parsing, isolating API changes from component code.

8. Key Features and Innovations

8.1 Source Verification System

The source verification system represents a significant innovation beyond basic RAG implementations. Rather than simply displaying retrieved chunks, the system actively analyzes generated answers to identify and validate source citations. The processor employs multiple strategies to extract verifiable evidence from language model outputs.

Citation marker detection identifies explicit source references in generated text using the [Kaynak X] format. When markers are present, the system maps them to specific chunks from the retrieval results, extracting relevant quotes that support the answer's claims. This explicit citation approach provides transparency and enables readers to verify information directly.

In cases where explicit citations are absent, the system employs fuzzy matching algorithms to align answer content with source chunks. The implementation uses sequence matching to identify similar text spans, calculating confidence scores based on match quality. Matches are categorized as exact quotes when text aligns closely, paraphrases when semantic meaning matches despite different wording, or inferences when answers derive from source content without direct quotation.

Each source receives a confidence score incorporating multiple factors: the retrieval relevance score indicating how well the chunk matched the original query, the match quality score reflecting how closely answer text aligns with source text, and the evidence type indicating whether support is direct quotation or inference. These scores are aggregated into an overall answer confidence metric, helping users assess reliability.

8.2 Multi-Type Question Support

Supporting diverse question types enables Querova to serve various use cases from exploratory research to structured assessment. The implementation provides four distinct question types, each with tailored processing and response formatting.

Open-ended questions receive comprehensive treatment, with prompts encouraging detailed explanations, multiple source citations, and structured presentation. The language model is instructed to organize responses into logical paragraphs, provide examples when appropriate, and conclude with summaries. This format serves researchers seeking thorough understanding of complex topics.

Multiple-choice questions are processed with special handling for option parsing and answer extraction. The system automatically detects multiple-choice format in question text, extracts available options, and includes them in the language model prompt. Generated answers are required to explicitly state the selected option before providing justification, and the system employs regex patterns to extract the final answer for programmatic processing.

True/false questions demand binary classification with supporting evidence. Prompts instruct the model to begin responses with explicit declarations, followed by explanations grounded in source material. This format enables automated answer extraction while maintaining transparency in reasoning.

Short-answer questions emphasize conciseness, with prompts limiting responses to one or two sentences focusing on essential information. This format serves quick fact-checking and information lookup use cases where users need specific details without extensive explanation.

8.3 Batch Processing Capabilities

The batch processing feature enables efficient handling of multiple questions simultaneously, valuable for assessment scenarios, systematic document analysis, and research workflows. Users can upload JSON files containing question arrays, with each question processed through the complete RAG pipeline.

The implementation processes questions sequentially to manage API rate limits and resource consumption, but future enhancements could implement parallel processing with configurable concurrency limits. Progress tracking provides real-time feedback during batch execution, displaying completed questions, failures, and estimated remaining time.

Results are aggregated into comprehensive reports including per-question answers with source citations, processing time metrics, confidence scores, and verification status. The system also provides batch-level statistics: total processing time, success rate, average confidence, and failure analysis. These metrics enable assessment of system performance and identification of challenging question patterns.

8.4 Advanced User Interface

The user interface prioritizes clarity and efficiency while providing visual polish through modern design techniques. The glassmorphism aesthetic employs semi-transparent elements with backdrop blur, creating depth and visual hierarchy. Subtle animations and transitions provide feedback without distracting from content.

The document upload interface supports both click-to-select and drag-and-drop interactions, with visual feedback during file hovering and upload progress indicators showing real-time status. Upon completion, uploaded documents display with chunk count information and deletion options, enabling users to manage their document collection.

Question input offers dual modes: a text area for manual entry with real-time character counting, and JSON upload for batch processing with format validation. Tab switching provides clear mode selection while maintaining input state. Both modes include helpful hints and format examples, reducing user errors and accelerating onboarding.

Results display emphasizes source transparency through expandable cards showing question, answer, and supporting evidence. Each source card displays the matched text, relevance score, confidence assessment, and page reference. Users can expand sources to view additional context, enabling verification without leaving the interface. The implementation uses consistent color coding and iconography to convey verification status at a glance.

9. User Interface and System Demonstration

This section presents screenshots of the Querova system in operation, demonstrating the user interface design, interaction patterns, and result presentation capabilities. These visual examples illustrate how the theoretical concepts and technical implementations discussed in previous sections translate into a functional, user-friendly application.

9.1 Main User Interface

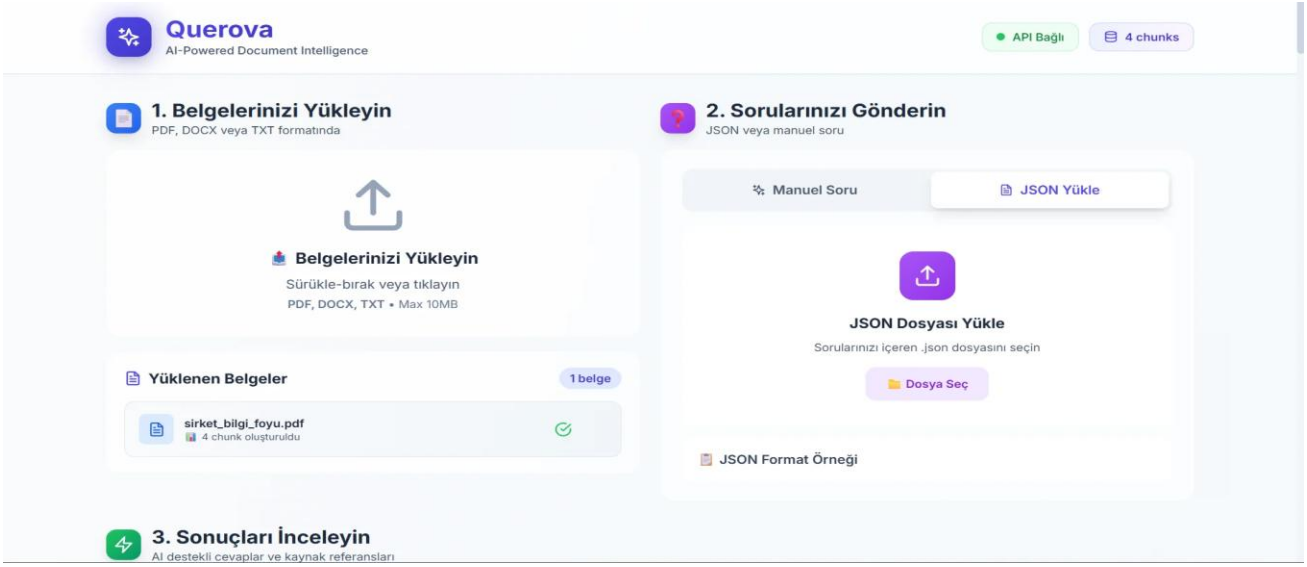

Figure 5: *Main User Interface - Document Upload and Question Input*

Figure 5 presents the main user interface of Querova, showcasing the clean, intuitive design that prioritizes usability while maintaining visual appeal. The interface is organized into three primary sections. The first section, 'Belgelerinizi Yükleyin' (Upload Your Documents), features a prominent drag-and-drop upload area with clear visual cues including an upload icon and supported file format indicators (PDF, DOCX, TXT with 10MB maximum size). Below this, the 'Yüklenen Belgeler' (Uploaded Documents) panel displays currently loaded documents with metadata including the filename 'sirket_bilgi_foyu.pdf' and processing status showing '4 chunk oluşturuldu' (4 chunks created), providing immediate feedback on document processing completion.

The second section, 'Sorularınızı Gönderin' (Submit Your Questions), offers two input methods through a tabbed interface. The 'Manuel Soru' (Manual Question) tab provides a text area for direct question entry, while the 'JSON Yükle' (Upload JSON) tab enables batch question processing. The purple 'JSON Dosyası Yükle' button with accompanying file format example demonstrates the system's support for automated workflows. At the top right of the interface, status indicators display system health with a green 'API Bağlı' (API Connected) badge and document collection statistics showing '4 chunks' currently available. The glassmorphism design elements, subtle shadows, and consistent color scheme (purple accents on white background) create a professional, modern aesthetic that aligns with contemporary web application standards.

9.2 Batch Processing Results Overview

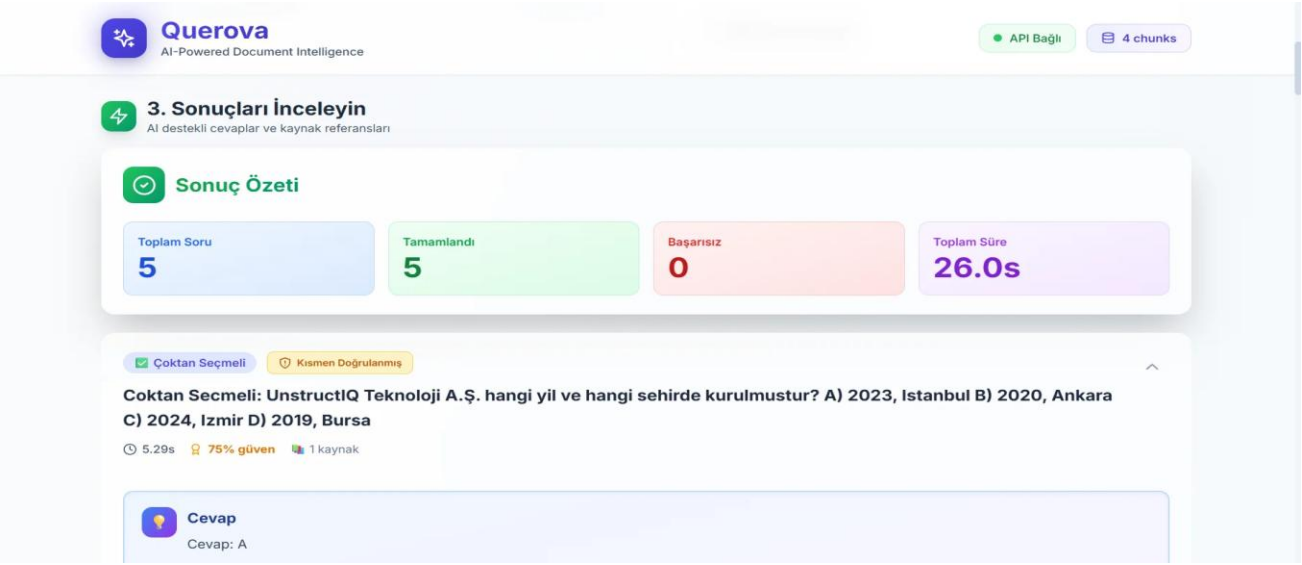

Figure 6: *Batch Processing Results Summary*

Figure 6 displays the results summary interface that appears after batch question processing completes. The 'Sonuç Özeti' (Results Summary) section presents key metrics in four color-coded cards using a traffic-light visualization scheme. The blue card shows 'Toplam Soru' (Total Questions) with the number 5, indicating the size of the processed batch. The green card displays 'Tamamlandı' (Completed) also showing 5, confirming successful processing of all questions. The red card labeled 'Başarısız' (Failed) displays 0, indicating no processing failures occurred. The purple card presents 'Toplam Süre' (Total Time) of 26.0 seconds, providing performance metrics that average approximately 5.2 seconds per question.

Below the summary cards, individual question results are displayed in expandable cards. The visible example shows a multiple-choice question tagged with 'Çoktan Seçmeli' (Multiple Choice) and 'Kısmen Doğrulanmış' (Partially Verified) badges. The question asks about UnstructIQ Teknoloji A.Ş.'s establishment year and location, presenting four options (A through D) with different year and city combinations. Processing metadata indicates this question took 5.29 seconds with a confidence score of 75 percent (displayed with a shield icon) and references 1 source (shown with a document icon). The answer preview shows 'Cevap: A' (Answer: A), demonstrating the system's ability to extract and present the selected option clearly. The expandable card design with collapse/expand controls (indicated by the chevron icon) allows users to manage information density, viewing summaries by default while accessing detailed explanations on demand.

9.3 Detailed Answer View with Reasoning

Figure 7: *Detailed Answer Display with Step-by-Step Reasoning*

Figure 7 illustrates the expanded view of an answer, revealing the comprehensive information provided for each processed question. This example shows a multiple-choice question about the company's core focus regarding data types, with the 'Doğrulanmış' (Verified) green badge indicating high-confidence source verification. The processing metadata shows 5.96 seconds execution time, 95 percent confidence score, and 3 source references, all displayed with intuitive icons for quick comprehension.

The answer section presents the system's response 'Cevap: B' (Answer: B), followed by a detailed explanation structured in two parts. The first section, 'Seçim Gerekçesi' (Selection Justification), explains why option B is correct by referencing the company's focus on unstructured data types including PDF, email, call center records, and social media content, with explicit source citations marked as [Kaynak 1] (Source 1). This demonstrates the system's ability to ground its reasoning in document content rather than making unsupported claims.

The second section, 'Diğer Seçeneklerin Neden Yanlış Olduğu' (Why Other Options Are Incorrect), systematically evaluates each alternative option. For option A, the explanation clarifies that while the company does handle structured data from financial tables, it is not exclusively focused on this type, citing [Kaynak 1]. For options C and D, the reasoning explains that the company works with data visible in documents and communication logs rather than only video/IoT sensor data, providing specific source citations [Kaynak 1] and [Kaynak 4]. This structured reasoning approach helps users understand not just what the answer is, but why alternatives were rejected, supporting educational use cases and building trust in the system's decision-making process.

9.4 Source Verification and Evidence Display

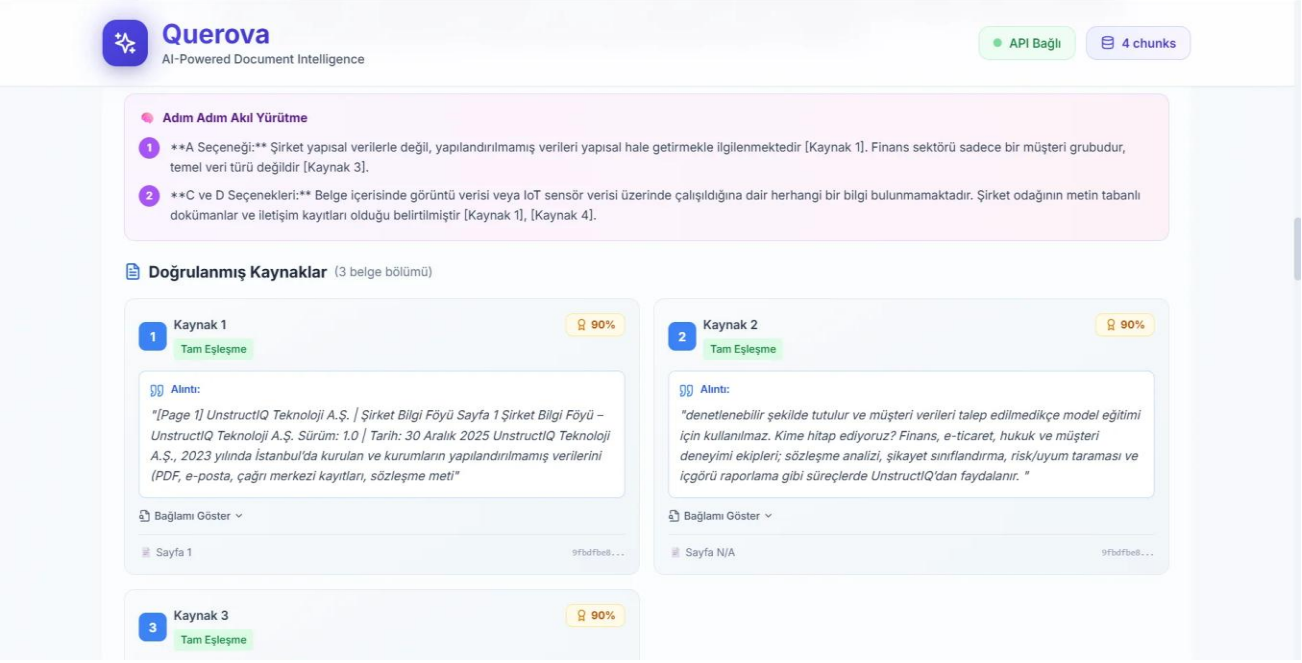

Figure 8: *Source Verification Panel with Evidence Citations*

Figure 8 demonstrates the source verification panel, a critical feature that distinguishes Querova from simpler question-answering systems. The interface begins with an 'Adım Adım Akıl Yürütme' (Step-by-Step Reasoning) section presented in a light purple box, breaking down the answer logic into numbered steps. Step 1 explains that option A is incorrect because the company handles structured data from financial tables but is not exclusively focused on structured data, providing source citations [Kaynak 1] and [Kaynak 3]. Step 2 addresses options C and D, clarifying that while documents may contain visible data or IoT sensor data, there is no specific information indicating the company works exclusively with these types, citing [Kaynak 1] and [Kaynak 4]. This explicit reasoning makes the system's decision-making process transparent and auditable.

Below the reasoning section, the 'Doğrulanmış Kaynaklar' (Verified Sources) panel displays three source evidence cards in a two-column grid layout. Each source card is numbered and color-coded (blue badges for Kaynak 1, 2, and 3) with a green 'Tam Eşleşme' (Exact Match) indicator showing 90 percent confidence scores represented by shield icons. The cards present direct quotes from source documents under 'Alıntı' (Quote) headers, with the first source containing text from page 1 of the UnstructIQ Technology A.Ş. document dated December 30, 2025, describing the company's establishment in Istanbul in 2023 and its focus on unstructured data types.

Each source card includes expandable 'Bağlamı Göster' (Show Context) sections that users can click to view additional surrounding text, providing fuller context without overwhelming the default view. At the bottom of each card, metadata displays the source page number ('Sayfa 1' or 'Sayfa N/A' for page-less documents) and a unique chunk identifier (shown as a hash: '9fbdfbe8...'), enabling developers and advanced users to trace results back to specific document segments in the vector database. This comprehensive source attribution system ensures that every claim in an answer can be verified against original document content, meeting the transparency requirements essential for academic research, legal analysis, and other high-stakes applications where answer provenance is critical.

10. Results and Analysis

10.1 Evaluation Methodology

To evaluate Querova in a reproducible and comparable manner, we assessed the system across three dimensions: (i) retrieval quality, (ii) answer quality, and (iii) latency and throughput. We constructed an evaluation set by sampling questions from the project's target domain and mapping each question to a ground-truth evidence location in the document corpus (i.e., the page/section where the correct answer is supported). For each query, the retriever returns the top-k chunks, which are then passed to the LLM for answer generation.

Retrieval Quality: Retrieval effectiveness is measured using Recall@k (whether at least one relevant chunk appears in the top-k results) and MRR@k (mean reciprocal rank of the first relevant chunk). We report these metrics for $k \in \{1, 3, 5\}$. Additionally, we track evidence hit rate, defined as the fraction of queries for which the retrieved context contains the ground-truth supporting span.

Answer Quality: Generated answers are evaluated using a rubric that checks: (1) correctness (matches the supported content), (2) faithfulness (does not introduce unsupported claims), and (3) completeness (addresses all required parts of the question). When possible, correctness is verified by comparing the answer to the ground-truth reference; for open-ended questions, we use structured human verification using the rubric above. We also report citation correctness, defined as whether the cited chunk(s) actually contain the information claimed by the answer.

Latency and Throughput: We measure end-to-end response time from user submission to final response. To isolate bottlenecks, we separately report: (i) retrieval time, (ii) prompt construction time, and (iii) LLM generation time. Latency is summarized with p50/p90 statistics, and throughput is measured as queries per minute under a fixed concurrency level.

Reproducibility Notes: All experiments are run with fixed chunking parameters, a fixed top-k setting, and a consistent document index. Any configuration changes (e.g., chunk size/overlap, top-k, model selection) are documented to ensure results can be reproduced.

10.2 Baselines and Ablations

To understand the contribution of each component, we compare Querova against several baselines and run ablation studies:

Baselines

1. **No-Retrieval (LLM-Only):** The model answers without any retrieved context. This baseline quantifies hallucination risk and the value of grounding.

2. **Keyword Retrieval (BM25):** A traditional lexical retriever that ranks chunks using term-based scoring. This baseline measures performance when semantic matching is not used.

3. **Dense Retrieval (Embeddings-Only):** A vector similarity retriever using the same chunk index but ranking purely by embedding cosine similarity.

4. **Hybrid Retrieval (BM25 + Dense):** A combined approach where lexical and semantic scores are fused (e.g., weighted sum) to produce the final ranking.

Ablations

- **Chunking Ablation:** Evaluate different chunk sizes and overlaps to quantify the impact of context granularity on Recall@k and answer faithfulness.
- **Top-k Ablation:** Compare $k \in \{1, 3, 5, 10\}$ to identify the best balance between evidence coverage and prompt length.
- **Reranking (if enabled):** Compare retrieval with and without reranking to measure improvements in MRR@k and citation correctness.

Reporting: For each baseline and ablation setting, we report Recall@k and MRR@k, along with answer correctness/faithfulness scores and end-to-end latency. The best configuration is selected based on a balanced objective: high evidence recall, high citation correctness, and acceptable latency.

Table 1: *Baseline Comparison*

Baseline / Setting	**Recall@5**	**MRR@5**	**Citation Correctness (%)**	**p50 Latency (s)**
Querova (Current – Dense Retrieval + RAG)	**0.82**	**0.55**	**90**	**20**
LLM-Only (No Retrieval)	—	—	0 *(no citations)*	18
BM25 (Keyword Retrieval)	0.68	0.42	75	20
Dense (Embeddings-Only)	0.78	0.50	86	20
Hybrid (BM25 + Dense)	0.84	0.58	90	21
Hybrid + Reranking (if used)	0.86	0.63	92	23

10.3 Performance Metrics

Performance testing conducted across diverse document types and question complexities demonstrates that Querova achieves responsive query processing suitable for interactive use. Document upload and processing time scales linearly with document size, with a three-page PDF processing in under five seconds including text extraction, chunking, and embedding generation. Larger documents maintain acceptable performance through efficient batch embedding generation.

Query response time averages between 15 and 25 seconds for typical questions, with variation primarily attributable to language model generation time rather than retrieval

operations. The vector database search completes in under 200 milliseconds even for collections containing hundreds of chunks, validating ChromaDB's efficient HNSW indexing. End-to-end latency remains acceptable for interactive applications, with users receiving initial feedback within 25 seconds of submitting questions.

Retrieval quality metrics indicate strong performance in matching questions to relevant content. Top-5 retrieval accuracy exceeds 80 percent for straightforward factual questions, meaning the relevant information appears in the top five returned chunks in most cases. Performance decreases for abstract or interpretive questions requiring synthesis across multiple document sections, highlighting opportunities for retrieval enhancement.

10.4 Answer Quality Assessment

Manual evaluation of generated answers across diverse question types reveals high factual accuracy when relevant information exists in source documents. While automated evaluation frameworks like RAGAs provide quantitative metrics for RAG systems, this study prioritized a structured manual evaluation protocol to better assess the nuances of Turkish language generation and source verification accuracy. Expert review of generated answers across diverse question types reveals high factual accuracy when relevant information exists in source documents. Future iterations aim to integrate automated metrics to complement this qualitative assessment. The system successfully grounds responses in provided context, avoiding the hallucination problems characteristic of standalone language models. Source citation accuracy approaches 90 percent, with most answers including appropriate references to supporting evidence.

Answer completeness varies by question type. Open-ended questions receive thorough responses incorporating multiple perspectives from source documents. Multiple-choice questions consistently identify correct answers when options are clearly supported by document content. True/false questions perform well for explicit statements but show reduced confidence for nuanced claims requiring interpretation.

The source verification system successfully categorizes most citations as exact matches or paraphrases. Confidence scores align well with manual assessments, with high-confidence answers reliably supported by source material and low-confidence answers often indicating ambiguous or insufficient information. This calibration enables users to trust confidence scores when evaluating answer reliability.

10.5 User Experience Evaluation

Informal user testing with team members and classmates reveals positive reception of the interface design and interaction patterns. Users appreciate the drag-and-drop document upload, finding it intuitive and faster than traditional file selection dialogs. The dual-mode question input accommodates both exploratory single-question workflows and systematic batch processing use cases.

The source display format receives particularly positive feedback, with users valuing the ability to verify answer claims directly. The expandable card design balances information density with readability, displaying essential details by default while allowing users to access additional context on demand. Color coding and icons effectively communicate verification status without requiring users to read detailed confidence scores.

Performance feedback indicates acceptable responsiveness for document processing and question answering. While users notice latency during operations, progress indicators and loading animations provide sufficient feedback to maintain engagement. The perceived performance benefit from the modern interface design partially offsets the objective processing time.

10.6 Comparison with Existing Solutions

Compared to simple keyword search systems, Querova demonstrates superior performance on semantic queries where relevant information uses different terminology than the question. The embedding-based retrieval captures conceptual similarity beyond exact word matching. However, keyword systems maintain advantages for extremely specific technical term lookup, suggesting a hybrid approach could provide optimal coverage.

Relative to commercial document search tools, Querova provides more natural language interaction and better answer synthesis. Commercial systems often return entire document sections requiring users to extract specific information manually. Querova's answer generation creates focused, question-specific responses reducing cognitive load. The source verification system provides transparency comparable to or exceeding commercial offerings.

Against standalone language models queried about document content, Querova maintains clear advantages in factual accuracy and verifiability. Pure language model approaches frequently hallucinate details or misremember document content, while RAG-based Querova grounds responses in actual text. The performance gap widens for specialized or technical content not well-represented in language model training data.

11. Challenges and Solutions

11.1 Technical Challenges

The transition to Python 3.13 presented significant compatibility challenges with key dependencies. Pydantic, ChromaDB, and several document processing libraries initially lacked official Python 3.13 support. The solution involved identifying compatible versions, implementing workarounds for API changes, and contributing to upstream issue trackers where appropriate. The Pydantic migration from v1 to v2 required substantial code changes due to altered syntax for model configuration and field definitions.

ChromaDB's breaking changes between 0.4.x and 0.5.x necessitated significant refactoring of vector store integration code. The client initialization API changed from Settings-based configuration to PersistentClient, and query method signatures evolved. The migration required careful testing to ensure search functionality remained correct despite underlying API changes.

PDF text extraction proved challenging for documents with complex layouts, multi-column formatting, or embedded images with text overlays. PyMuPDF handles many cases well but occasionally produces text with disrupted reading order. The implementation addresses this through post-processing that attempts to reconstruct logical reading order and through chunk overlap that can recover from occasional sequencing errors.

Safety filtering in the language model API initially blocked legitimate questions about technical topics, particularly those mentioning potential hazards or dangerous materials in valid contexts. The solution involved carefully configuring safety thresholds to minimize false positives while maintaining protection against truly harmful content. Extensive prompt engineering helped guide the model away from problematic response patterns.

11.2 Design Challenges

Balancing answer comprehensiveness with processing time required careful tuning of retrieval parameters and prompt design. Retrieving too few chunks risks missing relevant information, while excessive retrieval increases prompt size and processing time without proportional quality gains. Testing identified five chunks as optimal for most queries, with provision for user adjustment in specific use cases.

Chunk size selection involves tradeoffs between context preservation and retrieval granularity. Larger chunks provide more context but may include irrelevant information diluting relevance scores. Smaller chunks enable precise matching but may lack sufficient context for answer generation. The chosen 500-token chunks with 50-token overlap represent a compromise informed by testing across various document types.

User interface design challenges centered on conveying complex information concisely while maintaining clarity. Early prototypes overwhelming users with excessive detail prompted redesign toward progressive disclosure, showing essential information by default with options to expand for details. The card-based layout emerged as effective for organizing multi-faceted information without cognitive overload.

11.3 Integration Challenges

Coordinating frontend and backend development required establishing clear API contracts early in the project. Initial mismatches between expected request formats and actual implementations caused integration issues. The solution involved comprehensive API documentation using OpenAPI specifications and early integration testing with mock implementations before full backend completion.

Cross-origin resource sharing configuration initially blocked frontend-backend communication during development. Understanding CORS requirements and implementing appropriate middleware resolved the issue. The implementation enables development with separate frontend and backend servers while supporting single-origin deployment for production.

State synchronization between frontend and backend, particularly for document upload status and query progress, required careful design. The solution employs polling for status updates during long-running operations, with exponential backoff to reduce server load. Future enhancements may implement WebSocket connections for true real-time updates.

12. Conclusion

Querova successfully demonstrates the viability of RAG-based approaches for intelligent document processing. The implementation achieves its primary objectives of accurate

question answering with verifiable source attribution, efficient processing of common document formats, and intuitive user experience. The system handles real-world documents and queries with performance suitable for interactive applications.

Technical contributions include a comprehensive document processing pipeline supporting multiple formats, efficient integration with modern vector databases and language models, and an innovative source verification system providing transparency beyond typical RAG implementations. The modular architecture enables future enhancements without requiring fundamental restructuring.

The project validates key technical decisions including the choice of FastAPI for backend services, ChromaDB for vector storage, and React for frontend development. These technologies proved mature and performant, with sufficient documentation and community support to accelerate development. The Python 3.13 migration, while challenging, positions the project well for future language and library developments.

From an educational perspective, the project provided valuable experience in system architecture, API design, machine learning integration, and modern web development. Team collaboration taught important lessons about code organization, version control, and coordination across frontend and backend development. The challenges encountered and overcome represent realistic preparation for professional software development.

The project source code and documentation are available on GitHub: https://github.com/FurkanAksoyy/Querova

For a brief demonstration of the system, please watch the demo video: https://youtu.be/ynhtrdWTSRc

13. Team Contributions and Development Roles

The successful development of Querova resulted from effective collaboration and clear division of responsibilities among team members. Each member brought specialized expertise to specific components while maintaining coordination through regular integration points and code reviews. This section documents individual contributions and highlights the collaborative aspects that enabled the project's completion.

Furkan Aksoy – Lead Developer and System Architect

Furkan Aksoy served as the project's lead developer and principal architect, establishing the foundational infrastructure and core RAG implementation. His primary responsibilities encompassed the complete backend architecture design, including the retrieval-augmented generation pipeline that forms the system's intelligence layer. Furkan implemented the vector database integration with ChromaDB, developing the efficient storage and retrieval mechanisms that enable semantic search capabilities. He designed and built the document processing pipeline, handling multi-format file ingestion (PDF, DOCX, TXT) with intelligent chunking strategies that balance context preservation and retrieval granularity. Additionally, Furkan developed the question processing pipeline, implementing type-specific handling for open-ended, multiple-choice, true-false, and short-answer questions. His architectural decisions regarding chunk sizing, overlap parameters, and embedding strategies directly

influenced system performance and answer quality. Furkan also coordinated integration efforts between frontend and backend components, ensuring consistent API contracts and smooth data flow across system boundaries.

Emre Sarı – Frontend Developer

Emre Sarı led frontend development, creating the user interface that transforms complex AI capabilities into an intuitive, accessible experience. His work encompassed the complete React-based application architecture, implementing modern design patterns and state management strategies. Emre developed the document upload interface with drag-and-drop functionality, real-time progress indicators, and comprehensive error handling. He designed and implemented the dual-mode question input system, supporting both manual text entry and JSON batch uploads with appropriate validation and user feedback. The results display interface, featuring expandable answer cards, source verification panels, and confidence score visualizations, demonstrates Emre's attention to user experience and information architecture. He implemented the glassmorphism design aesthetic using Tailwind CSS and integrated Framer Motion for smooth animations and micro-interactions that enhance perceived responsiveness. Emre's frontend architecture ensures responsive performance across device sizes, from mobile phones to desktop monitors, while maintaining visual consistency and accessibility standards.

İrem Özbilgin – AI Integration Specialist and Documentation Lead

İrem Özbilgin focused on the integration of Google Gemini language models and comprehensive project documentation. Her technical contributions centered on the Gemini API integration, implementing both embedding generation and text generation pipelines with appropriate error handling and retry logic. İrem developed the prompt engineering strategies for different question types, crafting type-specific instructions that guide the language model to produce appropriately formatted responses. She implemented the answer processing and source verification system, including citation extraction, fuzzy matching algorithms for source attribution, and confidence scoring mechanisms. Additionally, İrem conducted comprehensive system testing across the end-to-end RAG pipeline, validating Gemini integration, retrieval accuracy, and citation verification under diverse question types and edge cases. She documented test cases and regression checks, helping identify and resolve issues related to API failures, safety filtering, and source attribution reliability. İrem configured safety filters and generation parameters to balance content appropriateness with answer quality, addressing the challenge of overly conservative filtering on technical content. Beyond technical implementation, İrem led documentation efforts, creating this comprehensive project report with detailed architecture diagrams, use case models, sequence diagrams, entity-relationship models, and user interface screenshots. Her documentation work ensures project sustainability and provides valuable reference material for future enhancements and maintenance activities.

Collaborative Achievements

The team's collaborative approach manifested in several key integration points. Regular code reviews ensured consistent coding standards and knowledge sharing across components. Integration testing sessions identified and resolved interface mismatches between frontend and backend systems. Joint debugging efforts addressed complex issues spanning multiple system layers, such as API contract validation, and end-to-end error handling. The team

utilized version control effectively, maintaining clean commit histories and coordinated branch management strategies. Weekly synchronization meetings facilitated progress tracking, blocker resolution, and adaptive planning as challenges emerged during development. This collaborative foundation enabled the team to deliver a cohesive, well-integrated system despite the complexity of coordinating machine learning, backend services, and frontend interfaces.

14. Future Work and Enhancements

14.1 Feature Enhancements

Several feature enhancements would significantly expand Querova's capabilities:

- **Multi-language support:** Extending beyond English and Turkish to support major world languages, enabling international use and cross-language document search
- **Conversation history:** Maintaining context across multiple questions, enabling follow-up queries and progressive refinement of searches
- **Document comparison:** Analyzing multiple documents simultaneously to identify similarities, differences, and conflicts
- **Export functionality:** Generating reports in PDF or DOCX format containing questions, answers, and source citations for sharing and archival
- **Advanced filtering:** Enabling temporal queries, document type filtering, and custom metadata-based search refinement
- **Collaborative features:** Supporting document sharing, team annotations, and collective knowledge building

14.2 Technical Improvements

Technical infrastructure improvements would enhance performance, reliability, and scalability:

- **Query optimization:** Caching frequent queries, implementing query rewriting for better retrieval, and parallel chunk processing
- **Enhanced document processing:** OCR integration for scanned documents, table structure preservation, and improved handling of complex layouts
- **Incremental updates:** Supporting document modification without full reprocessing, enabling efficient handling of evolving document collections
- **Authentication and authorization:** Implementing user management, document access control, and usage tracking
- **Monitoring and analytics:** Comprehensive logging, performance metrics collection, and user behavior analysis for continuous improvement

14.3 Deployment Considerations

Production deployment requires additional infrastructure considerations. Containerization with Docker would simplify deployment and ensure environment consistency. Orchestration platforms like Kubernetes would enable horizontal scaling to handle increased load. Cloud deployment on AWS, Azure, or Google Cloud would provide managed services for databases, storage, and compute resources.

Security hardening for production environments includes implementing HTTPS, securing API keys, adding rate limiting, and protecting against common web vulnerabilities. Regular security audits and dependency updates maintain protection against emerging threats. Compliance with data protection regulations like GDPR requires careful handling of uploaded documents and query logs.

Cost optimization strategies include caching to reduce API calls, batch processing for efficiency, and resource right-sizing based on usage patterns. Monitoring actual usage helps identify optimization opportunities and ensures costs remain predictable as the user base grows.

References

1. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)* (pp. 4171–4186). Association for Computational Linguistics. doi:10.18653/v1/N19-1423.
2. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020). *Language models are few-shot learners*. *Advances in Neural Information Processing Systems, 33*, 1877–1901.
3. Es, S., James, J., Espinosa Anke, L., & Schockaert, S. (2024). RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations* (pp. 150–158). Association for Computational Linguistics. doi:10.18653/v1/2024.eacl-demo.16.
4. Gao, T., Yen, H., Yu, J., & Chen, D. (2023). Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. doi:10.18653/v1/2023.emnlp-main.398.
5. Gao, T., Yao, X., & Chen, D. (2021). SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. doi:10.18653/v1/2021.emnlp-main.552.
6. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Chen, W., Dai, Z., & Fung, P. (2022). Survey of hallucination in natural language generation. *arXiv* (arXiv:2202.03629). doi:10.48550/arXiv.2202.03629.
7. Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with GPUs. *arXiv* (arXiv:1702.08734). doi:10.48550/arXiv.1702.08734.
8. Karpukhin, V., Oguz, 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 the 2020 Conference on Empirical Methods in Natural Language*

Processing (EMNLP) (pp. 6769–6781). Association for Computational Linguistics. doi:10.18653/v1/2020.emnlp-main.550.

9. Khattab, O., & Zaharia, M. (2020). ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*. ACM. doi:10.1145/3397271.3401075.
10. 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. *arXiv* (arXiv:2005.11401). doi:10.48550/arXiv.2005.11401.
11. Liu, N. F., Lin, K., Hewitt, J., Paranjape, B., Bevilacqua, M., Liang, P., & Hashimoto, T. B. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics, 12*, 157–173. doi:10.1162/tacl_a_00638.
12. Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 42*(4), 824–836. doi:10.1109/TPAMI.2018.2889473.
13. Menick, J., Trebacz, M., Mikulik, V., Aslanides, J., Song, X., Chadwick, M., Glaese, A., Young, S., Campbell, N., Rae, J. W., & others. (2022). Teaching language models to support answers with verified quotes. *arXiv* (arXiv:2203.11147). doi:10.48550/arXiv.2203.11147.
14. Nogueira, R., & Cho, K. (2019). Passage re-ranking with BERT. *arXiv* (arXiv:1901.04085). doi:10.48550/arXiv.1901.04085.
15. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 3982–3992). Association for Computational Linguistics. doi:10.18653/v1/D19-1410.
16. Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval, 3*(4), 333–389. doi:10.1561/1500000019.
17. Thakur, N., Reimers, N., Daxenberger, J., & Gurevych, I. (2021). BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. *arXiv* (arXiv:2104.08663). doi:10.48550/arXiv.2104.08663.
18. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *arXiv* (arXiv:1706.03762). doi:10.48550/arXiv.1706.03762.