Generative AI for Business

Frameworks, Techniques, and Governance

Wei Chen
University of Connecticut

Liwei Chen
University of Cincinnati

First Edition (2025)

Paperback ISBN 979-8-9997161-0-1
Hardcover ISBN 979-8-9997161-1-8
Kindle Edition ISBN 979-8-9997161-2-5

First edition, first printing, 2025

Published by GenAI Flows Publishing
For permissions, instructor copies, or inquiries, contact publishing@genai4all.org.
Supplementary materials available at https://genai4all.org

PREFACE

Welcome to *Generative AI for Business*. When we first began teaching generative AI (GenAI) to university business students in Fall 2024, we noticed a common pattern: the students were bright, ambitious professionals eager to harness this transformative technology, but lacked the structured resources to bridge theory and practice. At our universities, the University of Connecticut and the University of Cincinnati, we heard the same questions, whether we were teaching in person, online, or in hybrid formats: "How can I apply this without becoming a programmer?" and "Do I need expensive infrastructure to get started?" We also heard similar sentiments from colleagues at other universities: "Do you have a textbook for this?" and "How can I teach this to my students?"

This book was born from these questions. The landscape of artificial intelligence has undergone a remarkable transformation before our eyes. Only a few years ago, technologies like ChatGPT and DALL·E were experimental curiosities whose use was confined to research papers and technical demonstrations. Today, these technologies are reshaping entire industries and creating new possibilities at breathtaking speed. Business leaders, product managers, entrepreneurs, and professionals across multiple sectors find themselves navigating a revolution in real time, often without a map. While many recognize that generative AI represents a pivotal shift, translating that recognition into practical, value-generating applications remains challenging. Our aim was to bridge this gap.

This book offers a structured approach to implementing and governing GenAI technologies in business contexts. We move beyond the initial excitement of AI demonstrations to focus on effective and responsible business applications. Chapter 1 provides an accessible introduction to GenAI fundamentals, while Chapter 2 delves deeper into the technical architecture for readers who desire a more comprehensive understanding.

The book then progresses through key enabling GenAI frameworks and techniques for business applications. Chapters 3 and 4 cover prompt engineering techniques, providing both essential methods and advanced strategies to control GenAI outputs with precision. Chapters 5 and 6 explore retrieval augmented generation (RAG) systems, demonstrating how to connect GenAI models with enterprise knowledge bases. Chapters 7 and 8 examine agentic systems that enable AI to operate autonomously in solving complex business problems. Chapter 9 covers fine-tuning. Chapter 10 starts to examine the key challenges organizations face in using GenAI responsibly, including issues of privacy, bias, and reliability. Finally, Chapters 11 and 12 offer practical and governance-oriented solutions, covering how to op-

erationalize GenAI systems effectively and integrate them responsibly within organizational frameworks.

Throughout these pages, practical examples, case studies, and prompt samples illustrate how theoretical concepts translate into business value. For undergraduate students and business professionals who seek practical implementation knowledge, we recommend focusing on Chapters 1, 3, 5, 7, 8, and selected sections from Chapters 10 to 12. Readers who want to develop deeper technical expertise can benefit from the more in-depth material in Chapters 2, 4, 6, and 9. This flexible approach accommodates readers with different backgrounds and learning objectives.

Whether you are taking your first steps with GenAI or seeking to optimize advanced implementations, this book provides actionable insights for leveraging these powerful technologies. Our goal is not simply to explain how GenAI works, but to structure how it can work for you and your organization. We hope this book will serve as both a practical guide for implementation and a foundation for continued exploration as the field evolves. Welcome to the journey of putting GenAI to work in your business!

Disclaimer

In writing this book, we have utilized GenAI tools to assist in drafting and refining content. However, we have meticulously reviewed and edited all of the writing to ensure accuracy, clarity, and relevance. We firmly believe that although GenAI can be a powerful aid, human evaluation remains indispensable. As authors, we have taken great care to validate and refine the outputs of these tools to ensure that the final content reflects our expertise and judgment.

Resources

We also recognize that GenAI is a rapidly evolving field. The technologies and frameworks discussed in this book are current at the time of writing, but we encourage readers to stay engaged with the latest developments through our open-source course website at https://genai4all.org. The landscape is changing quickly, and new tools, techniques, and ethical considerations will continue to emerge. We would like to have you join us in building the future together.

ABOUT THE AUTHORS

Wei Chen, Ph.D.

University of Connecticut

Wei Chen is an Associate Professor of Operations and Information Management (OPIM) at the School of Business, University of Connecticut. He is also the academic director of the UConn Digital Frontiers Initiative. Wei's current research focuses on the design and governance of generative AI. His work on Platforms, Crowds, and Financial Technologies (FinTech) has been published in leading academic journals, such as *Management Science*, *Information Systems Research*, and *MIS Quarterly*. Wei teaches Generative AI for Business in the graduate programs at UConn School of Business to MS BAPM, MS FinTech, and MBA students, and frequently speaks to industry partners and conferences on GenAI.

Liwei Chen, Ph.D.

University of Cincinnati

Liwei Chen is an Associate Professor in the Department of Operations, Business Analytics, and Information Systems at the Carl H. Lindner College of Business at the University of Cincinnati. Liwei's research focuses on the empowering roles of IT in healthcare and business contexts. She has published in leading academic journals such as *MIS Quarterly*, *Information Systems Research*, *Journal of the Association for Information Systems*, *European Journal of Information Systems*, and *Journal of Medical Internet Research*. She teaches Generative AI for Business to students in the MS Business Analytics, MS Information Systems, and MBA programs, and the Artificial Intelligence in Business Graduate Certificate at the Carl H. Lindner College of Business, University of Cincinnati.

To Karen, for your unwavering love, patience, and partnership—and to my mom, for your endless care and devotion to Atlas and now to Ivy.

—Wei

To Ruizhi, for being my partner in every step of the journey.
To Hunter and Hayden, whose curiosity and joy inspire me each day.
And to my parents, whose love and support have made everything possible.

—Liwei

CONTENTS

Part II

Frameworks and Techniques

CHAPTER 5

RETRIEVAL-AUGMENTED GENERATION (RAG)

Large language models (LLMs) are powerful, but they have a major limitation: They cannot learn new information after pre-training without a retraining process, which can require a lot of time and resources. This makes them less useful for tasks that need more up-to-date or specific information, such as answering questions about new events or private data.

Retrieval-augmented generation (RAG) solves this problem by letting LLMs pull in fresh, relevant information from external sources when needed. This means that the AI system can provide more accurate and relevant answers by combining what the LLM already knows with the most up-to-date data. In this chapter, we discuss the basic pipeline of a RAG system.

5.1 Introduction to RAG

In recent years, the capabilities of GenAI models have expanded dramatically. However, as the technology has developed, the limitations have also become apparent. The models that power these systems, while vast in their knowledge, are static in nature. Once trained, these models cannot incorporate new information until they are retrained, which is both time consuming and resource intensive. This static nature creates gaps, especially when dealing with dynamic or real-time information needs.

This is where RAG comes into play. RAG represents a paradigm shift in the field of GenAI by enabling models to interact with external data sources in real time (P. Lewis, Perez, Piktus, Petroni, Karpukhin, Goyal, Küttler, M. Lewis, W.-t. Yih, et al. 2021). Instead of relying solely on the information encoded during pre-training, RAG systems retrieve relevant, up-to-date information at the time of generation, allowing for more customized and contextually accurate responses (Y. Gao et al. 2024). This approach fundamentally changes how we interact with GenAI models, transforming them from static repositories of pre-existing knowledge into dynamic tools that adapt to changing information.

By combining retrieval mechanisms with generative capabilities, RAG allows GenAI systems to maintain relevance and accuracy, even in the face of rapidly evolving data. This chapter delves into how a typical RAG system works, exploring the RAG pipeline and its components from embeddings to vector databases and retrieval models.

Why RAG?

The introduction of RAG addresses one of the most pressing limitations of pretrained LLMs: their static nature. Once trained, these models capture the knowledge that is available up to the specific point in time when the training data is collected, effectively freezing the model's understanding. This limitation can lead to outdated responses, especially in domains where information changes frequently, such as technology, medicine, or news. Without retraining, a costly and lengthy process, the model cannot reflect new information or adapt to real-time queries. More dangerously, it may produce hallucinations, that is, incorrect or fabricated information, when asked about topics outside its training data. The reasons for RAG include the following.

Up-to-date Knowledge for Real-Time Queries

Traditional language models are invaluable for tasks that rely on generalized knowledge. However, for applications requiring up-to-date, contextual information, such as customer support, legal research, or academic inquiry, these models fall short. For example, a medical model trained on general health data may not know the latest treatments, while a customer support model may lack specifics about product updates. RAG allows for integration with external databases or resources, ensuring that responses are not only accurate but also current.

Enhanced Contextualization

By combining retrieval mechanisms with generation, RAG enables a more personalized user experience. Instead of merely relying on generalized knowledge, RAG systems can draw from specific information sources relevant to the user's query, delivering a higher level of contextual accuracy. This dynamic retrieval process means that responses can be tailored to specific industries, preferences, or even individual use cases, making GenAI interactions more meaningful and valuable.

Reducing LLM Hallucinations

GenAI models excel in understanding and generating language but often produce hallucinations when faced with niche or proprietary knowledge gaps (Y. Zhang, Y. Li, et al. 2023). For instance, a medical model trained on general health data might fabricate information about the latest treatments, or a customer support model could invent details about product updates. Research has shown that larger models do not solve this problem and may even hallucinate more (L. Zhou et al. 2024). RAG mitigates these issues by dynamically sourcing information from specialized or proprietary knowledge repositories, thus ensuring that responses are accurate, reliable, and grounded in factual data.

By augmenting response generation with these retrieval capabilities, RAG not only broadens the potential applications of GenAI but also improves the reliability of responses. In the following section, we dive into a deeper exploration of how the RAG pipeline functions and the components that make this possible.

5.2 An Overview of the RAG Pipeline

The RAG pipeline operates by combining retrieval and generation processes (hence the name retrieval-augmented generation) to produce accurate and context-aware responses. As shown in Figure 5.1, the pipeline begins with the user submitting a query, which serves as the input for the system. This query is processed by a retrieval model that searches a knowledge base for the most relevant documents or pieces of information related to the query. These retrieved documents are then provided as context to a pre-trained LLM, which integrates the query and the retrieved information to generate a response.[1]

By leveraging the retrieval component for factual accuracy and the generative component for fluency and coherence, the RAG pipeline ensures that users receive responses that are both informative and well articulated. This synergy between retrieval and generation makes the RAG pipeline particularly effective for tasks requiring domain-specific knowledge or precise contextualization.

Comparing the RAG pipeline to the operation of a traditional LLM without RAG show-

[1] In this chapter, we focus on what (Y. Gao et al. 2024) call "Naive RAG," and expand to more complicated cases in the next chapter.

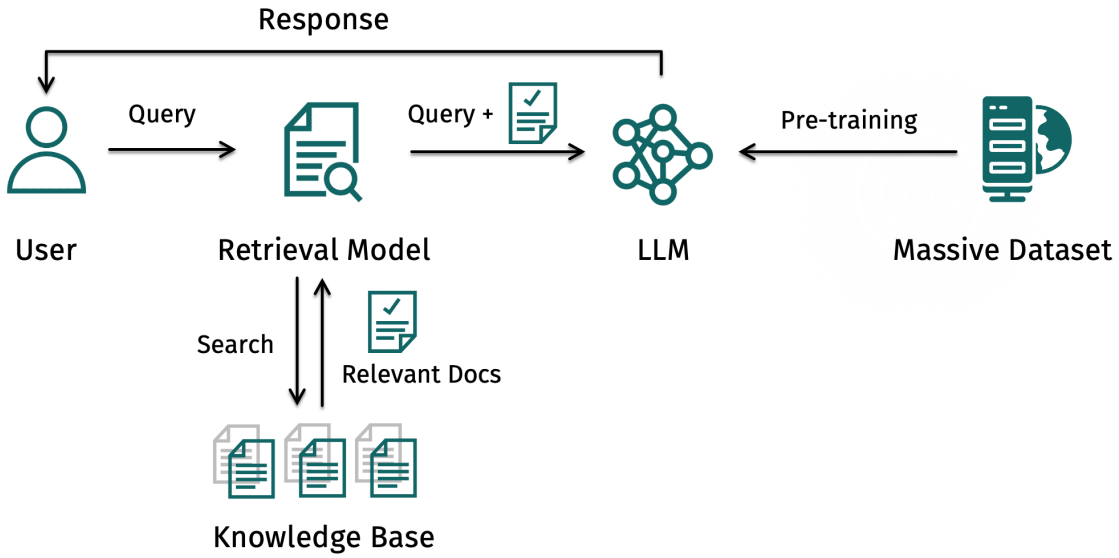

Figure 5.1: The RAG Pipeline

cases its benefits. Without RAG, an LLM functions similarly to a student taking a closed-book exam, relying solely on its internal memory to respond to questions. The model's answers are confined to the knowledge it was trained on, which may not include up-to-date or domain-specific information. This can result in gaps in accuracy, particularly when the model is confronted with inquiries that fall outside its training scope.

In contrast, the RAG pipeline transforms the scenario by introducing a "cheat sheet" into the exam process. Here, the retrieval model plays the role of accessing a large library to pull relevant information, creating a condensed cheat sheet tailored to the user's query. This cheat sheet is then given to the LLM, which integrates the fresh, query-specific information into its response. As a result, the RAG pipeline combines the factual depth of retrieval with the LLM's generative fluency, ensuring more accurate and contextually relevant responses in comparison to the closed-book model of a standalone LLM.

The Key Components of the RAG Pipeline

Figure 5.2 provides a detailed overview of the RAG pipeline, with the elements enclosed in the dashed rectangle representing the retrieval model. We emphasize three key components within this pipeline: *embeddings*, *vector databases*, and the *retrieval process*. Together, these elements improve the generative capabilities of LLMs by incorporating accurate and context-sensitive information retrieval into the response generation process. For a full understanding of the RAG pipeline, we first explain the function of each component in the workflow and then delve deeper into each of these in the following sections.

Text embeddings are the foundation of the retrieval model: it converts documents, web pages, or other pieces of textual information into dense numerical representations called embeddings (Mikolov, Sutskever, et al. 2013; Devlin et al. 2019a). These embeddings capture the semantic meaning of the text, allowing the system to efficiently measure the similar-

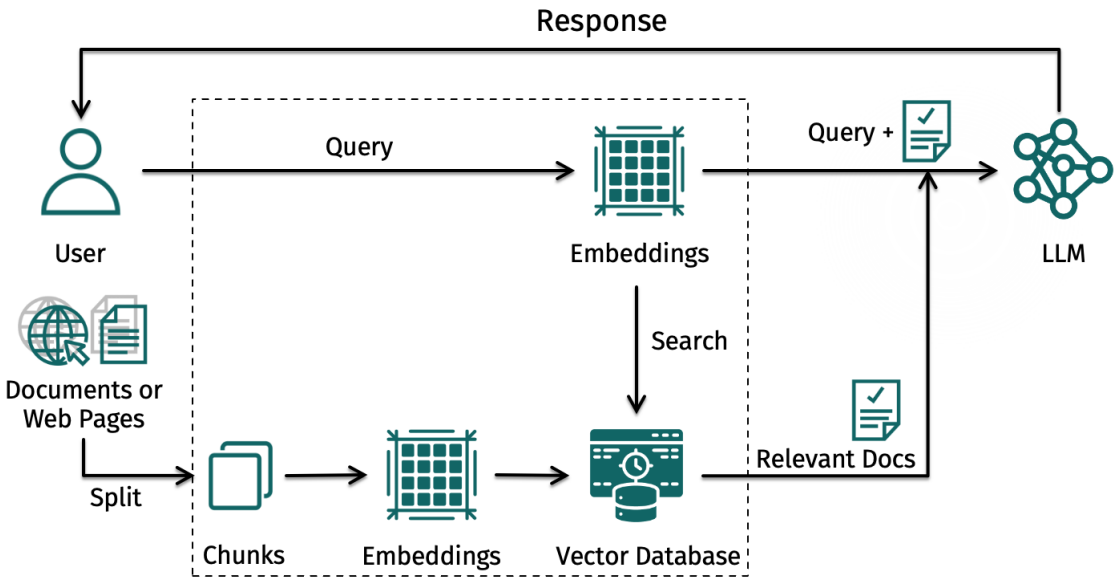

Figure 5.2: The Detailed RAG Pipeline

ity between queries and stored documents. Each document is split into smaller chunks to ensure manageable units for processing, and these chunks are then transformed into their corresponding embeddings.[2]

The *vector database* acts as the storage and retrieval engine for these embeddings. Rather than storing raw text, this database houses the numerical embeddings, enabling fast and accurate similarity searches. When a query is submitted, its embedding is generated in real time and compared to the stored embeddings in the vector database. This comparison identifies the most relevant chunks, effectively narrowing the vast collection of documents down to those that are most contextually relevant to the query.

The *retrieval process* ties everything together. Starting with the query, the system searches the vector database for the closest matching embeddings. The retrieved chunks are then passed on as context to the LLM. This step ensures that the LLM is not operating in isolation but is instead enriched with query-specific, up-to-date information. By integrating these components, the retrieval model enables the RAG pipeline to combine the precision of a knowledge base with the creative, generative power of the LLM.

[2]Recently, there have been discussions centered on comparing long-context LLMs and RAG systems. For interested readers, several notable papers provide valuable insights into this comparison (Jiang, Ma, et al. 2024; Z. Li et al. 2024; Laban et al. 2024).

5.3 Embeddings: Semantic Representations of Text

What Are Embeddings?

At its core, an embedding is a way to represent words, sentences, or even entire documents as numerical vectors. These vectors capture the semantic meaning of the text, allowing computers to understand relationships and similarities between different pieces of information. Traditional keyword-based methods, which rely on exact word matches, can miss relevant information if the exact words from a user's query are not present in the document. For example, a search for "renewable energy solutions" might not return documents about "solar panels" or "wind turbines."

Embeddings solve this by creating a high-dimensional space where texts with similar meanings are clustered near each other. For example, in this space, phrases like "renewable energy solutions," "green technology," and "solar energy systems" would have similar vector representations and thus be located close together. This semantic proximity ensures that related documents can be retrieved, even when they don't use the same wording as the query. By capturing the deeper meaning behind a text, embeddings enable more accurate and contextually relevant searches.

How Embeddings Work

How, precisely, do embeddings map a piece of text into a high-dimensional space? Embeddings transform text into numerical vectors through a sophisticated process of semantic encoding. Each dimension in these vectors represents a learned feature or characteristic of the text, and collectively they capture both explicit and subtle aspects of meaning. This is illustrated in Figure 5.3, which shows the embedding of the word "king" as a 50-dimensional vector.[3] While the individual numbers might appear abstract, each value contributes to a mathematical representation that captures semantic relationships. Together, these dimensions create a unique representation that allows computers to process, compare, and understand the relationships between different pieces of text.

To better comprehend the semantic encoding of embeddings, Figure 5.4 offers a visual representation of word embeddings in 50 dimensions. Each word is depicted as a row of 50 values, with color intensity indicating the magnitude and direction of each dimension's contribution to the word's meaning. This visualization aids in understanding how different words are encoded in the embedding space and how their value patterns create unique representations that capture semantic meaning (dark red represents values close to +2, dark blue represents values close to -2, and white represents values close to 0). Notably, there's a prominent blue column that spans all person-related words (dimension 26), suggesting that they share a common semantic feature along that dimension. Upon closer examination, we

[3]The examples in this section are inspired by Jay Alammar (2019). The illustrations were replicated using a Google Colab notebook. The vector in Figure 5.3 was generated using the "fse/glove-wiki-gigaword-50" model available from Hugging Face at https://huggingface.co/fse/glove-wiki-gigaword-50.

```
[0.50451, 0.68607, -0.59517, -0.022801, 0.60046, -0.13498, -0.08813, 0.47377, -0.61798, -
0.31012, -0.076666, 1.493, -0.034189, -0.98173, 0.68229, 0.81722, -0.51874, -0.31503, -
0.55809, 0.66421, 0.1961, -0.13495, -0.11476, -0.30344, 0.41177, -2.223, -1.0756, -1.0783,
-0.34354, 0.33505, 1.9927, -0.04234, -0.64319, 0.71125, 0.49159, 0.16754, 0.34344, -0.25663,
-0.8523, 0.1661, 0.40102, 1.1685, -1.0137, -0.21585, -0.15155, 0.78321, -0.91241, -1.6106,
-0.64426, -0.51042]
```

Figure 5.3: The Embedding Vector: An example of a 50-dimensional embedding vector representing the word "king." Each number in this vector represents a different semantic feature learned by the embedding model. The combination of these values creates a unique "fingerprint" that captures the meaning of "king" in a way that computers can process and compare.

can observe how gendered pairs like "woman"/"girl" and "man"/"boy" exhibit similar patterns across multiple dimensions (e.g., dimensions 5 and 22), while "king" and "queen" display distinct patterns that differentiate them from the other person-related words (e.g., dimensions 12 and 39)—possibly encoding a concept of royalty. In contrast, the word "water" demonstrates notably different patterns across most dimensions, highlighting how embeddings can effectively distinguish between semantic categories.

The proximity of these vectors within the embedding space indicates the semantic similarity between the represented texts. For example, the vectors for "king" and "queen" will be positioned closely, as will those for "man" and "woman," highlighting their semantic connection. Moreover, the vector difference between "king" and "man" will be akin to that between "queen" and "woman," illustrating analogical relationships. This is often represented by the famous vector equation: $king - man + woman \approx queen$, demonstrating how embeddings can mathematically capture complex semantic relationships. As shown in Figure 5.4, this capability of capturing intricate relationships and context applies not only to words but also to complete sentences, paragraphs, and entire documents.

Embeddings in RAG

Embeddings enable the RAG system to retrieval information based on meaning rather than exact keyword matching. In the RAG pipeline, as shown in Figure 5.2, embeddings are used in two key steps. First, documents in the knowledge base are split into smaller chunks, each converted into an embedding vector. These vectors are stored in a vector database that forms the foundation for semantically aware retrieval. For example, if a user's query is about "carbon reduction strategies," embeddings help the system locate information on related topics like "reducing CO_2 emissions," even if the exact query differs. This ability to understand queries on a semantic level significantly improves the system's retrieval accuracy.

Second, when a user submits a query, the query itself is also converted into an embedding vector. This vector represents the query's meaning and is compared against the pre-stored document embeddings in the vector database using similarity metrics such as cosine similarity.

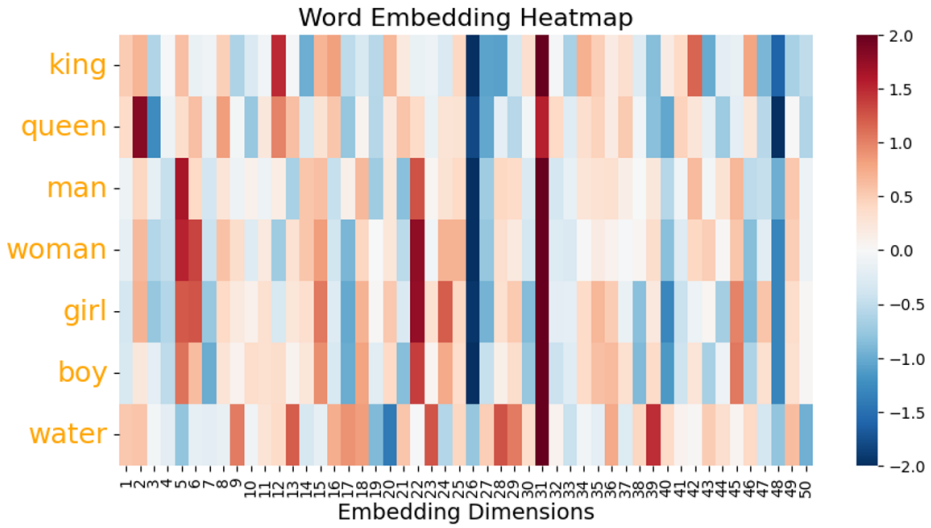

Figure 5.4: Word Embedding Heatmap: A visualization showing how different words are encoded across 50 embedding dimensions. Each row represents a word, and each column represents a dimension in the embedding space. The color intensity indicates the value in that dimension, with dark red representing large positive values (up to 2.0), white representing neutral values (around 0), and dark blue representing large negative values (down to -2.0). Related words like "king" and "queen" or "man" and “woman" show similar patterns across the dimensions, demonstrating how embeddings capture semantic relationships through these numerical representations.

The system retrieves the most relevant document chunks based on the proximity of their embeddings to the query embedding in the vector space. This semantic matching allows the RAG pipeline to locate relevant information even when the query and documents do not share identical keywords.

5.4 Vector Databases: Managing Embeddings Efficiently

The second component of the RAG pipeline, the vector databases, is a specialized infrastructure designed to handle and retrieve embeddings efficiently. Traditional databases are structured to store tabular data, making them inadequate for high-dimensional vectors (e.g., embeddings) that need rapid comparison. Vector databases store these embeddings in a format optimized for semantic search, allowing for fast and accurate data retrieval based on similarity (Pan et al. 2024).

These databases leverage advanced indexing methods, such as Approximate Nearest Neighbors (ANN), to facilitate rapid comparisons between millions of embeddings (Y. Han et al. 2023). This ensures that the system can identify relevant information quickly and accurately, even when operating at scale. For instance, tools like FAISS (Facebook AI Similarity Search), Weaviate, and Milvus are widely used in RAG implementations to support efficient and scalable storage and retrieval of embeddings. Table 5.1 compares some widely used vector databases, highlighting their key features and common use cases in RAG systems.

Table 5.1: Comparison of Popular Vector Databases for RAG Systems

Database	Key Features	Use Cases
FAISS	High-speed ANN search; CPU/GPU support; good for local or embedded deployment	Research, product search, prototyping
Weaviate	Built-in semantic search; GraphQL interface; supports hybrid search (text + vector)	Enterprise RAG, knowledge graphs
Milvus	Highly scalable; cloud-native; supports billions of vectors and hybrid search	Large-scale AI apps, video/image search
Pinecone	Fully managed service; real-time updates; automatic scaling and sharding	SaaS products, personalized recommendations
Qdrant	REST/gRPC API; vector + metadata filtering; focuses on simplicity and performance	Lightweight RAG, AI chatbots

Vector databases also enable seamless scalability, allowing RAG systems to maintain their performance as the size of the knowledge base grows. For example, a vector database might store embeddings for millions of documents, enabling the system to retrieve relevant content in real time for a wide range of applications, from customer support to academic research. By supporting high-speed, semantic similarity-based retrieval, vector databases

form the backbone of the RAG pipeline's retrieval capabilities. They not only ensure efficient handling of embeddings but also enhance the RAG pipeline's ability to deliver timely and contextually relevant responses, making them indispensable for modern generative AI systems.

5.5 The Retrieval Process

The retrieval process is the last critical piece in the RAG pipeline, where the system identifies the most relevant information to respond to a query—that is, what a "cheat sheet" provides to the LLM for the specific query. During this phase, the query embedding is compared with the pre-stored document embeddings in the vector database. Using similarity metrics such as cosine similarity (Mikolov, Sutskever, et al. 2013), the system determines the closest matches. This process ensures that retrieved content aligns closely with the semantic meaning of the query.

For example, consider a query about "global warming impacts on farming." The RAG system might have access to a repository of articles and research papers on climate-related topics, with each document represented as an embedding. Even if the query uses phrasing like "global warming impacts on farming," the system can retrieve relevant documents discussing "temperature rise and crop production" or "climate impact on food systems" because the embeddings capture semantic similarities.

Recent developments in retrieval technology have introduced *hybrid search* techniques that combine traditional *keyword-based* methods with *embedding*-based retrieval (X. Wang, Z. Wang, et al. 2024). Hybrid search leverages the strengths of both approaches: *embedding*-based retrieval excels at finding semantically relevant content by capturing the underlying meaning of a text, while keyword-based search ensures precision by matching exact terms in the query. For instance, if a user queries "effects of climate change on agriculture," embedding-based retrieval might identify documents discussing related topics such as "global warming and crop production" or "impact of extreme weather on farming," even if the exact terms in the query are not present. Keyword search, on the other hand, could prioritize documents that explicitly contain the phrase "climate change" to ensure that they align more closely with the user's intent. By combining both of these techniques, hybrid search delivers results that are both contextually rich and specific, which makes it particularly effective for ambiguous or multi-faceted queries.

Advanced implementations of RAG systems enhance this retrieval process by introducing pre-processing and post-processing steps, effectively transforming the pipeline into a flexible workflow (Y. Gao et al. 2024). Pre-processing may include actions like enhancing data granularity, applying filters based on metadata, or rewriting the query through techniques such as synonym expansion or rephrasing to capture broader contexts. Post-processing, in turn, involves refining the retrieved results through reranking based on domain-specific priorities, summarizing large volumes of text, or applying rules to generate responses that align with user requirements. For example, in a legal domain, pre-processing could filter documents by jurisdiction, while post-processing might prioritize case law that directly addresses the

query's focus.

By integrating hybrid search with advanced workflows, the RAG pipeline is able to go beyond simply retrieving information to provide nuanced, actionable, and highly relevant responses.

5.6 Case Study: Student Admission Chatbot

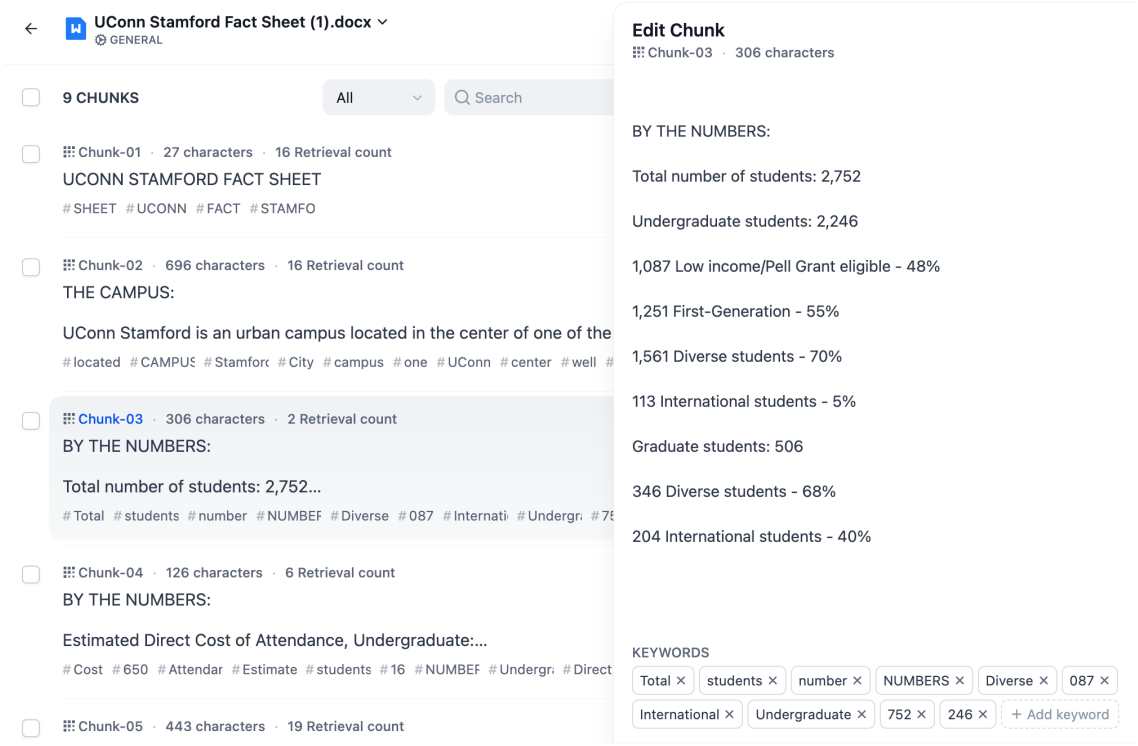

Figure 5.5: Screenshot showing chunks from one of the provided documents provided by the admissions office, to illustrate the chunk-based retrieval process in Stammy's RAG pipeline.

To illustrate the concepts discussed in this chapter, let's consider a case study of a student admission chatbot. Notably, this chatbot was developed in a single semester by four undergraduate students as part of their capstone project, even though they had no prior experience with GenAI and RAG systems. The chatbot, named "Stammy" by the team, was designed to assist students in their admission process by providing relevant information and answering their queries about the Stamford campus of the University of Connecticut.

Stammy uses the RAG pipeline to retrieve information. When a student asks a question, the chatbot first converts the question into an embedding vector, which represents the semantic meaning of the question. The chatbot then compares this vector with the pre-stored document embeddings in the vector database. The system retrieves the most relevant document chunks based on the proximity of their embeddings to the query embedding in the vector space. Figure 5.5 displays the document chunks from one of the documents provided by the

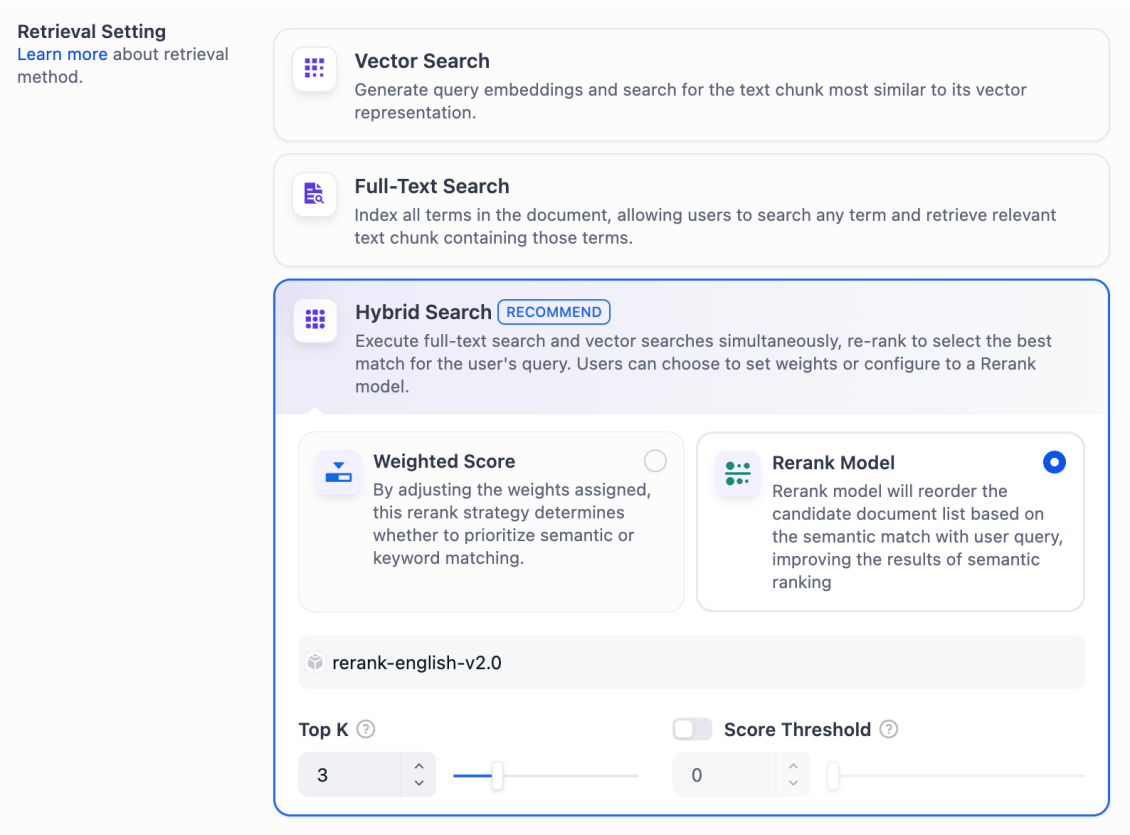

Figure 5.6: Screenshot illustrating the hybrid search setup in Stammy's RAG pipeline.

admission office. This semantic matching allows Stammy to locate relevant information even when the query and documents do not share identical keywords.

Stammy also uses hybrid search techniques that combine traditional keyword-based methods with embedding-based retrieval. This allows the chatbot to find semantically relevant content by capturing the underlying meaning of the text, while also ensuring precision by matching exact terms in the query. The chatbot's retrieval process was enhanced by introducing pre-processing and post-processing steps. Post-processing, for example, involves refining the retrieved results through *reranking*. Figure 5.6 illustrates the hybrid search and reranking setup in the RAG pipeline.

A key factor in the chatbot's strong performance was the significant effort the student team invested in maintaining and updating the underlying document collection. The original document collection was all scanned PDFs from the admission office. The team spent a lot of time organizing the documents into a structured format and ensuring that the chunks were well organized. Because the team ensured that the documents were current and well-organized, the chatbot was able to provide highly relevant and accurate responses to user queries.

When introduced to the admission office, Stammy generally performed well. However, the main limitation of the system arose when the provided documents were outdated. In such cases, the chatbot's responses could miss recent policy changes or new information,

highlighting the importance of keeping the knowledge base up-to-date for optimal RAG system performance.

5.7 Conclusion

In this chapter, we discussed the basic pipeline of a RAG system. We explained the key components of the RAG pipeline, including embeddings, vector databases, and the retrieval process. We also provided a case study of a student admission chatbot to illustrate the concepts discussed in this chapter.

Together, these components allow RAG systems to overcome the limitations of foundational language models by integrating real-time, dynamic information. This pipeline makes it possible for RAG-based AI systems to produce responses that are accurate, timely, and contextually relevant.

Exercises

Exercise 5.1. RAG does not address the hallucination problems of LLMs. (True/False)

Exercise 5.2. RAG is like an LLM taking an open-book exam. (True/False)

Exercise 5.3. Which of the following is NOT a step in the RAG process?

A) Retrieving relevant documents
B) Updating the prompt with retrieved context
C) Relying solely on pre-existing model knowledge
D) Generating an answer based on augmented prompts

Exercise 5.4. Embeddings in the RAG pipeline are:

A) Structured database tables
B) Dense vectors in semantic space
C) Random tokens with no meaning
D) Labels used for classification

Exercise 5.5. What does the distance between two embeddings in semantic space represent?

A) Visual size of the words
B) Similarity in their meanings
C) Frequency of usage
D) Word position in a sentence

Exercise 5.6. Vector databases are essential in RAG because:

A) They generate embeddings
B) They facilitate similarity search and fast retrieval
C) They replace fine-tuning
D) They eliminate hallucination completely

Exercise 5.7. Give two examples of vector databases commonly used in RAG pipelines.

Exercise 5.8. Scenario: A legal firm wants an AI assistant that can pull relevant clauses from a contract database when asked a question. Describe in 2–3 sentences how RAG would solve this problem.

Exercise 5.9. Which of the following best describes embeddings?

A) They are one-hot encoded vectors
B) They are dense numerical vectors capturing semantic meaning
C) They are manually designed linguistic rules
D) They are positional encodings in transformers

Exercise 5.10. Why is RAG considered more flexible than fine-tuning for enterprise use cases? (short answer)

Exercise 5.11. Explain in 2–3 sentences how RAG helps with domain-specific knowledge (e.g., finance or healthcare).

Exercise 5.12. Compare RAG with "closed-book" LLMs. List one key advantage and one limitation of each.