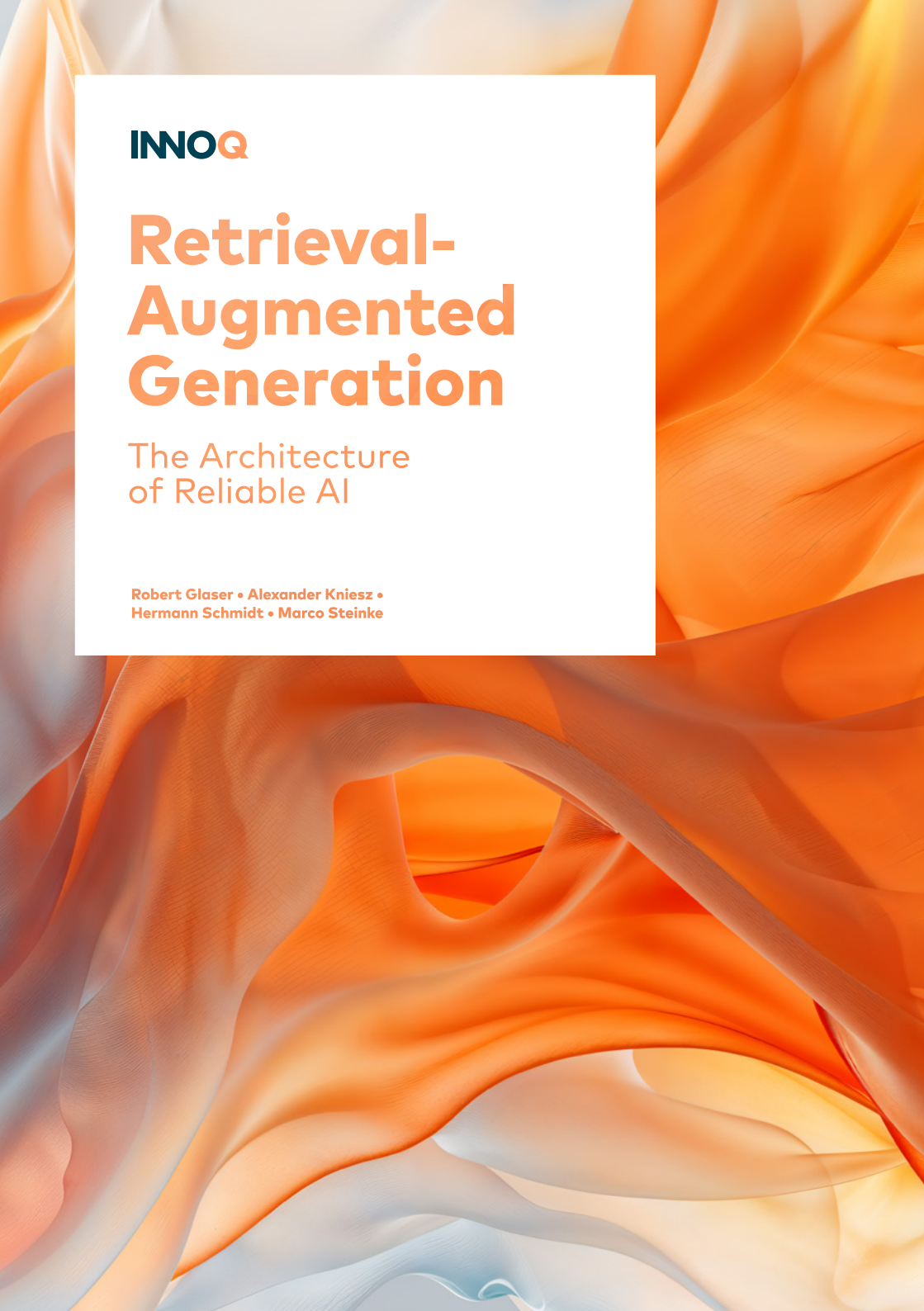

INNOQ
Retrieval-Augmented Generation
The Architecture of Reliable AI
Robert Glaser • Alexander Kniesz • Hermann Schmidt • Marco Steinke

RAG: Retrieval-Augmented Generation

The Architecture of Reliable AI

Robert Glaser
Alexander Kniesz
Hermann Schmidt
Marco Steinke

innoQ Deutschland GmbH
Krischerstraße 100 · 40789 Monheim am Rhein · Germany
Phone +49 2173 33660 · www.INNOQ.com

Layout: Tammo van Lessen with XeLaTeX
Cover: Murat Akgöz
Typesetting: André Deuerling

RAG: Retrieval-Augmented Generation – The Architecture of Reliable AI
Published by innoQ Deutschland GmbH
Edition 1 · November 2024

Inhaltsverzeichnis

1 Fundamentals of Large Language Models (LLMs)

Generative AI and Large Language Models (LLMs) have revolutionized the world of artificial intelligence in recent years. They represent not only a technological breakthrough as a new general-purpose technology but fundamentally change how we interact with computers and process information. For software developers and architects, understanding the functionality, strengths, and limitations of this technology is essential to make informed decisions about its implementation.

1.1 How do LLMs work?

At their core, LLMs are highly complex neural networks trained on enormous amounts of data. They utilize the Transformer architecture with its attention mechanism to capture and process extensive text relationships in parallel. An interactive explanation is available through the Transformer Explainer[1].

The term „Language“ in „Large Language Models“ can be misleading. Although LLMs were originally designed for language and text processing, their applications are significantly more diverse. Essentially, anything that can be fed into Transformer models in token form is suitable for processing by LLMs. This includes not only text but also:

- Images
- Video
- Spoken language
- Molecular structures (e.g., proteins)

This versatility makes LLMs powerful tools for a variety of applications that extend far beyond pure text processing.

The training process of an LLM can be simplified as follows:

[1] https://poloclub.github.io/transformer-explainer/

1. **Data collection**: First, massive amounts of data are gathered from various sources. This can include texts and other types of data that can be converted into tokenized form.
2. **Preprocessing**: The data is cleaned and converted into a uniform format. It is broken down into „tokens“ - small units that can represent words, word parts, image pixels, audio frames, or other elements depending on the data type.
3. **Training**: The model learns to predict the next token in a sequence. This occurs through repeated passes through the training data and adjustment of the model parameters.
4. **RLHF** (**Reinforcement Learning from Human Feedback**): After initial training, model behavior is further refined through RLHF. This process, also known as „alignment,“ uses human feedback to adjust the model to generate responses that better align with human preferences, values, and expectations. The goal is to bring the AI system more in line with human intentions.

The key to understanding LLMs lies in their ability to predict tokens. They don't learn individual facts but capture complex statistical patterns in data. This enables them to generate context-dependent and often surprisingly coherent outputs, whether in the form of text, images, or other data types.

1.2 Strengths and Limitations of LLMs

The strengths of LLMs are impressive:

- **Flexibility**: They can be applied to a wide variety of tasks without specific programming. In doing so, they often make previous Machine Learning (ML) approaches seem rigid and disrupt numerous traditional ML use cases.
- **Context understanding**: LLMs can capture the context of a request remarkably well and generate relevant responses.
- **Creativity**: They can generate new ideas and support creative tasks.
- **Ease of integration**: LLMs are straightforward for developers to incorporate, often through simple API calls.

However, LLMs also have significant limitations:

- **Lack of explainability**: Unlike rule-based systems, the non-deterministic nature of LLMs often makes it difficult to understand how they arrive at a particular output.
- **Resource intensity**: Training LLMs requires enormous computing power and energy. But: both training and inference [2] costs are coming down, as they are benefiting significantly from economies of scale.
- **Information currency**: LLMs are limited to the state of their training data and cannot access current or internal information without additional mechanisms such as RAG or *Function Calling*.
- **Hallucinations**: LLMs can generate plausible-sounding but factually incorrect information, especially when context is missing.

As a general-purpose technology, listing use cases across industries is both overwhelming and exciting[3]. Fundamentally, LLMs enable features that were previously too expensive or simply impossible to implement.

[2] During inference, an LLM generates responses one token at a time–much like someone speaking spontaneously without pausing to deliberate extensively. This process, often referred to as „test time", involves predicting each subsequent token based on the context provided. Newer models like OpenAI's o3 employ an internal reasoning mechanism called a chain-of-thought (CoT), where additional tokens are generated as intermediate steps to refine the final answer. Depending on the vendor's implementation, these CoT tokens may be folded away (kept hidden), summarized into the final response, or output raw within special markers for debugging or transparency purposes. In essence, regardless of the specific handling of these internal tokens, all modern language models rely on sequential token generation to produce coherent and contextually appropriate responses.

[3] https://www.innoq.com/en/articles/2024/10/generative-ai-in-business-software/

1.3 Comparison of Different Approaches

It's important to consider LLMs in the context of other AI approaches:

1. **Rule-based systems**:
   - Advantages: Predictability, explainability, low resource requirements
   - Disadvantages: Limited flexibility, labor-intensive manual maintenance
2. **Traditional Machine Learning models**:
   - Advantages: Efficient for specific tasks, often more interpretable than LLMs
   - Disadvantages: Typically require manual feature engineering and elaborate data preparation, less flexible
3. **Large Language Models**:
   - Advantages: High flexibility, ability to generalize, natural language processing
   - Disadvantages: High resource requirements, limited explainability, potential hallucinations

The choice of the right approach depends heavily on the specific task, available resources, and requirements for explainability and control.

1.4 Misconceptions and Clarifications

A common misconception is that LLMs „understand" what they output. In reality, they operate on statistical patterns without genuine understanding in the human sense. This can lead to unexpected results when the model encounters statistically unfamiliar or ambiguous contexts.

Another critical point is the potential bias in LLMs. Since they are trained on existing data, they can reproduce societal prejudices and inequalities. Leading providers like OpenAI or Anthropic employ sophisticated RLHF processes to minimize this issue, but bias cannot be completely eliminated. This remains a challenge, especially with smaller or specialized models.

1.5 Conclusion and Outlook

LLMs represent a significant breakthrough in AI but also bring new challenges. Their effective use requires a deep understanding of their functionality, strengths, and limitations.

A central question many organizations face is: How can we leverage the capabilities of LLMs with our internal data? This leads us to the concept of Retrieval-Augmented Generation (RAG), which we will examine in more detail in the following chapters.

2 Retrieval-Augmented Generation: When World Knowledge Meets Specialized Knowledge

In the first chapter, we learned about the basic functionalities and strengths of LLMs. While LLMs have enormous potential to answer generic queries based on their comprehensive knowledge acquired during training, they show weaknesses when dealing with current, specialized, or verified information. This is precisely where Retrieval-Augmented Generation (RAG) comes in – an approach that combines the language capabilities of LLMs with access to dynamic and specific data sources. This allows world knowledge to be connected with specialized knowledge to deliver contextual and precise answers while also documenting which data was used to respond to a prompt.

2.1 The Limitations of LLMs and RAG as a Solution

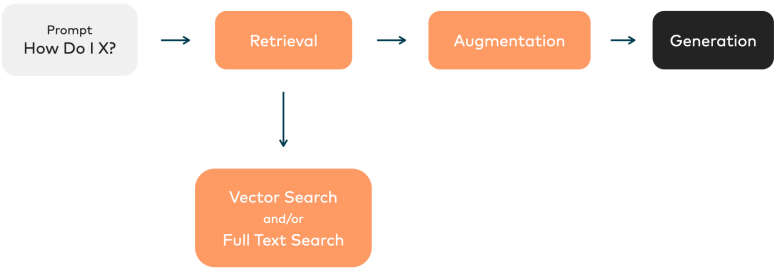
LLMs acquire their knowledge from a training dataset and can therefore only retrieve information that existed up to their last training date. In dynamic environments where up-to-date information and expertise are crucial, this is a major limitation. RAG offers a solution by enabling LLMs to incorporate information from external and current data sources. This connection to verified knowledge increases the accuracy and timeliness of the generated content.

2.1.1 What is RAG?

Retrieval-Augmented Generation combines the language capabilities of an LLM with a retrieval system that can access relevant data and make it available to the model. The process works like this:

1. **User query**: A user submits a query that may require current or specific information.
2. **Retrieval**: A retrieval system searches defined external sources, such as databases or knowledge repositories, and finds matching documents or text passages.
3. **Augmentation**: This relevant information is passed to the LLM and serves as the contextual basis for generating the response.
4. **Generation**: The LLM uses the additional data to create a well-founded, contextualized answer.

Through this combination, RAG can deliver answers that contain not only general knowledge but also highly specific and current information – a significant improvement over pure LLMs. [1]



[1] It's actually an illusion that the LLM gains additional knowledge through *in-context learning*. The provided context influences the calculations in the *attention mechanism* of the Transformer network. Nevertheless, an LLM cannot calculate anything it doesn't know. Due to the enormous size of LLMs, the supply of patterns is virtually inexhaustible and can hardly be pushed to its limits by context from typical business data. What we perceive as hallucinations is the result of misguided calculations. RAG provides additional context for more stable, targeted calculations in our subject area. This creates the illusion that the LLM has „understood" our data.

2.2 Grounding: Anchoring in Verified Data

A key advantage of RAG is what's known as „grounding" – anchoring answers in verifiable data sources. This makes the generated knowledge more precise and reliable since the answers are based on explicit data sources that users can trace. This anchoring is crucial in fields like medicine, science, and law, where accurate and verifiable information is essential. With RAG, generated answers can be based on a verifiable data foundation, enhancing the reliability and quality of the information.

2.3 The Motivation Behind RAG

The development of RAG stems from several important factors:

1. **Currency**: RAG allows LLMs to access current information through external sources. Organizations that frequently produce new content – such as research reports or market analyses – can always incorporate up-to-date data into their answers through RAG.

2. **Specialized Knowledge**: Many companies possess internal knowledge that is essential for specific use cases. Through RAG, this specialized knowledge can be incorporated into answers, increasing the relevance and applicability of the generated responses in professional contexts.

3. **Trustworthiness**: With RAG, it's possible to track which sources have been incorporated into an answer. This is particularly valuable in critical scenarios where accuracy and reliability are decisive.

4. **Efficiency and Scalability**: RAG makes it possible to efficiently utilize specialized knowledge without having to constantly retrain the underlying LLM. Additionally, only relevant information is passed to the LLM, which reduces costs and response time.

2.4 Conclusion: An LLM with Reliable Data

RAG bridges the gap between the generic capabilities of an LLM and the requirements for updated, specialized knowledge. By combining LLMs with dynamic, verified data sources, the model becomes both a comprehensive knowledge provider and a specialized advisor. This creates a system that can deliver not only informative but also more well-founded and contextualized answers – a crucial improvement for many professional and industrial applications.

In the upcoming chapters, we will delve deeper into the components and practical application of RAG and demonstrate how to implement Retrieval-Augmented Generation and use it effectively in enterprises.

3 Document Ingestion

For Retrieval-Augmented Generation to be implemented, we must first prepare selected data, convert it into an appropriate format, and then make it available to the LLM in the correct format. Document Ingestion is a central step in the Retrieval-Augmented Generation (RAG) process, as it forms the foundation for accessing external data sources. The quality of answers that a RAG system can provide depends significantly on how carefully the document ingestion step is implemented. In this chapter, we'll explore how documents are prepared for later use, what challenges need to be overcome, and why the so-called „chunking" plays a decisive role.

3.1 What is Document Ingestion?

Document Ingestion is the process of collecting, preparing, and storing documents so they can be made available to a retrieval system. These documents can come in various formats and contain different types of content, including PDFs, websites, database entries, technical documentation, research reports, or FAQs. The goal of document ingestion is to transform these diverse information sources into a structured, searchable form that the retrieval system can efficiently and precisely query.

3.2 Why is Document Ingestion so Important?

The quality of answers generated by a RAG system largely depends on how well the underlying documents are prepared and structured. Poor document preparation can result in missing important information or the retrieval system struggling to find relevant content. Therefore, document ingestion must be carefully planned and tailored to the type and structure of the available documents.

Example: To answer questions as precisely as possible, we might divide a book differently depending on how information is distributed: either page by page

when relevant content is compact, or chapter by chapter when information spans multiple pages.

A key challenge is the heterogeneity of documents. Different document types can have varying structures and contents. A scientific article, for example, is organized into paragraphs, headings, and citations, while technical documentation might consist of tables, code snippets, and detailed step-by-step instructions. Processing all documents using the same approach would therefore be ineffective. This is where „chunking“ comes into play.

3.3 Chunking: The Right Granularity for Success

„Chunking“ refers to the process of breaking documents into smaller, contextually coherent sections (chunks). The size and structure of these chunks is crucial as they form the units that the retrieval system later accesses. The challenge is finding the optimal granularity that provides enough context to the retrieval system without having to search through excessively large blocks of information.

- **Why is chunking so important?** The accuracy of subsequent answers depends heavily on the relevance and precision of the chunks returned for a query. If we break documents into chunks that are too large, we risk including irrelevant information, which dilutes the generated answer. If we break them into chunks that are too small, the necessary context for a precise answer may be missing.

- **Adaptation to document structure:** A crucial aspect of chunking is adapting to the structure of each document type. For a scientific article, chunks might be individual paragraphs or thematic sections. For technical documentation, chunks could be individual steps in a guide or descriptions of specific functions. There’s no universal „one-size-fits-all“ approach to chunking. Instead, we must tailor it to the specific document type and planned use cases.

- **Dynamic chunking:** In some cases, implementing dynamic chunking may be beneficial, where the granularity varies depending on context and task. An initial classification of document types and structures can help automatically

guide the document ingestion process. Such a flexible approach increases retrieval precision by better considering context. For example, the same ingestion pipeline could split PDFs into pages while dividing HTML files into paragraphs. Chunks from different documents are processed in specific ways to prepare them for production use.

3.4 Proper Structuring: More Than Just Text

Another critical aspect of document ingestion is the proper structuring of content. Not all information in a document is equally important or relevant. Headings, bullet points, tables, and highlights often provide important clues about a chunk's content focus. Therefore, extracting and preserving metadata and structural information during document ingestion is important.

- **Metadata:** Information such as document title, author, creation date, original page number, and keywords provides the retrieval system with valuable hints about which chunks are particularly relevant for a specific query.

- **Content indexing:** Beyond breaking content into chunks, indexing is an essential step. Each chunk is assigned a unique identifier as well as relevant keywords and contextual properties. This index enables the retrieval system to search quickly and precisely for the most relevant content, such as by URI or page number.

3.5 Challenges and Best Practices

- **Heterogeneous data sources:** One of the biggest challenges is handling a variety of different document types and formats. PDF files, HTML pages, CSV files, database entries – all these formats require different processing strategies. The document ingestion process should use good conversion tools and parsers that change data into a standard, searchable format. Alternatively, it can use appropriate databases that make different document formats easy to search.

- **Quality control:** Since chunk quality directly influences the quality of generated answers, regular verification of document ingestion is essential. It's advisable to set up an automated process for validation and quality control that ensures document ingestion works consistently and correctly.

- **Long-term maintenance:** Document ingestion is not a one-time process. As information constantly changes and new documents are added, it must be continuously monitored and adjusted. Automated update and monitoring mechanisms ensure that the database remains current, allowing the RAG system to always access up-to-date, relevant information.

3.6 Conclusion: The Foundation for High-Quality Answers

Document ingestion is a fundamental and complex step in the RAG process that decisively influences answer quality. Through careful structuring and preparation of documents, adapted to their individual characteristics, we lay the foundation for efficient and precise retrieval. Chunking in particular is a critical factor: it determines how much context is available to the retrieval system and how precisely information can be extracted.

Well-designed document ingestion ensures that the RAG system can draw from a broad and diverse information base, delivering reliable, precise, and contextualized answers. The quality of document ingestion is therefore a central building block for the success of a RAG architecture and should be implemented with the utmost care and expertise.

4 Retrieval

4.1 The Role of Retrieval

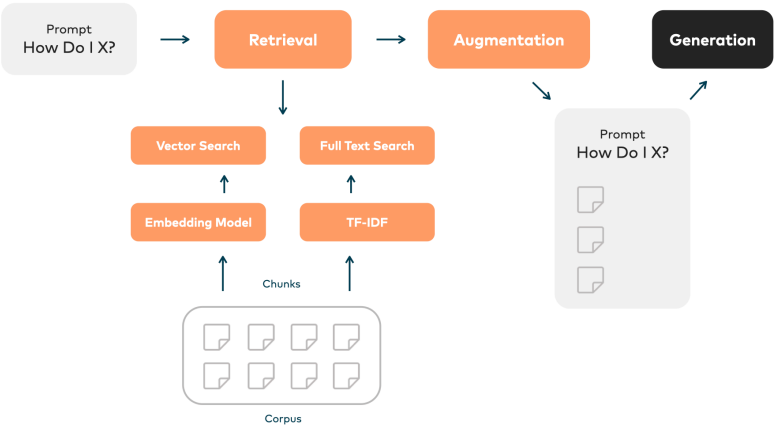
The *Retrieval* step finds relevant chunks for a prompt. These chunks were created during ingestion and are indexed in a database.

- Input: Prompt text
- Output: List of chunks (text sections)

Retrieval must balance thoroughness, runtime, and cost. In an interactive application, feeding the LLM with 100 chunks of 1,000 tokens each isn't helpful if it leads to minute-long wait times. Resources are limited.

Retrieval techniques are rapidly evolving in the LLM space. We must distinguish between theoretical *proposals* and *practical solutions*. We'll focus on techniques that work in practice and are easy and cost-effective to implement.

Retrieval itself isn't new—search has always existed. What's new is vector search, which builds on language model concepts.



4.1.1 Prompt versus Query

In retrieval contexts, we refer to the query rather than the prompt. In the simplest case, they're identical, but this approach alone isn't sufficient for effective retrieval.

4.2 Vector Search

Nearly all technical articles on RAG cover vector search, also known as *Semantic Search*.

The process involves calculating an *embedding vector* from the query and passing it to a vector index as a search value. The index calculates the semantic similarity between the search vector and the embeddings of the chunks, delivering the best matches. Similarity measures like *dot product* or *cosine similarity* can be used.

The specific similarity measure isn't critical—what matters is that the index provides a list of chunks sorted by similarity.

We refer to this similarity as a *score*. A high score suggests potentially high relevance.

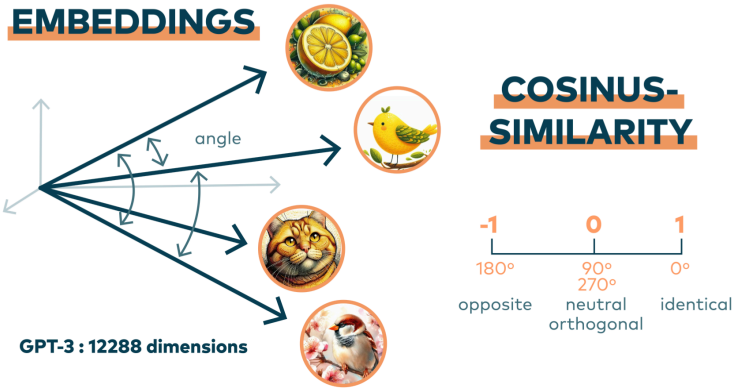
4.2.1 Characteristics

Vector search always returns results, unlike full-text search or traditional database queries.

For cosine similarity, these theoretical values are noteworthy:

- -1: Search vector and comparison vector are perfect opposites
- 0: Search vector and comparison vector have nothing in common
- +1: Search vector is identical to the comparison vector

In practice with document chunks, values typically range between 0.5 and 0.87. Even questions seemingly unrelated to the topic will fall somewhere in this range.



4.2.2 Misconception: More is Better

Some try to maximize vector search scores at all costs, believing this finds more relevant chunks. One approach is *Query Rewriting*, where an LLM expands a simple prompt into a longer, more detailed query.

This naturally increases scores since there's more material in the query to match content. However, it doesn't necessarily deliver more relevant chunks—often the opposite occurs. By „fattening" the query, terms and concepts not in the original question are added. Are these even relevant? Additionally, the LLM creates slightly different content each time, making search results inconsistent for identical inputs.

We therefore use the **unchanged prompt as the query**. *Query rewriting* may be valuable if your application has relevant context (e.g., about the current user).

The absolute scores aren't important—what matters is the relevance ranking of the matches. The best chunk should be first, the second best next, and so on. Whether the best score is 0.87 or 0.73 is irrelevant.

4.2.3 Misconception: You Need a Minimum Similarity

It might seem logical to filter results based on a minimum similarity threshold. This assumption is incorrect. As explained earlier, the absolute value doesn't matter. For one query, 0.75 might be ideal, for another 0.80. There's no universal threshold. **Simply limiting the number of results is more effective**.

4.2.4 Misconception: It's All About Vector Search

Vector search is valuable as a supplement, especially for general questions, but it's insufficient alone. It must be complemented by full-text search—a point that most technical literature fails to emphasize clearly. This leads us to our next topic.

4.3 Hybrid Search

Vector search requires support from full-text search. In fact, full-text search should be *the primary search method*!

The two approaches complement each other. Full-text search excels with specific questions and compensates for embedding vagueness.

Full-text search is a mature technique that needs little explanation. The common approach uses **TF-IDF** with **BM25**, implemented by indexers like Lucene. Similar to vector search, a full-text index scores results for a search text and delivers the best matches. Unlike vector search, text search can return empty results.

4.3.1 Multi-stage Text Search Strategy

We employ a multi-stage search strategy that extracts details and alternative terms from a prompt to form multiple queries.

Stage 1: Prompt = Query

First, the prompt is passed unchanged to the search index.

Stage 2: Keywords, Compounds, Synonyms

- Keyword extraction: The LLM extracts keywords from the prompt, considering professional context to recognize relevant terms.
- Compound words: The LLM breaks down compound words into components (e.g., „firefighter“ becomes „fire“ and „fighter“).
- Synonyms: The LLM generates synonyms for all collected words.

These collected keywords are then submitted to the search index.

4.3.2 Full-text Search Takes Priority

If text search returns too few results, we skip the vector search. The likelihood of finding relevant chunks through semantic search is then practically zero.

4.3.3 Rank Fusion

The two-stage text search and vector search together produce three lists of chunks. Since scoring differs between text and vector searches, we can’t simply combine and sort them. Instead, we use *Reciprocal Rank Fusion*.

The exact formula is readily available elsewhere. The principle is that only the rank order in a list matters. The top match has rank 1 in its list. During merging, each occurrence of a chunk in a list increases its overall rank. Chunks appearing in multiple lists effectively rise to the top.

4.3.4 Top Chunks

After rank fusion, we limit the number of chunks. Our experiments show that a maximum of 30 chunks is reasonable—larger quantities didn’t improve results.

4.4 Overlaps or Windowing

Experiments revealed that the LLM sometimes prefers to access the chunks before or after a match—occasionally even favoring these over the match itself. The match essentially points to valuable surrounding content.

After ranking, we add one chunk before and two chunks after the match to the list. We apply a filter so only chunks with a cosine similarity to the match of > 0.82 are included. This threshold requires experimental determination. Furthermore, only the top 10 chunks undergo this process.

4.5 Limitation

For interactive applications, we limit LLM input to approximately 25,000 tokens. This equals about 40 pages of densely written PDF content. With the OpenAI API using GPT-40 (Oct. 2024), this results in roughly 15-second runtimes for detailed answers.

4.6 The LLM Has the Final Say

In practice, the LLM in the generation step has its own „opinions“ about relevance when processing chunks with the prompt.

In our experience, the top 10 chunks are generally preferred, but beyond that, the pattern becomes unpredictable. Surprisingly, chunks ranked around position 20 are sometimes selected. The distribution varies completely with each prompt.

Sometimes vector search high-scorers are preferred; other times text search results prove more relevant. **This observation led us to combine complementary search strategies rather than focusing exclusively on vector search optimization**.

4.7 General Limitations of Retrieval

Retrieval can only provide a subset of chunks. It cannot perform comprehensive searches like database queries.

Questions like „find all...“ or „what is the summary of...“ are inevitably answered partially. Including more chunks for broad questions creates unnecessary burden for specific queries.

We attempted to classify questions as „broad“ or „narrow,“ but this wasn't reliably effective. Furthermore, this approach merely shifts the limitation without providing complete answers.

Retrieval cannot perform aggregation.

4.8 Simple RAG is Just an Intermediate Step

Retrieval is clearly inefficient. We send material to the LLM via *force feeding* in hopes that it's relevant.

The next development stage will allow the LLM to decide whether it needs retrieval. This represents an *Agentic Workflow* requiring *Function Calling*, which could theoretically make retrieval dynamic and on-demand. Whether this works reliably remains uncertain.

An LLM doesn't know what it doesn't know. Force feeding bypasses this problem at the cost of efficiency.

4.9 Contextual Retrieval

Anthropic introduced *Contextual Retrieval*[1]. This technique „fattens“ all chunks with additional context during ingestion. The search process is then duplicated by using both the additional context and the chunks. Otherwise, the process follows the methodology described above.

[1] https://www.anthropic.com/news/contextual-retrieval

4.9.1 Ingestion

An LLM receives the entire document and a chunk as context. It's instructed to classify the chunk content in the document's context. The result is included in the embedding alongside the chunk. Anthropic uses the prompt:

```
Please give a short succinct context to situate this chunk
within the overall document for the purposes of improving search
retrieval of the chunk.
```

(Source: Anthropic Cookbook[2] on Contextual Retrieval)

This additional context references the chunk and is stored and indexed similarly.

4.9.2 Retrieval

Retrieval is performed identically for both the chunks and the additional contexts. Anthropic also employs hybrid search with full text (BM25) and vector. After rank fusion and top selection, an optional reranking model may reorder the chunks by relevance.

4.9.3 Considerations

- During ingestion, the entire document and chunk are placed in context, which can be problematic depending on the model and material.
- Embedding models have limited context sizes. The embedding processes both the chunk and extra context. With PDF pages as chunks, content-rich pages may approach maximum context size. The extra context might exceed this limit.
- Generating additional context can significantly increase processing time.

[2] https://github.com/anthropics/anthropic-cookbook/tree/main/skills/contextual-embeddings

5 Augmentation

5.1 The Role of Augmentation

The term means *enhancement*. The LLM's context is enhanced with the chunks found during the retrieval step. This is where the complete message list that will be passed to the LLM is prepared.

Input:

- List of chunks
- System message
- Prompt
- If applicable, list of messages from a previous dialogue

Output:

- List of messages to the LLM (the context) consisting of system message, chunk texts, previous dialogue, last prompt

5.2 Incorporating Chunks

There are various recommendations on where and how chunks should be incorporated. With OpenAI models, we've had good results placing them in the *system message*. For Anthropic, it's recommended to use the first *user message*. Generally, Anthropic provides more specific formatting guidelines than OpenAI. When in doubt, follow their recommendations.

We incorporate chunks as follows:

```
<data>
[document_id:sampledocument.pdf]
[pagenumber:123]
...additional metadata...
```

```
Here begins the text of the chunk.
</data>
```

The XML tags work just as effectively as triple backticks “‘. The metadata can be referenced in instructions in the system message to provide precise citations. This task is actually more challenging than it appears.

5.3 Document References

For our use case, we needed to provide exact page numbers for references. That’s why our chunking was based on PDF pages. The more common chunking methods, which don’t account for page boundaries, weren’t suitable for our needs.

Crafting a reliable instruction in the system message proved difficult. Even minor changes in wording had significant effects. We eventually settled on the following format:

```
IMPORTANT: Use references to the source everywhere
in the format [id:document_id,page:pagenumber].
Write the references directly after the statement
where the source is used.
Each paragraph must contain at least one reference.
Each statement must include a reference.
Always include file extensions with document_id.
Use the original text of the document_id.
Leave ae, oe, ue unchanged.
```

This format works consistently with GPT-4o. GPT-4o mini tends to lose track of the task when dealing with larger chunk lists.

5.4 Without References Everything Is Flying Blind

How can you determine if the chunks were ultimately relevant to the LLM if you don't generate references? This feedback is crucial! It allows us to fine-tune our retrieval process. Without references, you're essentially flying blind. It also doesn't help to manually evaluate the chunks yourself. As mentioned in the retrieval chapter, the LLM has the final say. **What constitutes relevant versus irrelevant content isn't intuitively obvious**.

5.5 Prompt Rewriting

Consider a scenario where someone enters only a keyword, such as „present value." What should the LLM do with that? We first task the LLM with checking whether the prompt is a complete sentence. If not, we have the LLM rewrite the prompt into a question that incorporates our technical context. This rewritten prompt works behind the scenes and isn't shown to users.

5.6 Keep Chunks or Not?

Should we retain the chunks collected during the previous retrieval? We believe not, for several reasons:

- The dialogue already contains the LLM's response to the prompts. You can reference previous content without keeping the underlying chunks.
- Managing context size becomes complicated if chunks aren't discarded. Which ones should you keep? What happens when the user changes topics?
- Each new pass becomes slower and more expensive.
- The LLM often ignores chunks from retrieval when the prompt refers to the dialogue history. For example, with a prompt like „Make a PowerPoint slide outline from this for me."

As is often the case, it's unwise to make assumptions about how the LLM works. You'll frequently be mistaken.

6 Generation

Generation is the third stage in a RAG architecture and helps expand the boundaries of traditional natural language processing (NLP). While the retrieval part extracts relevant information from sources, generation transforms this information into coherent, fluent, and contextually appropriate text outputs.

Basic Principles of Generation

Generation in a RAG system uses an LLM with Transformer architecture. These Transformers are specifically optimized to create human-like text. The most well-known example is GPT (Generative Pre-trained Transformer), which has learned to reproduce both syntax and semantics of natural language through extensive pre-training on large text corpora. The strength of such a model lies in its ability to fill knowledge gaps with plausible information – a capability enabled by finely tuned language modeling mechanisms.

Text Generation Process

The generation process begins with input that, in RAG systems, comes from the retrieval component and is combined with a system prompt during augmentation to form a final prompt. This input is generally called a *prompt*. Different types of prompts exist, such as system and user prompts, which the LLM processes differently. For clarity, we'll refer to these as messages. A prompt therefore consists of various messages.

In RAG, the prompt has two main components: text passages from retrieval that provide context, and the system message containing instructions for the model. These instructions might specify that the LLM should only use information from the provided text passages.

The LLM uses this prompt to establish an initial state for text production. Most modern LLMs like GPT-4o have been specifically trained for this user-LLM interaction and understand how to appropriately weight different statements.

From this initial state, text generation proceeds token by token. The system continuously considers previously generated text to ensure it creates an understandable and coherent output.

Challenges and Optimization

A key concern in generation is ensuring quality and relevance. Several techniques help with this, including *beam search*, which evaluates multiple possible text continuations. *Top-k sampling* limits selection to only the most probable tokens for the next position. The more widely known *temperature adjustment* controls how creative versus predictable the word choices should be. These methods help balance creativity and relevance by intelligently constraining the selection of possible words, thus avoiding inappropriate or uninteresting outputs.

Important note: While an LLM can be configured to avoid undesirable outputs, there are no absolute guarantees—the model might still generate such content despite precautions.

However, more reliable safeguards can be implemented through *guardrails* around the LLM. These include input and output guardrails that protect both the LLM from users and users from potentially problematic LLM outputs.

Integration into the Business Environment

In business contexts, the adaptability of generation systems to specific needs is particularly valuable. RAG represents one good approach for this. Another option is fine-tuning, which can adapt the model to correctly use industry-specific terminology. This process offers significant advantages for applications like automated customer service, marketing content creation, or personalized user interactions. However, fine-tuning requires caution, as it permanently adjusts the model's weights. These adjustments can only be modified through additional, iterative fine-tunings, which can become a considerable cost factor, both initially and over time. In environments where the underlying data changes frequently, a RAG system is typically recommended. These approaches aren't mutually exclusive and can be combined when necessary.

7 Use Cases

Customer Support

In the field of customer support, RAG systems represent a transformative approach by enhancing traditional chatbots with the ability to retrieve and generate contextually relevant and well-structured responses. For example, in the automotive industry, a RAG system can access extensive workshop knowledge and technical manuals. When a customer inquires about specific troubleshooting solutions or maintenance procedures, the system retrieves relevant data from its repository. Combined with an LLM, this produces tailored answers that effectively address support issues. This capability significantly improves the quality of customer interactions and satisfaction while simultaneously reducing resolution times.

Unlike conventional rule-based systems, which often fall short when handling nuanced questions or complex queries, RAG-powered solutions adapt by retrieving relevant information and presenting it concisely. This adaptability is crucial in areas such as e-commerce or technical support, where products and services evolve rapidly and knowledge bases must be continuously updated to remain relevant.

Document Processing in Logistics

Logistics companies routinely handle vast quantities of documents including invoices, shipping manifests, and compliance forms. LLMs and RAG systems can significantly streamline document processing workflows by automating the extraction and interpretation of relevant data. These systems can quickly analyze diverse documentation sources to identify key shipping details, optimize data processing procedures, and enhance the accuracy of logistical operations. By increasing processing speed, they reduce manual effort and minimize human error, enabling more efficient supply chain management.

Medical Applications

In healthcare, information is critical for timely and accurate decision-making. RAG can play a vital role by providing physicians and nursing staff with access to current information such as drug interactions or clinical research findings. For

example, when medical professionals inquire about potential drug interactions, a RAG system can immediately deliver precise, comprehensive, and scientifically supported information from the latest medical research and databases. This facilitates informed decisions, improves patient management, and ensures better care outcomes.

Considerations for Non-Recommendation

While RAG offers numerous benefits across various domains, it is not a universal solution. Its dependence on retrievable data makes it unsuitable for environments requiring immediate real-time processing and decision-making. Scenarios such as air traffic control or financial trading demand rapid responses based on continuous, second-by-second data and complex decision-making capabilities that RAG cannot provide.

Furthermore, RAG's effectiveness depends on the quality and scope of underlying databases. In contexts requiring high interpretive abilities or creative decision-making, such as strategic management or artistic creation, RAG might prove insufficient. In highly sensitive environments with strict data protection requirements, caution is warranted, as data retrieval components may introduce potential security risks.

In summary, RAG systems excel in scenarios requiring the retrieval and generation of contextually accurate information from text-based resources. Their integration into customer support, logistics, and healthcare demonstrates their potential to transform data-intensive processes.

When implementing RAG, it's essential to carefully evaluate each use case. Legal data protection requirements and the quality of available data should be thoroughly considered. A well-designed proof-of-concept can help determine whether RAG is the optimal solution or if simpler alternatives would suffice. Through targeted testing and analysis, organizations can ensure RAG systems are deployed where they deliver maximum value.

8 Technical Challenges in Implementation

Implementing RAG comes with various technical challenges. These range from effectively managing large volumes of data, optimizing retrieval and generation processes, to questions of scalability and data security. In this chapter, we examine the key technical hurdles that can arise when implementing a RAG system and provide guidance on how to overcome these challenges.

8.1 Setting Up and Operating Vector Databases

A central component in the RAG approach is the retrieval system, which can be implemented using vector databases and relational databases. Vector databases are specifically designed for storing and quickly querying high-dimensional vectors, such as those created by embeddings. The following challenges arise:

- **Storage requirements**: As embeddings are created for each document in the database, storage needs can quickly grow. Selecting a database that scales well and offers efficient storage management is therefore essential.

- **Query speed**: Fast response times during retrieval are crucial for finding relevant documents in real-time. Techniques like *Approximate Nearest Neighbor (ANN)* help accelerate searches, though at the cost of accuracy. Instead of exact matches, only the closest entries from the vector database are retrieved.

- **Indexing and updating**: Documents are continuously added and updated. This means that the vector database must be regularly refreshed to always provide current information. Additionally, outdated documents must be removed from the vector database.

8.2 Challenges in Creating Embeddings

RAG uses embeddings to map the semantic content of documents, creating the foundation for the retrieval process. The following challenges arise:

- **Quality of embeddings**: Different models and API providers offer varying embedding qualities. Choosing the right embedding model is crucial for obtaining the most accurate representation of documents.

- **Consistency**: To find relevant documents for a prompt, both the documents in the vector database and the prompt must be processed with the same embedding model. If the embedding model changes, all entries in the database must be re-indexed with the new model.

- **Performance and API costs**: Creating embeddings through model provider APIs incurs costs. Smaller embedding models cost significantly less but are also much less effective at retrieving relevant documents. It's necessary to determine which model offers acceptable costs without compromising retrieval quality.

8.3 Combining Retrieval and Generation

One of the biggest challenges in implementing RAG is effectively incorporating retrieved information into the generation process. This requires seamless integration of the „Retrieval" and „Generation" modules with respect to the following aspects:

- **LLM context size**: All LLMs have a limited context length. If the retrieved documents or text fragments exceed this length, relevant content must be selected or summarized. An intelligent selection strategy is crucial here to avoid overloading the model with irrelevant information and unnecessarily increasing costs.

- **Model limitations**: The LLM should use the received information to generate an answer. However, the LLM doesn't know whether the information is actually suitable or what additional information might be missing.

- **Controlling answer quality**: The quality of the generated answer heavily depends on the relevance of the retrieved information. Testing and adjusting retrieval parameters are often necessary to obtain consistent and reliable answers.

8.4 Scalability and Performance Optimization

Processing large volumes of data and integrating real-time queries pose significant challenges to the scalability and performance of a RAG system:

- **Throughput**: In heavily used applications, query frequency in the vector database and generation load on the LLM can increase dramatically. Load balancing and providing sufficiently scalable resources are essential to ensure consistent response times. Running the models in-house is challenging, as LLMs require specialized hardware.

- **Caching mechanisms**: To save resources and increase speed, a caching system can be implemented to store frequently requested or similar answers. This reduces the need for repeated queries and generations.

- **Optimization of API requests**: Using APIs like the OpenAI API for generation and embedding creation comes with costs that can be reduced through optimized requests. Techniques such as batch processing and caching can also contribute to cost reduction.

8.5 Data Management and Security

RAG systems often access sensitive or protected information, especially in corporate applications that tap into internal data sources. Various challenges in data security arise:

- **Access controls**: It's essential to ensure that only authorized users have access to the retrieval system and the processed data. Finely tuned access management is crucial to protect sensitive data.

- **Compliance and auditability**: Depending on the industry, data is subject to specific compliance requirements. A RAG system must ensure that all retrieved and generated data complies with applicable regulations and can be audited if necessary.

8.6 Bias and Quality Control

Although RAG offers a way to base LLM answers on verified information, quality control remains a challenge:

- **Bias in generation**: Since LLMs are based on statistical patterns, they can reproduce societal biases. RAG can partially mitigate this through verified external data, but continuous monitoring and fine-tuning are necessary.

- **Evaluation of answer quality**: The quality and accuracy of generated answers are critical. Regular testing, such as through quality metrics or human feedback, is necessary to adjust and improve the system. It's also helpful to compare answer quality for different interpretations of the system prompt and reformulations of user queries, an approach known as „Prompt Evaluation."

8.7 Cost and Resource Management

Implementing a RAG system can consume significant computational and financial resources. The following aspects are important to control costs:

- **Efficient use of cloud resources**: Many RAG systems use cloud services for managing vector databases and generating answers. Demand-based provisioning and serverless approaches can help reduce operating costs.

- **Monitoring and cost control**: Comprehensive monitoring is crucial to continuously track performance and costs. Tools for monitoring API usage and optimization strategies contribute to cost reduction.

8.8 Success Factors and Recommendations

The technical implementation of a RAG system comes with several challenges that can be overcome through careful planning and selection of appropriate tools and strategies. The key success factors are:

- **The right embedding model** for fast and relevant results.
- **Deep domain knowledge** to recognize which documents are truly relevant and to retrieve them for matching queries.
- **Integration of a robust security concept** to protect sensitive data.
- **Regular quality control and fine-tuning** to minimize bias and improve answer quality.

9 Our Offering

9.1 Development, Consulting and Operations

Generative AI enables your business to implement features that were previously impossible or cost-prohibitive, and to digitize manual processes more rapidly.

IT is an enabler for business. It makes sense to leverage generative AI to streamline repetitive and time-consuming activities to maintain delivery capabilities. Throughout this process, humans always remain „in the loop."

We're happy to support you with your development projects—from initial exploration, through product development, to all operational aspects.

More information: https://www.innoq.ai

9.2 Training: GenAI with RAG for Developers

In this training, we explore how to enhance LLMs with your organization's proprietary data, focusing on the Retrieval-Augmented Generation (RAG) architecture. The goal of this course is to familiarize you with all components of a basic RAG architecture. Through step-by-step experiments, we introduce each component individually. By the end of the training, you'll create a fully functional chatbot that can work effectively with your own documents.

More information and dates: https://www.socreatory.com/de/trainings/genAI

We provide honest consulting, innovative thinking, and passionate development. The result: Successful software solutions, infrastructures, and business models.

As a technology company, we specialize in strategy and technology consulting, software architecture and development, methodology and technology training, and platform infrastructures.

With over 160 employees across locations in Germany and Switzerland, we help companies and organizations conceptualize and implement complex projects and enhance existing software systems.

We actively contribute to open source projects and the iSAQB e.V., and share our knowledge and experience at conferences and meetups, as well as through numerous books and technical articles.

Visit us: **www.innoq.com**

The Authors

Robert Glaser

Robert Glaser leads Data and AI at INNOQ. With roots in software engineering and a passion for creating user-friendly web applications, he now guides companies through the AI landscape, helping them develop strategies and products for challenging technical problems. Fascinated by practical uses of generative AI in software, he hosts the podcast "AI und jetzt," discussing AI's potential across industries. Robert bridges tech and business, advocating user-centric digitization. Off duty, he enjoys exploring the local food scene.

Alexander Kniesz

Alexander started in 2017 as a working student. Since 2021 he now works as a consultant at INNOQ. Since his master's degree in data science, his focus is on topics like data processing, machine learning, and AI. With those specializations, he is also interested in other software development topics to help bringing ML into production. Therefore he also assists in topics like Machine Learning Operations.

Hermann Schmidt

Hermann Schmidt is a Senior Consultant at INNOQ. After more than two decades as a developer and architect focusing on the “how” of software development, Hermann has shifted his focus to the “what” and “who”. As a facilitator, he is particularly interested in team dynamics, development processes, innovation processes, and methods. The problems hiding in the cloud between business and development have his attention. Recently, the appearance of large language models have ignited a spark that reminds him of his days as a 17-year-old in high school, sitting in awe and excitement in front of the only computer, writing his first programs.

Marco Steinke

Marco Steinke works as Consultant at INNOQ, where he specializes in software architecture. His professional interests also encompass artificial intelligence, with a particular emphasis on the design and incorporation of AI systems.

Retrieval-Augmented Generation (RAG) ermöglicht es, die Fähigkeiten von Large Language Models für unternehmensinternes Wissen zu erschließen. Das ermöglicht nicht nur Antwortsysteme wie Chatbots, sondern auch die Umsetzung von zuvor schwierigen oder unmöglichen Features auf der Grundlage von internen, unstrukturierten Daten.

In diesem Primer führen wir systematisch in die Konzepte und Architektur von RAG ein. Wir behandeln sowohl theoretische Grundlagen als auch praktische Implementierungsaspekte wie Chunking, Embedding und Vektordatenbanken. Außerdem teilen wir unsere Praxiserfahrung aus echten Projekten.

Für Softwarearchitekt:innen und -entwickler:innen, die einen kompakten, aber fundierten Einstieg ins Thema suchen und den Einsatz von RAG in der eigenen Organisation bewerten wollen.

innoq.com