Universität Leipzig
Faculty of Mathematics and Computer Science

RAGBench - A Benchmark Framework for Retrieval-Augmented Generation Systems

Master's Thesis

Jan Albrecht
Born Feb. 17, 1995 in Freital

Matriculation Number 3772326

1. Referee: Prof. Dr. Norbert Siegmund
2. Referee: Prof. Dr. Unknown Yet

Submission date: January 31, 2025

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Leipzig, January 31, 2025

.....
Jan Albrecht

Abstract

A short summary.

Contents

Chapter 1

Introduction

The year 2017 can be stated as the beginning of the interesting journey of artificial intelligent language models. With the publication of "Attention is all you need" from Vaswani et al. [2017, revised in 2023] the rapid development of language models and later large language models (LLMs) took of.

Today there is a variety of real world products that are using this technology, such as content generators like ChatGPT (OpenAI [2022]) or Claude (Anthropic [2023]), translators like DeepL (DeepL [2017]) and Coding Assistants like Github Copilot (Friedman [2022]). The list can be expanded with technologies like sentiment analysis, question answering systems, market research or education systems. Open-source models are available for each of these technologies, providing strong alternatives to proprietary services.

The remarkably capacity of large language models has led to wide acceptance in society. However LLMs have fundamental problems that can not be solved with more training or larger models. Training models frequently is expensive, so daily training isn't feasible. Therefore every new information such as elections, weather or sport results which occurred between last training and user prompt are unknown to the model. On top of that, models can only be trained on available data. Private informations of users that might be relevant for the prompt are not considered in the generation process. LLMs struggle also with long-tail information, which occures rarely in the training data (Kandpal et al. [2022]).

When information is missing or underrepresented, outputs will deviate from user inputs, repeat previous outputs or may be made up by the LLM. (Zhang et al. [2023]). This technology is already used many sectors with billions of customers such as marketing, retail and eCommerce, education, healthcare, finance and law and media. It is crucial to develop systems that are as correct as possible.

The solution to potential missing information in training is to provide all

necessary information to the LLM beforehand within the prompt, so that the generator just have to construct a coherent text for the user. This can be achieved with so call Retrieval-Augmented Generation Systems (RAGs), where the raw user prompt is used to retrieve relevant data from an database that gets summarised and inserted into the prompt for the generator. This method overcomes many of the challenges LLMs face. Data can be accessed from private and up-to-date sources. The frequency of information occurrences no longer matters as long as the database includes it for retrieval. For having recent information, it is not longer required to train the underlying model.

The price for RAGs is high. The system requires additional steps between the prompt request and output. Most of those steps can not be parallelized. That results in longer inference times and also leads to the more resource intensive system. Next to the increasing infrastructure costs, developing and maintaining a RAG is more time consuming than developing a LLM, because the LLM is a part of the larger system. RAGs are not by default significantly better systems than pure LLMs as Simon et al. [2024] showed. These complex systems are highly sensitive to small configuration changes.

Therefore there is an important question to be asked:

> "Is an advanced RAG system necessary for your use case, or is a standalone LLM sufficient?"
>
> "Is your RAG the best one for your specific use-case?"

The answers to this questions are hard to find, because you have to implement a RAG, to test it on your problem and your data. The scientific landscape of Retrieval-Augmented Generation Systems is a vast community with rapid development. Staying up-to-date with that research topic is time consuming for companies and research departments.

The implementation of the RAG is just one part of the decision process. It is difficult to evaluate LLMs and all systems that are based on that technology, because outputs are not deterministic and such models are like a black-box. It is not obvious why the model responds with a certain output. Another problem is that there is a whole set of potential correct answers, because text-based outputs can have many forms. Lets consider following example:

Question: *Is the evaluation process of LLMs an easy task?*
Answer 1: *The evaluation process of LLMs is not an easy task.*
Answer 2: *No, the evaluation process is a difficult task.*

Both answers are correct, but which metric must be used to measure this outcome?

The Evaluation of RAGs is even harder, because it has the same problems next to its own ones. RAGs are complex and consists of many parts. Each part can lead to errors and therefore all components of the systems need to be evaluated next to an end-to-end evaluation of the whole system as Salemi and Zamani [2024] and Yu et al. [2024] showed.

There are companies and research groups that successfully solved parts of this problem with developing tools, frameworks and libraries such as AutoRAG (2024), Llama-Index (2022), LangChain (2022), RaLLe (2023), FlashRAG (2024), RAGLAB (2024), Haystack (2019) and FastRAG (2023). While an in-depth analysis will follow later in this thesis, it can be stated, that all of those tools and frameworks are focussed on developing RAG-Variants, make them production-ready or evaluating them for performance, ignoring the fact, that RAGs must be measured for hardware metrics like latency, inference time and CPU usage to determine if the benefits in performance compensate the disadvantages. Additional to this, Simon et al. [2024] showed there is lack of external validity in the development of RAGs, because the iterative reconfiguration of those systems that leads to the best performance is an hyperparameter tuning process that overfits the model to the tested data and therefore requires an dataset split with an validation dataset and a holdout test dataset, which is only used to estimate the generalization error.

With that master thesis I will make two contributions to the scientific landscape of RAGs: (i) A novel benchmarking framework following the systematical blueprint showed by Simon et al. [2024] and evaluating hardware metrics next to SOTA performance evaluations, (ii) A RAG for the software engineering task configuration validation that is evaluated with the here presented benchmarking framework. For (i), the framework will extend Haystack 2019 next to the FastRAG library 2023. Haystack is an open-source framework for LLMs, RAGs and SOTA search systems. FastRAG build upon Haystack and adds special RAG architectures.

Here comes most important results
Here comes main conclusions
Here comes Chapter Outline ...

Chapter 2

Background

2.1 Large Language Models

NLP is a subfield of AI that focuses on interactions between computers and humans via natural language. LLMs are deep neural network models trained on large corpora of text data. They are predominantly based on the Transformer architecture Wolf et al. [2019]. Understanding Large Language Models and its hyperparameters like temperature or top-p is crucial for the development of RAG applications. Below, we explain the Transformer architecture and its main components.

2.1.1 Transformer Architecture

There are many well-known models based on the Transformer architecture, such as BERT, GPT-3.5, LLaMA, and others (Yin et al. [2024]). The Transformer architecture was introduced by Vaswani et al. in 2017. ~~Briefly summarized~~ it's based on the self-attention mechanism, which allows the model to weigh the importance of each word in a sentence. The architecture has been shown to outperform other architectures in various NLP tasks.

Figure ?? illustrates the Transformer architecture. The encoder processes the entire input text. The decoder is autoregressive and uses its own outputs to generate further text. In the following the

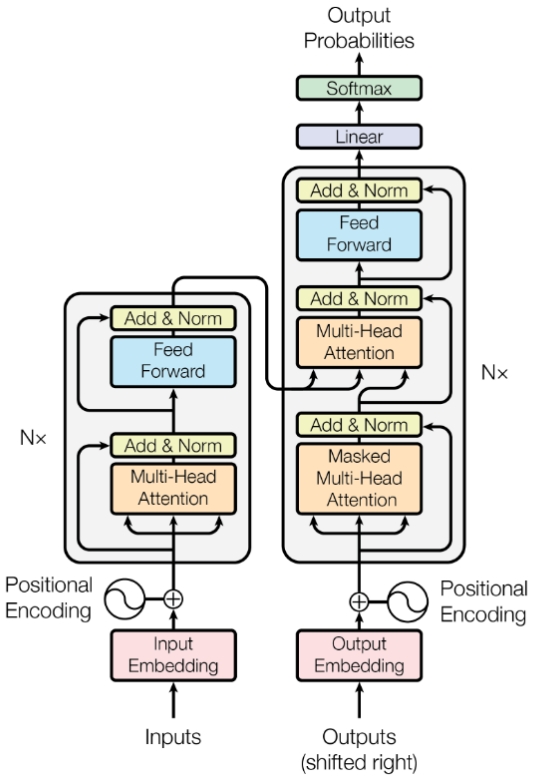

Figure 2.1: Transformer Architecture, Source: Vaswani et al. [2017, revised in 2023]

main components of the Transformer architecture are explained.

Encoder The encoder transforms the sequences in a list of tokens and passes them to the input embedding layer and to the the positional encoding layer. Subsequently, it passes through multiple layers of multi head self attention and feed forward neural network blocks. The outputs of the encoder is passed to the decoder, if the architecture is an encoder decoder architecture. Encoders can also stand alone for tasks like text and code classification. (Hou et al. [2023])

Decoder In the encoder decoder architecture the decoder begins with an initial output sequence or an empty sequence. The output sequence is passed to the output embedding layer and to the posi

tional encoding layer. After that it gets passed to several layers of masked and non masked multi head self attention, plus a feed forward neural network blocks. The outputs of the decoder is passed to the output layer, which is a linear layer followed by a softmax function. The output layer predicts the next token in the output sequence. The output sequence is passed to the decoder again, so that the model can predict the next token based on the previous tokens. This process is called autoregressive generation. The decoder can also stand alone for autoregressive tasks like code completion, text to code generation, debugging, etc. as shown in Hou et al. [2023].

Input Embedding Depending on the model, the input and output sequences are tokenized into subwords, words or characters. The input embedding layer converts tokenized sequences into dense vector representations. The input embedding layer is trained along with the rest of the model. The output of the input embedding layer is passed to the positional encoding layer. The positional encoding layer adds information about the position of each token in the input sequence, so that the model can distinguish between words with the same token but different positions. Positional encodings are combined with input embeddings before being fed into the encoder. The output embedding is equivalent to the input embedding, but it is used to convert the output tokens into a dense vector representation. The output embedding is used in the decoder part of the Transformer architecture.

Multi-Head Self-Attention Block Both in the encoder and in the decoder, a block consists of several multi head self attention layers followed by a feed forward neural network layer. The multi head self attention layer is the main component of the Transformer architecture. It allows the model to weigh the importance of each token in the input sequence for each other token. The multi head

self attention layer consists of multiple heads. Each head consists of a Query, Key and Value matrix, which are used to calculate the attention scores between the input tokens. The attention scores are used to weigh the importance of each token in the input sequence for each other token. The attention mechanism is mathematically defined as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Query (Q), Key (K) and Value (V) are matrices of the input tokens. This mechanism is based on common retrieval. Query can be seen as the search query, Key as the potential candidates and Value as the retrieved information.

All matrices are learned during training. Query and Key are used to calculate the attention scores. The division by $\sqrt{d_k}$ is used to stabilize the gradients during training. The softmax function is used to normalize the attention scores. For the decoder part with the masked multi head self attention, the attention scores are masked with $-\infty$ for every token after the current calculated one right before the softmax, so that the model can only attend to previous tokens in the output sequence and the next predicted token is based only on the previous tokens. AAll attention heads are concatenated, passed through a linear layer, which is finally followed by a layer normalization to stabilize the gradients during training.

Putting it all together After the input sequence is passed through the encoder, the output sequence is passed through the decoder. The decoder uses the encoder output as additional input and autoregressively predicts the next token in the output sequence. The output sequence is passed to the output layer, which is a linear layer followed by a softmax function. The output layer predicts the next token in the output sequence. The output sequence is passed to the decoder again, so that the model can predict the next token

based on the previous tokens. This process is called autoregressive generation.

There are several ways to predict the next token within the output layer. The most straightforward way is to predict the token with the highest probability. This is called greedy decoding. Another way is to sample the next token from the probability distribution. This is called sampling. There are also other ways to predict the next token, such as beam search, nucleus sampling, top k sampling, top p sampling, etc. These methods are used to improve the performance of the model. In the following the most important tuning parameters for LLMs are explained.

2.1.2 Tuning Parameters

There are several tuning parameters that can be used to improve the performance of LLMs. Some of the most important tuning parameters are:

Temperature The temperature parameter is used to control the randomness of the generated text. A high temperature value leads to more randomness, while a low temperature value leads to less randomness. The temperature parameter T modifies the softmax function as follows:

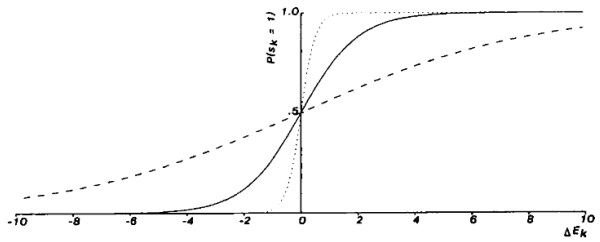

Figure 2.2: The softmax function with different temperature values, T=1.0 (solid), T=4.0 (dashed), T=0.25 (dotted), Source: ACKLEY et al. [1985]

$$softmax : o(z)_i = \frac{e^{\beta z_i}}{\sum_{j=1}^{K} e^{\beta z_i}}$$

In figure 2.2 the softmax function is shown. A low temperature value leads to a peaky distribution, while a high temperature value leads to a more uniform distribution and therefore to more randomness.

Top-K Sampling Top-K sampling by Fan et al. [2018] samples from the top K tokens with the highest probability before applying the softmax function. Therefore the higher the K value, the more tokens are considered for sampling. Higher values increase output randomness. With Top-K equal to 1, the model behaves like greedy decoding.

Top-P Sampling The Top-P sampling by Holtzman et al. [2019] samples from the smallest set of tokens whose cumulative probability exceeds the probability P. Therefore the higher the P value, the more tokens are considered for sampling. The outcomes gets more random. It is a generalization of Top-K sampling, where the number of tokens to sample from is not fixed.

2.1.3 Prompting Techniques

The Transformer architecture is not deterministic as shown in previous sections. Additionally LLMs have a high variance for its outputs too. The input prompt critically influences output quality. Therefore there are plenty of prompting techniques, to stabilize desired outputs. Schulhoff et al. created a taxonomy and recognized 58 LLM prompting techniques. I discuss four representative prompting techniques: Few-Shot, Zero-Shot, Chain-of-Thought (CoT), and Chain-of-Verification (CoVe). The list is not complete, but those are very commonly used and easy to implement.

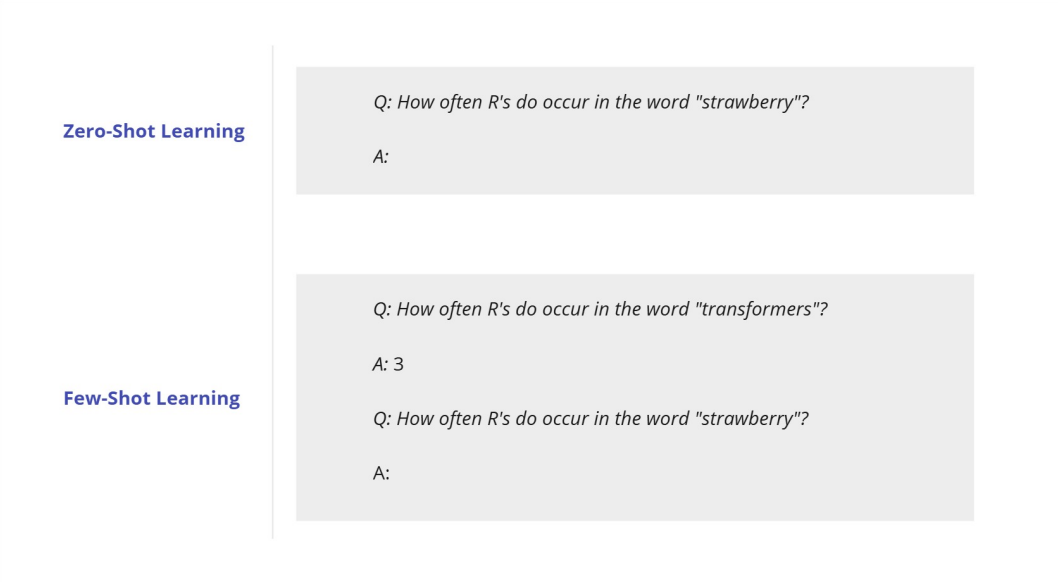

Figure 2.3: An example with a difficult query, that many LLM struggle, first without example (Zero-Shot) and below with one example (Few-Shot).

Few-Shot vs. Zero-Shot Few-Short prompting is a form of In-Context Learning (ICL), in which the prompt has examples of similar tasks to show the model how a task should be done. Brown et al. showed that large models can leverage examples for various tasks. Zero-Shot prompting refers then to prompting without showing examples before the task. Even though Few-Shot prompting might increase performance of LLMs, it comes with increased costs, as there are more tokens to process. An example of those prompting methods can be seen in figure 2.3.

Chain-of-Thought This method introduced by Wei et al. leverages Few-Shot learning through simulating a written thought process in prior examples. The LLM is therefore forced to take over this behaviour, which leads to more accurate outputs in reasoning and math problems. Schulhoff et al. refers it to the class of Thought Generation techniques. Figure 2.4 shows an example prompt.

Chain-of-Verification

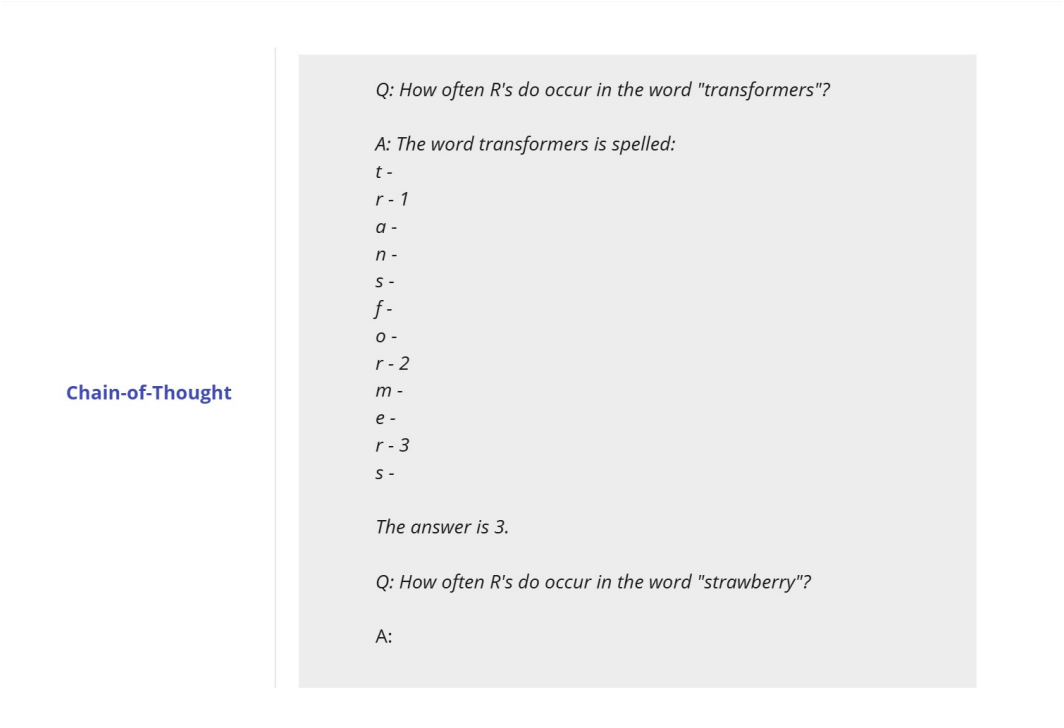

Figure 2.4: An Chain-of-Thought example with a difficult query, that many LLM struggle without prompting techniques

2.2 Text Retrieval

There are several types of text retrieval for RAG systems. Zhao et al. [2024] classifies four retrieval techniques based on how retrieved information is passed to the generation process. This thesis focuses exclusively on query-based retrieval techniques due to their prevalence and broad adoption. The other three types are latent-representative-based retrieval, logit-based and speculative retrieval.

Another categorization of text retrieval can be done by the type of retrieval. There are three main types of retrieval: sparse retrieval, dense retrieval and hybrid retrieval. Sparse retrieval is based on traditional information retrieval techniques like TF-IDF or BM25. Dense retrieval is based on transformers models and hybrid retrieval combines both sparse and dense retrieval techniques. Following I will describe those techniques in more detail.

2.2.1 Sparse Retrieval

TF-IDF and BM25 are widely implemented in libraries like LangChain (Chase [2022]) and LlamaIndex (Liu [2022]), representing standard sparse retrieval approaches.

TF-IDF TF as shown in Manning et al. [2009] refers to the term frequency of a term t in a document d. The inverse document frequency (IDF) is calculated as the logarithm of the total number of documents N divided by the number of documents containing the term t. Therefore the IDF factor is low for terms that occur in all documents and high for distinguishing terms that occur in frequently in a low number of documents. The position of the words is ignored.

$$\mathit{TF\text{-}IDF}(t, d, D) = \mathit{TF}(t, d) \cdot \mathit{IDF}(t, D) = \mathit{TF}(t, d) \cdot \log\left(\frac{N}{\mathit{DF}(t, D)}\right)$$

BM25 Manning et al. [2009] presents multiple variants of BM25. A simplified version is defined as follows:

$$BM25(t, d, D) = IDF(t, D) \cdot \frac{(k_1 + 1) \cdot TF_{t,d}}{k_1((1 - b) + b \cdot \frac{L_d}{L_{ave}}) + TF_{t,d}}$$

The BM25 score is an advanced version of the TF-IDF score with two free parameters k_1 and b. The parameter k_1 is a scaling factor to determine how relevant term frequency is. The parameter b is for document length scaling, reducing scores of long documents.

2.2.2 Dense Retrieval

Dense Passage Retrieval as shown in Karpukhin et al. [2020] utilizes Bidirectional Encoder Representation from Transformers (BERT, Devlin et al. [2019]). BERT corresponds to the encoder component

of a sequence-to-sequence Transformer architecture.Therefore it can be used to encode the text passages used as contexts at the classification token [CLS]. Text passages are mapped into a d-dimensional vector space under the assumption that semantically similar passages exhibit proximity. The query is also encoded into this space and the similarity between the mapped query vector and each passage vector is calculated. The similarity is calculated by the dot product or other similarity functions. The top-k similar passages are then selected and used for the generation.

Contemporary specialized embedding models are benchmarked through frameworks like HuggingFace's MTEB (Muennighoff et al. [2022]). While MTEB rankings suggest optimal embedding models, no universal solution exists. Embedding models are language- and domain-sensitive as Gao et al. [2023] points out.

2.2.3 Hybrid Retrieval

Both sparse retrieval techniques TF-IDF and BM25 are good for keyword specific searches, but perform poorly on a semantic comparison of documents and query. Encoders like BERT, trained for language understanding, excel at semantic comparisons. Often it is not trivial to dertermine if a text retrieval needs a keyword-relevant sparse retrieval or dense retrieval considering semantics.

Hybrid models compute both dense and sparse retrieval scores, combining them through weighted summation: $\alpha \cdot dense + (1 - \alpha) \cdot sparse$. An factor $\alpha = 1$ results in using only dense retrieval. There is no universal performant value for α. This makes α a tunable hyperparameter requiring optimization.

2.3 Vector Databases

Storing high-dimensional vector data cannot be done efficiently with traditional databases. Vector databases (VDBs) specialize in

efficiently searching for semantically similar entities within high-dimensional data. Especially for dense retrieval, similarity search is a highly complex topic. Given a query vector v_1, finding the nearest neighbor v_n would require comparing v_1 with every vector in the dataset. Jing et al. [2024] notes this brute-force approach yields $O(dN+N\ log\ k)$ complexity, making it impractical for large-scale applications. Efficient alternatives to brute-force search constitute the key differentiating capability of modern vector database systems. To date, no comprehensive comparison of search latency across VDB providers exists to our knowledge. Pan et al. [2024] compared providers regarding features they offer and pointed out that cross-disciplinary comparison of vector search algorithm and systems are scarce.

While numerous optimization techniques exist, this thesis focuses on two representative methods: We detail Product Quantization (PQ) and the Hierarchical Navigable Small World (HNSW) algorithm, a state of the art approximate nearest neighbor (ANN) method—due to their widespread adoption and effectiveness. I refer to Jing et al. [2024], and Kukreja et al. [2023], which explained many other techniques in great detail.

Hierarchical Navigable Small World (HNSW) Approximate Nearest Neighbor is a term referencing all Nearest Neighbor algorithms, that do not search for the most similar or closest point, but rather for the match that is close enough. One version of that is the HNSW algorithm, which was developed by Malkov et al. [2014] and uses a skip pattern invented by Pugh [1990]. The algorithm is visualized in figure 2.5. HNSW constructs a hierarchical graph where nodes connect via randomly sampled edges across multiple layers. Upper layers contain long range connections, while lower layers progressively feature shorter, localized edges. The algorithm starts comparing on the top level layer and if it reaches local minimum, it continues on the next layer, till it finds its local minimum

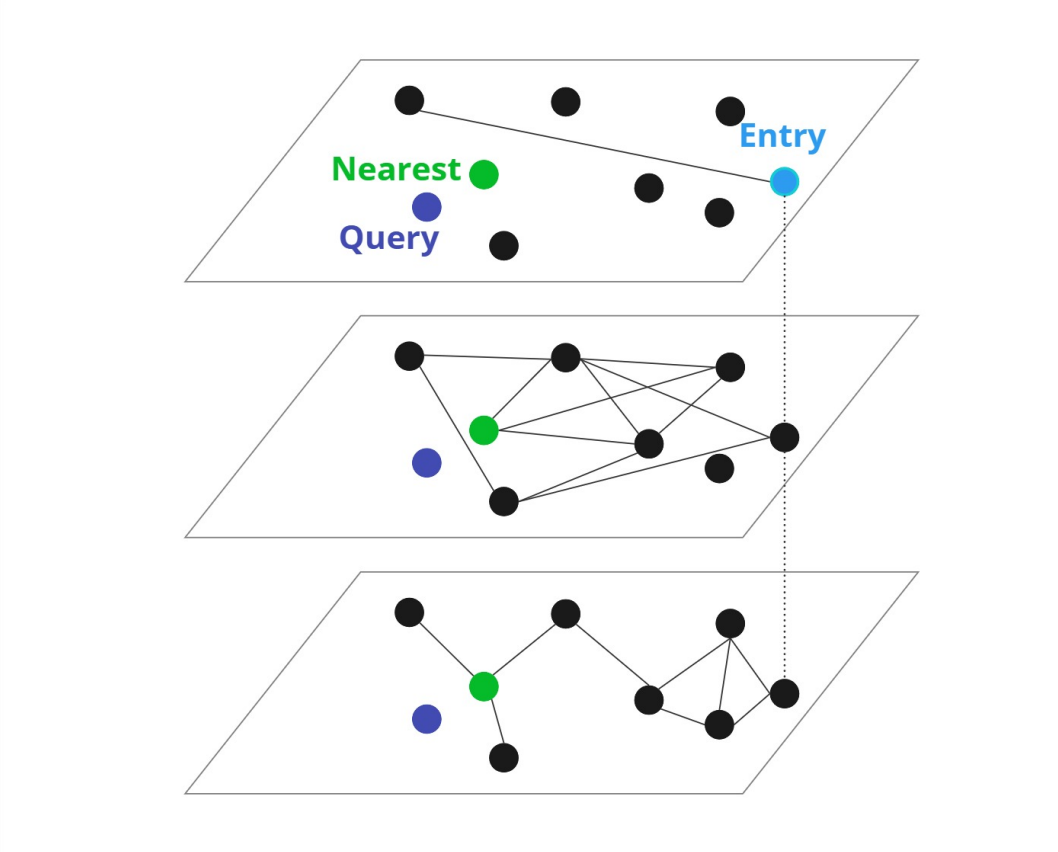

Figure 2.5: HNSW algorithm visualized, highly inspired by Pinecone [22.01.2025]

~~on the bottom layer. While HNSW sacrifices guaranteed optimality, it achieves a favorable trade off between recall and execution speed, as demonstrated by Erik Bernhardsson [22.01.2025].~~

2.4 Retrieval-Augmented Generation Systems

Large language models (LLMs) exhibit fundamental limitations that cannot be fully resolved through training or prompt engineering. Gao et al. [2023] stated that LLMs have significant limitations in knowledge-intesive or domain-specific tasks. Insufficient domain-specific training data often results in erroneous outputs. For large language models this is called *hallucinations* as defined in Huang et al. [2024]. In retrieval-augmented generation (RAG) systems,

hallucinations refer to outputs unsupported by provided context. Rashkin et al. [2021] defined hallucinations as any information that is neither inferable from nor stated by an external document. In this thesis I will use the definition for RAG systems.

Factual wrong output from an LLM can have several reasons. The required information to answer this question can be private, stored on a database that is not accessible during training. Additionally, the high cost of training/fine-tuning limits update frequency. This creates temporal gaps where post-training information remains unavailable. All those problems can be solved if the LLM is not used for the answer itself, but rather used for building a coherent text passage given a question and its answer. Then a database would store all relevant information and would be updated as frequent as required. During inference, the system retrieves relevant document chunks from the database to inform answer generation.

This concept is very successful as Shuster et al. [2021] showed. This approach reduces factual errors while enhancing open-domain conversational performance. Yu et al. [2024] concluded that this improves the reliability and richnes of the produced content. Chen et al. [2024] identifies external knowledge integration as crucial for improving LLM accuracy and reliability.

There is a high variety in architectures and approaches for retrieval-augmented generation models. In this section, I will give an overview of the most common approaches and architectures and start with the most basic ones. The section will follow the definition of RAGs in Gao et al. [2023] with naive RAGs, advanced RAGs, a modular architecture, and special cases.

2.4.1 Naive RAGs

Naive RAGs represent the foundational implementation of retrieval-augmented generation. These systems concatenate retrieved con-

text with user queries to guide LLM responses. Then the LLM is only used for generating a coherent answer. Therefore it is required to ingest relevant data for the given use case beforehand. The system then retrieves relevant chunks or documents of this data at inference time.

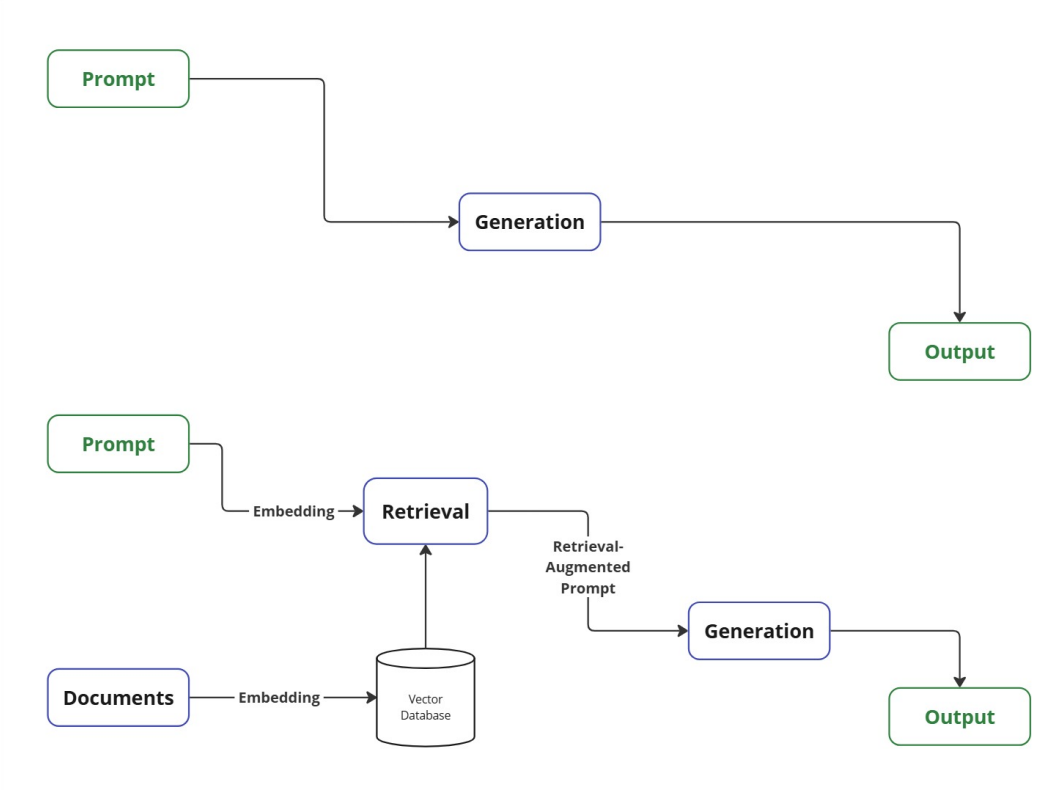

Figure 2.6: Comparison of the process, above a standalone LLM that has a prompt and generate a response, below a minimalistic RAG system that performs on a given set of documents an embedding or indexing and searchs for the most similar documents for a given prompt at inference time. Both prompt and documents are used for the generation process.

This procedure can be seen in figure 2.6. This generation phase is frequently termed the *read* operation in literature. In Gao et al. [2023] they define the naive RAG system as *Retrieve-Read*. The retrieval can be done with sparse retrieval (TF-IDF, BM25), dense retrieval (DPR) or a hybrid version as showed in section 2.2.

Before the system can be used, it is necessary to ingest data.

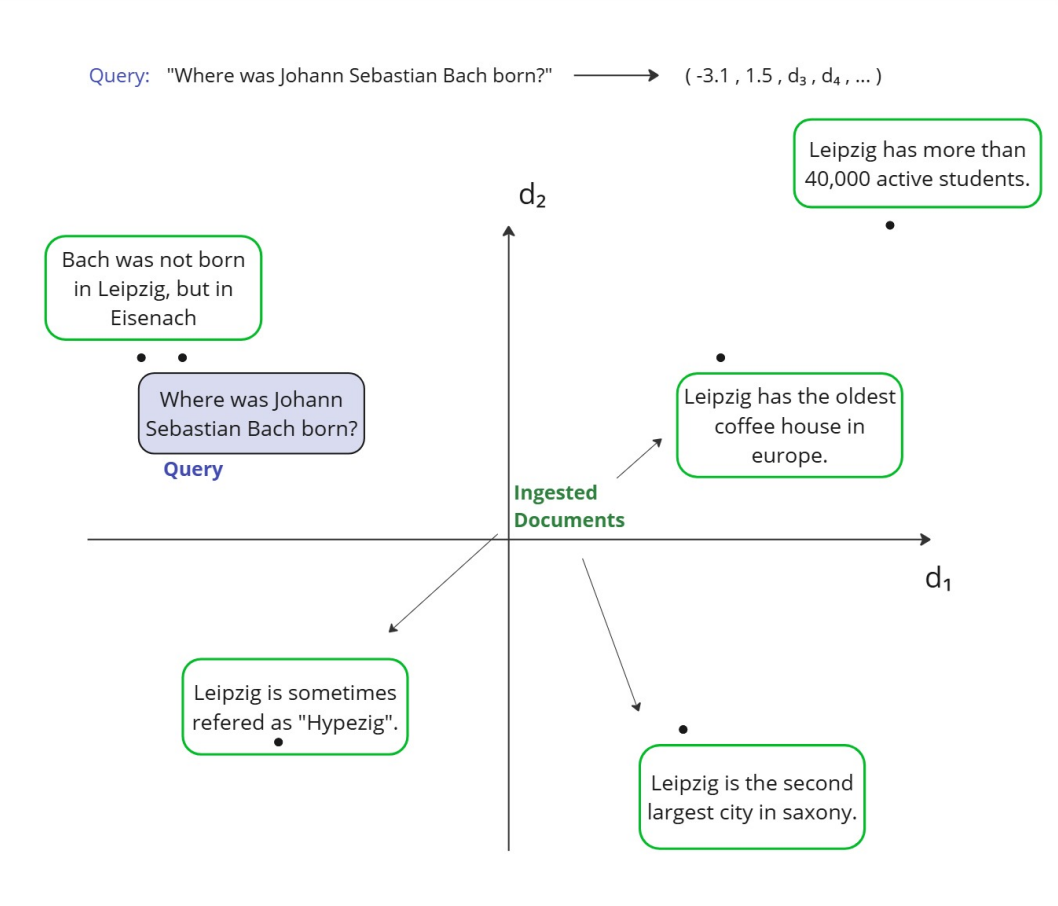

Figure 2.7: A vector space including mapped facts (documents) and a user query. The query and its correct answer have similar vectors

The ingestion process includes preprocessing and selection of data, that users are likely to need for their questions or use-cases. Therefore it is relevant to know the user base. It is not feasible nor efficient to ingest all availlable data. The preprocessing pipeline converts raw data into document chunks using methods detailed in Section ??. As described in 2.2.2, the chunks are then embedded into a d-dimensional real-valued vector space. There is a simplistic minimal example of a vector space in figure 2.7. The documents gets mapped in different loactions within the space. This assumes semantic alignment between queries and relevant documents in the embedding space. In reality there will be more documents close to

each other, documents and queries are longer or more complex and each query will return the "Top-K" chunks or documents, which are not always as relevant as in this example. Top-K can be seen as hyperparamter for the system. The vectors are then stored in vector databases as described in section 2.3.

Gao et al. [2023] listed several drawbacks for naive RAGs. The basic retrieval suffers from unsufficient recall and precision scores leading in irrelevant documents, missing context and bias. The integration of the provided context is a challenging process. The generator often overrelies on the augmented information, by just repeating the retrieved content and missing insightful conclusions. Therefore this simplistic form of RAG needs advanced techniques to overcome those issues.

2.4.2 Advanced RAGs

There is no strict definition of advanced versions of retrieval-augmented generation systems. The term describes a loose bundle of techniques to improve the quality of such systems. Following is a list, which items I will explain in greater details afterwards.

Chunking Chunking's importance becomes clear through a practical example: Let us use a sentence from Wikipedia for Leipzig from 2025. In the section "Music" there is the following sentence.

> "Johann Sebastian Bach spent the longest phase of his career in Leipzig, from 1723 until his death in 1750, conducting the Thomanerchor (St. Thomas Church Choir), at the St. Thomas Church, the St. Nicholas Church and the Paulinerkirche, the university church of Leipzig (destroyed in 1968)."

If there are thousands of wikipedia pages or websites to process, then this can not be splitted manually. What is the document or a

good chunk in this context. If the query asks if Johann Sebastian Bach lived in Leipzig, then embedding the whole sentence would not guarantee a high similarity vector score in the retrieval process. This differs from domain to domain. While chunking facts from sentences might be a valid strategy for wikipedia, processing internal contracts in a large company would need less granular chunking.

Therefore there are several chunking techniques to consider for tuning the RAG-system. Fixed-size chunking divides text into uniform segments (e.g., 400-character blocks), while sliding window variants add overlapping regions. The sliding window approach adds a overlap of few characters. Semantic chunking splits with prior defined characters such as "$\backslash n$, $\backslash\backslash n$ *or* $
$". There are also special use-case techniques such as Markdown, JSON, HTML and programming code chunkers. Almost all presented chunking techniques are offered in typical libraries e. g. Llama-Index (Liu [2022]) or Langchain (Chase [2022]).

Rewrite

Rerank

IRA Methods

2.4.3 Drawbacks of RAGs

-

Chapter 3

Relative Works

RAG vs LC LLM?

3.1 RAG Evaluation

Chapter 4

RAG-Evaluation Framework

4.1 Roter Faden ...

RAG seems to be a free lunch (see paper with RAG vs Long Context LLMs), but isn't.

- System more complexity increased
- Inference Time increased
- Only useful in some scenarios
- Hard to evaluate because of its many parts
- ...

All items are interesting in this thesis. The background ellaborated on the variety of advanced RAG techniques and here the conclusion would be that an evaluation framework should include hardware metrics next to performance metrics for measuring the overhead of the RAG compared to a standalone LLM. The usefulness is measured by the three stage evaluation through LLM, Naive RAG, Advanced configured RAG. Here it would be important to point out the failure points within a RAG system, as it was showed in lecture of Prof. Siegmund too.

Bibliography

D. ACKLEY, G. HINTON, and T. SEJNOWSKI. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985. ISSN 03640213. doi: 10.1016/S0364-0213(85)80012-4. 2.2

Anthropic. Introducing claude, Mar 2023. URL `https://www.anthropic.com/news/introducing-claude`. 1

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. URL `http://arxiv.org/pdf/2005.14165`. 2.1.3

Harrison Chase. LangChain. `https://github.com/langchain-ai/langchain`, October 2022. URL `https://github.com/langchain-ai/langchain`. 1, 2.2.1, 2.4.2

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17754–17762, 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29728. 2.4

DeepL. Deepl übersetzer: Der präziseste übersetzer der welt, 2017. URL `https://www.deepl.com/de/`. 1

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL `https://arxiv.org/abs/1810.04805`. 2.2.2

Erik Bernhardsson. erikbern/ann-benchmarks: Benchmarks of approximate nearest neighbor libraries in python, 22.01.2025. URL `https://github.com/erikbern/ann-benchmarks?tab=readme-ov-file`. 2.3

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation, 2018. URL `http://arxiv.org/pdf/1805.04833`. 2.1.2

Nat Friedman. Introducing github copilot: Your ai pair programmer, Feb 2022. URL `https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/`. 1

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2023. URL `http://arxiv.org/pdf/2312.10997`. 2.2.2, 2.4, 2.4.1, 2.4.1

Ari Holtzman, Jan Buys, Du Li, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2019. URL `http://arxiv.org/pdf/1904.09751`. 2.1.2

Yasuto Hoshi, Daisuke Miyashita, Youyang Ng, Kento Tatsuno, Yasuhiro Morioka, Osamu Torii, and Jun Deguchi. Ralle: A framework for developing and evaluating retrieval-augmented large language models. `https://arxiv.org/abs/2308.10633`, 2023. URL `https://arxiv.org/abs/2308.10633`. 1

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, Lo David, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review, 2023. URL `http://arxiv.org/pdf/2308.10620v6`. 2.1.1, 2.1.1

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, 2024. URL `http://arxiv.org/pdf/2311.05232`. 2.4

Peter Izsak, Moshe Berchansky, Daniel Fleischer, and Ronen Laperdon. fastRAG: Efficient Retrieval Augmentation and Generation Framework. `https://github.com/IntelLabs/fastrag`, February 2023. URL `https://github.com/IntelLabs/fastrag`. 1

Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. *CoRR*, abs/2405.13576, 2024. URL `https://arxiv.org/abs/2405.13576`. 1

Zhi Jing, Yongye Su, and Yikun Han. *When Large Language Models Meet Vector Databases: A Survey*. 2024. 2.3

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. Large language models struggle to learn long-tail knowledge, 2022. URL `http://arxiv.org/pdf/2211.08411`. 1

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020. URL `https://arxiv.org/abs/2004.04906`. 2.2.2

Jeffrey Kim and Bobb Kim. AutoRAG. `https://github.com/Marker-Inc-Korea/AutoRAG`, February 2024. URL `https://github.com/Marker-Inc-Korea/AutoRAG`. 1

Sanjay Kukreja, Tarun Kumar, Vishal Bharate, Amit Purohit, Abhijit Dasgupta, and Debashis Guha. Vector databases and vector embeddings-review. In *2023 International Workshop on Artificial Intelligence and Image Processing (IWAIIP)*, pages 231–236. IEEE, 2023. doi: 10.1109/iwaiip58158.2023.10462847. 2.3

Jerry Liu. LlamaIndex. `https://github.com/jerryjliu/llama_index`, 11 2022. URL `https://github.com/jerryjliu/llama_index`. 1, 2.2.1, 2.4.2

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014. ISSN 03064379. doi: 10.1016/j.is.2013.10.006. 2.3

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge Univ. Press, Cambridge, reprinted. edition, 2009. ISBN 9780521865715. doi: 10.1017/CBO9780511809071. 2.2.1, 2.2.1

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022. doi: 10.48550/ARXIV.2210.07316. URL `https://arxiv.org/abs/2210.07316`. 2.2.2

OpenAI, Nov 2022. URL `https://openai.com/index/chatgpt`. 1

James Jie Pan, Jianguo Wang, and Guoliang Li. Survey of vector database management systems. *The VLDB Journal*, 33(5):1591–1615, 2024. ISSN 0949-877X. doi: 10.1007/s00778-024-00864-x. URL `https://link.springer.com/article/10.1007/s00778-024-00864-x`. 2.3

Malte Pietsch, Timo Möller, Bogdan Kostic, Julian Risch, Massimiliano Pippi, Mayank Jobanputra, Sara Zanzottera, Silvano Cerza, Vladimir Blagojevic, Thomas Stadelmann, Tanay

Soni, and Sebastian Lee. Haystack: the end-to-end NLP framework for pragmatic builders. `https://github.com/deepset-ai/haystack`, November 2019. URL `https://github.com/deepset-ai/haystack`. 1

Pinecone. Hierarchical navigable small worlds (hnsw) | pinecone, 22.01.2025. URL `https://www.pinecone.io/learn/series/faiss/hnsw/`. 2.5

William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990. ISSN 0001-0782. doi: 10.1145/78973.78977. 2.3

Hannah Rashkin, David Reitter, Gaurav Singh Tomar, and Dipanjan Das. Increasing faithfulness in knowledge-grounded dialogue with controllable features. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 704–718, Stroudsburg, PA, USA, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.58. 2.4

Alireza Salemi and Hamed Zamani. Evaluating retrieval quality in retrieval-augmented generation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2395–2400, New York, NY, USA, 2024. ACM. doi: 10.1145/3626772.3657957. 1

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat,

Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. The prompt report: A systematic survey of prompting techniques. URL `http://arxiv.org/pdf/2406.06608`. 2.1.3, 2.1.3

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3784–3803, Stroudsburg, PA, USA, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.320. 2.4

Sebastian Simon, Alina Mailach, Johannes Dorn, and Norbert Siegmund. A methodology for evaluating rag systems: A case study on configuration dependency validation, 2024. URL `http://arxiv.org/pdf/2410.08801v1`. 1

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017, revised in 2023. URL `https://arxiv.org/abs/1706.03762`. 1, 2.1

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Le Quoc, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. URL `http://arxiv.org/pdf/2201.11903`. 2.1.3

Wikipedia. Leipzig, 2025. URL `https://en.wikipedia.org/w/index.php?title=Leipzig&oldid=1269825057`. 2.4.2

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin

Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2019. URL `http://arxiv.org/pdf/1910.03771`. 2.1

Junqi Yin, Avishek Bose, Guojing Cong, Isaac Lyngaas, and Quentin Anthony. Comparative study of large language model architectures on frontier. In Kyle Chard and Zhuozhao Li, editors, *2024 IEEE International Parallel and Distributed Processing Symposium*, pages 556–569, Piscataway, NJ, 2024. IEEE. ISBN 979-8-3503-8711-7. doi: 10.1109/IPDPS57955.2024.00056. 2.1.1

Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. Evaluation of retrieval-augmented generation: A survey, 2024. 1, 2.4

Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, et al. RAGLAB: A modular and research-oriented unified framework for retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, December 2024. 1

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. Siren's song in the ai ocean: A survey on hallucination in large language models, 2023. URL `http://arxiv.org/pdf/2309.01219`. 1

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey, 2024. URL `http://arxiv.org/pdf/2402.19473`. 2.2