Towards Personal Assistants for Energy Processes Based on Locally Deployed LLMs

Maximilian Orlowski[a], Emilia Knauff and Florian Marquardt
Research Group for Cloud Computing, University of Applied Science Brandenburg,
Magdeburger Straße 51, Brandenburg an der Havel, Germany
ﬂ

Keywords: GenAI, LLM, RAG, Embedding, Network-Operator, Business-Process, Assistant, Onboarding Coaching.

Abstract: This paper presents a coaching assistant for network operator processes based on a Retrieval-Augmented Generation (RAG) system leveraging open-source Large Language Models (LLMs) as well as Embedding Models. The system addresses challenges in employee onboarding and training, particularly in the context of increased customer contact due to more complex and extensive processes. Our approach incorporates domain-specific knowledge bases to generate precise, context-aware recommendations while mitigating LLM hallucination. We introduce our systems architecture to run all components on-premise in an our own datacenter, ensuring data security and process knowledge control. We also describe requirements for underlying knowledge documents and their impact on assistant answer quality. Our system aims to improve onboarding accuracy and speed while reducing senior employee workload.

The results of our study show that realizing a coaching assistant for German network operators is reasonable, when addressing performance, correctness, integration and locality. However current results regarding accuracy do not yet meet the requirements for productive use.

1 INTRODUCTION

The rapid advancement of generative artificial intelligence (GenAI) technologies, such as LLMs have gained a lot of attention lately. This increased focus is justified, as these innovative systems have had a profound impact across nearly all industries, including the realm of customer service operations. Notably enterprises providing customer services as part of their own value chain or as outsourcing providers are often characterized by relatively high employee turnover rates, leading to significant challenges in maintaining consistent quality and productivity throughout their operations. Therefore, fast and precise on-boarding along with effective coaching of new employees is crucial to deliver high quality customer service operation. Focusing specifically on customer service operation within German network operators the massive increase in private PV feed-in led to massively increased customer contact volumes in recent years. Network operators have traditionally relied on manual training processes, such as classroom sessions, new employees shadowing experienced colleagues, and providing extensive documentation. While foundational, these approaches are often time-consuming, resource-intensive, and struggle to keep pace with fast changing operational landscapes and technological advancements. Digital learning management systems (LMS) and knowledge bases have been introduced in recent years to streamline the training process (Turnbull et al., 2019).

However, these systems often lack the flexibility to quickly provide context-specific information, their effectiveness is heavily dependent on regular manual updates and maintenance and there is a need to regularly schedule and command instruct training sessions. Some organizations have started experimenting with AI-powered chat bots for employee support and customer service (Rakovac Bekeš and Galzina, 2023). While the knowledge bases of these initial AI implementations are vast, they often lack specificity. This results in a tendency to provide generic responses that may not address the specific needs of network operators. Moreover, in case LLMs are used, these systems tend to hallucinate, especially when the requested knowledge is sparsely represented in their training data. This poses significant risks in contexts where accuracy is crucial.

Many current AI solutions also face challenges in terms of integration with existing systems and work-

[a] https://orcid.org/0009-0001-2223-9376

Orlowski, M., Knauff, E. and Marquardt, F.
Towards Personal Assistants for Energy Processes Based on Locally Deployed LLMs.
DOI: 10.5220/0013175600003890

In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 3*, pages 695-706
ISBN: 978-989-758-737-5; ISSN: 2184-433X

flows, limiting their practical utility in day-to-day operations. Furthermore, cloud-based AI solutions raise concerns about data security and privacy, particularly when dealing with sensitive customer and process information (Pakmehr et al., 2023). These limitations highlight the need for a more advanced, secure, and context-aware solution that can provide accurate and timely support while being safe and secure in terms of privacy and seamlessly integrates with existing operational processes.

Our aim is to provide a ChatGPT-like user experience for new and existing employees, assisting them in navigating new knowledge during their work. Therefore, we need to ensure suitable performance, correct and helpful answers, seamless UI integration, safe and secure management of both special process knowledge as well as input user data.

We argue that four dimensions **performance**, **correctness**, **integration** and **locality** are crucial when building an AI-based assistants in compliance sensitive markets like the German energy sector.

To address these challenges, we propose an on-premise deployed Retrieval-Augmented Generation (RAG) system based on open-source Large Language Models, designed as a coaching assistant for network operator processes. We contribute a detailed description of the needed infrastructure, systems and data and evaluation measure outcomes. Implementing the introduced approach can substantially enhance accuracy and speed of onboarding while simultaneously reducing workload of seasoned employees.

2 UNDERLYING TECHNOLOGIES

In the context of this paper two types of models are relevant: Large Language Models as well as Embedding Models.

An LLM is a type of neural network that uses the transformer architecture (decoder-only transformer, to be precise) with self-attention heads(Yenduri et al., 2023). This is also reflected in the GPT abbreviation which stands for *Generative Pretrained **Transformer***. It takes a natural language text input of arbitrary length and, in turn, generates natural language text output. Because of its stochastical nature, the output with the highest probability for the given input is returned. The models are trained on large corpora of texts, such as the entirety of digitalized books of the world, the complete content of Wikipedia or huge amounts of scraped web content.

Since the first release of pretrained language models like GPT-1 by OpenAI(Radford et al., 2018), numerous models have been developed, ranging from proprietary to open-source implementations. These models often differ in the datasets (corpora) they were trained on, their size (which is determined by the number of parameters in the networks), the network architecture they use (encoder-decoder, decoder-only)(Fu et al., 2023) and the licenses they are released under.

All of these models suffer from certain problems, the most prominent of which is called hallucination(Brown et al., 2020), (Joshi et al., 2021). Hallucination describes the problem, where a model produces an output that is factual or logical incorrect for the given input, even though it might still be the most probable output the model could generate for that specific input, given its training data. To mitigate this problem, different techniques have been developed over time.

Firstly, looking at an LLM as a black box there are two parts that can be adjusted to cope with hallucinations. The model itself as well as the prompt. Tonmoy et al. describe these possibilities as *Developing Models* and *Prompt Engineering* respectively (Tonmoy et al., 2024). Developing the model could mean one of two things. One option is to train an entire model from scratch on a dataset derived from the respective domain. However this kind of pretraining comes with the burden of requiring massive computational ressources which is rarely feasible outside big tech companies. The second option is model fine-tuning, which also relies on a domain specific data set. However, unlike pretraining, fine-tuning only adjusts a significant smaller fraction of the models parameters, usually the last layer of the neural network. This reduces computation costs significantly and makes it possible to fine-tune models even on consumer hardware (like a NVIDIA GTX 4090) for small models (e.g., up to 7 billion parameters). Also the amount of data needed for fine-tuning is significant smaller in comparison with a pretraining.

Prompt Engineering, the second approach to mitigate hallucinations, aims to adjust the prompt (text input) in a way that reduces the likelihood of hallucinations. One commonly used tactic is to add text to the input, that tells the LLM not to produce an answer if that answer is not based on actual data and instead tell the user that a factual answer cannot be generated from the (pretrained) data.

In RAG Systems, in addition to this method, the LLM is provided with a prompt that already contains all the facts that are required to answer the question correctly. From this fact-enriched prompt the LLM only needs to generate a well formulated answer containing the provided facts.

In contrast to LLMs Embedding Models are models that are used to produce numeric vector representations of text (although other types of information, such as images, can be embedded too, we focus on textual representation). Although each LLM contains an embedding layer, standalone embedding models are usually used in RAG systems for the first step, known as retrieval. Initially all facts or chunks of knowledge are handed over to the embedding model which generates embedding vectors (in short *embeddings*) of them.

Retrieval Augmentation Generation (RAG) is a technique to enhance the performance of large language models (LLMs)(Lewis et al., 2021),(Guu et al., 2020). It allows the incorporation of domain-specific knowledge bases to generate more accurate and context-aware responses by an LLM. This approach is particularly beneficial when the LLM's pre-trained knowledge may be outdated, incomplete, or lacks specialized information. In our cases we can include the domain specific knowledge of distinct processes of the German energy industry that way.

The first step in setting up a RAG system is *retrieval*, which involves preprocessing a variety of documents in varying formats such as html, pdf, and others which can be referred to as the external knowledge base. They are preprocessed by the embedding model extracting relevant information and transforming them to embeddings which are stored in a local vector database. In our experiments, we used the chromadb vector database (Chroma, 2023) for the implementation. There are a number of other implementations and providers of online vector databases(Langchain, 2024). We chose chromadb for our experiments as it is easily deployable locally, well documented and does not require to register with any online service.

Through semantic similarity calculations on the query embeddings and the embeddings stored in the vector database, relevant document chunks can then be retrieved from the external knowledge base.

In the *augmentation* phase the retrieved document chunks are integrated into the prompt. This augmented prompt combines the original user query with the additional contextual information from the external knowledge, ensuring the model's response is grounded in domain-specific data.

Finally, the *generation* phase refers to the creation of a response based on the augmented prompt. The LLM now uses not only its internal knowledge but also the retrieved domain specific knowledge.

Retrieval Augmented Generation is categorized as a prompt engineering technique(Tonmoy et al., 2024). As RAGs require to insert all available knowledge into the prompt, the utilized LLM must have a prompting window, also known as context length, of suitable size.

Similarity Matching describes the process of computing a similarity score of two pieces of text to determine how semantically similar those pieces of text are to each other. To achieve this, the texts are first handed over to an embedding model (see above) which computes numerical vector representations of them and then a distance metric (euclidian distance, cosine, dot product) is calculated for those vectors (embeddings). The value this metric yields is tantamount to the semantical similarity of the corresponding texts(Tunstall et al., 2022).

Chunking is a necessary pre-processing step in RAG systems during which documents get partitioned into smaller segments, that are called chunks or splits. These chunks are later embedded by an embedding model which means that this model generates numerical vector representations for the documents' chunks. The generated vectors, called embeddings, are then used for semantic similarity matching in high dimensional vectorspaces. The concept is, that those vectors whose corresponding textchunks are semantically similar to each other, are located closely to each other in a given vectorspace. The similarity of the textchunks is measured by applying one of serveral different distance measures to the embeddings in the vectorspace. The distance measure used for this can be the cosine of the angle between two vectors, the euclidian distance between the vectors or the dot product of the two vectors(Levy et al., 2024).

3 INFRASTRUCTURE / SYSTEM ENVIRONMENT

The decisions regarding the infrastructure address the dimension *performance* as well as *locality*. For our system environment, the most critical constraint was to ensure data confidentiality. No classified data (customer data, internal company data) was allowed to leave the local datacenter. Hence, we had to host our models (LLMs, Embedding Models) on-premise and make sure that no data was sent to any external API, such as OpenAI, Google or others. This meant we could not make any remote calls to, e.g., the OpenAI API and could not use strong third party models or rely on their computational power either.

To run our own local Large Language Models, an NVIDIA A100 80GB PCIe AI Accelerator(NVIDIA Corporation, 2023b) was available. This hardware was necessary due to the fact that it provides a sufficiently large amount of VRAM (80GB), as we wanted

Table 1: System environment specification

Virtual Machine	# vCPU-Cores	RAM	OS
VM1	28	252 GB	Ubuntu 22.04.03 LTS
VM2	2	32 GB	Ubuntu 22.04.03 LTS

to be able to run models with up to 70 billion parameters or multiple models (with up to 13 billion Parameters) at the same time. The AI Accelerator was made available to the VM via PCIe Passthrough. NVIDIA allows for sharding of AI Accelerators (Ampere Arcitecture or newer) through its Multi-Instance-GPU (MIG) Feature(Corporation, 2024), which we used to run multiple models in parallel.

As the runtime environment for the LLMs we chose FastChat(lm-sys, 2024) which supports running multiple models in parallel and offers an OpenAI compatible API(OpenAI, 2024). It was released under the Apache 2.0 license which allows commercial use. We also tested Ollama(Ollama, 2024) as a runtime environment alternative which proved especially useful for tests on our local developer devices

Our system environment was composed of two virtual machines. To achieve a clean architectural separation, the first VM hosted the LLM and its runtime environment. Furthermore it had access to the underlying AI Accelerator. This VM was provisioned with CUDA in version 12.2.(NVIDIA Corporation, 2023a). The second virtual machine hosted all other components, such as the applications' backend and frontend and communicated with the first VM via HTTP using REST APIs. The most relevant hardware specifications of our virtual machines can be found in Table 1.

4 CRITERIA FOR CHOOSING MODELS

The selection of models for our system was based on the following criteria:

1. The model is published under a license that allows modification as well as commercial use.
2. The model can be used locally in our system environment, which constrains the number of parameters of the model to a maximum value of 70 billion (unquantized) to fit in the VRAM of our AI Accelerator.
3. The model had to be pretrained, or at least finetuned on German language.
4. The model's context window is large enough to fit in all the required text from the retrieved documents.

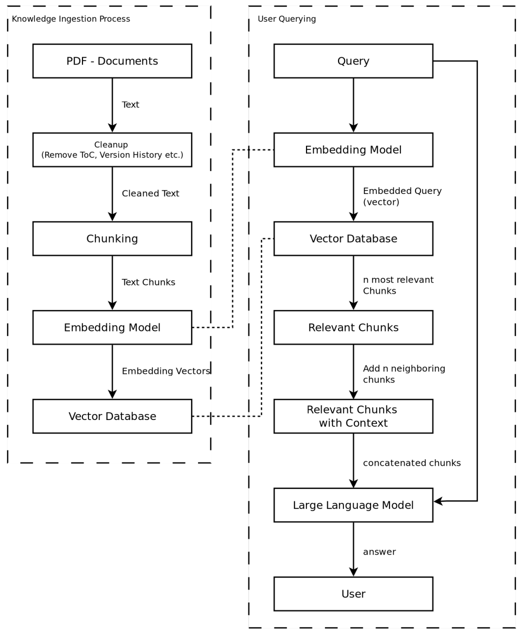

Figure 1: Processes of Ingestion and User Querying.

Another weak criterion is the number of parameters of the Large Language Models. It is of importance as larger models (in terms of parameters) tend to show better results in answer quality.

The same criteria and constraints applied for our embedding model. We started our experiments with standard models like *all-MiniLM-L6-v2* but quickly found that those English embedding models performed poorly for the German texts in our knowledge base and the German queries from our users. Therefore we decided to use *cross-en-de-roberta-sentence-transformer* an embedding model by T-Systems (T-Systems, 2024) which is finetuned for german language and performed reasonably well in our use cases. Later in the project, we also used the mxbai-embed-large-v1 by Mixedbread.AI (Mixedbread.AI, 2024) as that yielded even better results in our retrieval tests (see section 9).

Table 2 shows a comparison between the language models we tested. We first experimented with Vicuna in its 13B and 33B variants. While especially the 33B variant yielded good results in terms of answer quality, we had to discard it since its license does not allow for commercial use. We then switched to the Llama2 base models but those lacked in quality when it came to generating answers. The one exception to this was the 70B model. However, this variant was relatively slow and made further tests with multiple AI Accelerators running in parallel infeasible as it took up almost 100% of our VRAM.

In the end, we settled for the Llama-2-13B-chat-german (by jphme(Harries,)) model which is a fine-

Table 2: Comparison of candidates for utilized LLM.

Name	base / finetuned	Req. VRAM (8bit compr.)	Context Size	Licence	speaks German
Vicuna 33B	finetuned Llama2	48GB	4k Tokens	Non-commercial license	yes(well)
Llama2-13B	base	26GB	4k Tokens	Llama 2 license	yes(poorly)
Llama2-13B-Chat-german	fine-tuned	26GB	4k Tokens	Llama 2 license	yes(well)
Llama2-70B	base	76GB	4k Tokens	Llama 2 license	yes(mediocre)
Llama3-8B	base	13GB	8k Tokens	Llama 3 license	yes(mediocre)
Llama3-1-8B-Instruct	base	15GB	8k Tokens	Llama 3 license	yes(well)
Llama-3-SauerkrautLM-8b-instruct	fine-tuned	13GB	8k Tokens	Llama 3 license	yes(well)

tuned version of the corresponding 13B base model. Like the base model it takes up approximately 26GB of VRAM but at a much better ability to understand and output German language.

At the time of writing this paper we have also already tested the new Llama3 model family (including Llama3.1 but except the 405B versions, as they were too resource intense for our hardware), which was released by meta on April, 18th(Meta, 2024) and July 23th 2024 respectively(Touvron et al., 2024) as well as some of its German fine tuned derivatives like the Llama-3-SauerkrautLM-8b-Instruct model by Vago Solutions(VagoSolutions, 2024) which is published under Metas' Llama3 license(Meta, 2024).

5 SYSTEM ARCHITECTURE

From a process perspective, our RAG System can be divided in two main parts. The first part is the ingestion process of the documents and the second part is the user querying or inference (see Table 1). The ingestion process consists of two sub-processes that run in sequence. First, we preprocess the knowledge documents. This is necessary because each document contains a table of contents and a version history. Those components have to be removed as they can potentially be retrieved by our retrieval system, which feeds to the LLM, diminishing its answer quality.

After we clean up our documents, we chunk them. Chunking of the documents is necessary. Using the entire body of each document does not yield satisfactory embeddings that accurately reflect their semantic meaning. The larger the text that is to be embedded, the worse the precision of retrieved texts (by similarity matching the corresponding embeddings) becomes. This is because embedding models are trained on sentences of comparativly small or average sizes.

For instance, 85.1% of the sentences in the MultiNLI dataset (Williams et al., 2018), which was used together with the SNLI dataset for training SBERT, are at most 187 characters in length (1 Token roughly equals 3-4 characters, depending on the tokenizer used) as can be seen in (multiNLI, 2024).

For the user querying process we first embed the query using the same embedding model that the documents text chunks were embedded with. The embedded query is then used to perform a similarity matching against the embeddings within our vector database to determine which chunks of text are most similar and therefore most relevant to the users query. A number of n chunks is then retrieved from the database. Those retrieved chunks are comparatively small (max. 256 tokens) to keep the accuracy of the similarity matching high (see section 2).

While those small chunks are likely to represent information that is relevant to the querys' answering, it is also likely to be too little information to answer it fully. We mitigate this problem by also retrieving a number m of chunks before and after the chunk that was matched, creating a context window for each relevant chunk. This context window provides the LLM with enough information to generate a meaningful answer while preserving the relevancy of the information.

After retrieving the relevant chunks and expanding the context windows around them, enriched chunks are concatenated with each other to form the final context. This context is concatenated with the users original query forming the prompt that is fed to the LLM. The LLM is thereby provided with a prompt containing the users query as well as the relevant information for answering it.

Our system was implemented using Langchain(Chase, 2022) for the backend and streamlit for the frontend(Streamlit, 2022). Langchain is known as a swiss army knife framework as it offers extensive functionality for all kinds of use cases, not limiting itself to only RAG. An alternative to Langchain is llama-index(Liu, 2022). Llama-index is more tailored to RAG systems as it specializes in good retrieval of documents in large datasets. However, we chose to use Langchain for our applications as it provides out of the box implementations of features such as managed chat history which makes setting up chatbots easier.

To better integrate our RagBot system into the automatic evaluation process, we developed a wrapping API. This API allows for document retrieval and answer generation. As outlined in Section III, we utilized Fastchat as the runtime environment for our models. Fastchat offers an OpenAI-compatible API, which we access via Langchain. To expose the functionality written with Langchain as a REST API, we used the Flask framework(Pallets Projects, 2010).

The /documents endpoint of the API supports both GET and POST methods. The GET method returns documents matching a given search query, while the POST method performs document retrieval using vector search within a specified context window. The /llm_answer endpoint supports POST requests, generating answers using the LLM based on the provided query.

Both the Streamlit-based frontend and the RagBot API share the same underlying models and implementation. However, they differ in response delivery: the frontend UI streams tokens to the user as they are generated, while the API returns the entire generated answer in a single response after processing is complete. The latter approach, while leading to response times of several seconds, however is suitable for evaluation systems like Promptfoo, which usually handles large batch runs.

6 USE CASES

We examined two use cases. As a first test of our architecture and model choices, we built a conversational agent for the ”Marktstammdatenregister” (MaStR)(Bundesnetzagentur, 2019).

The Marktstammdatenregister is a comprehensive, centralized database in Germany, managed by the Federal Network Agency (”Bundesnetzagentur”). It records detailed information about all energy-producing units, including renewable and conventional sources, and selected energy-consuming units (in terms of the MaStR every device, which can or must be registered, such as a PV-system or a battery, is called a *unit*). The registry aims to improve market transparency, facilitate the planning of energy infrastructure, and support regulatory and policy-making processes. It ensures compliance with EU regulations by providing a reliable and consistent dataset for market participants and authorities. The implemented conversational agent demonstrated the capability to provide comprehensive responses. These responses cover all customer-relevant requests pertaining to processes facilitated by the Marktstammdatenregister. Although the use case was initially dedicated to end user support, it serves well as a coaching assistant for new service operators responsible for the MaStR customer support. In the remainder, this use case is referred to as ”MaStR”.

The second and more comprehensive use case is an employee coaching assistant for a big German network operator, where several hundred of employees as well as outsourced personnel are responsible to provide first, second and third level support for German energy customers. Customer service is delivered via multiple channels such as phone calls, email, mail, chat conversations, and more. The employee turnover rate is at 20 to 30 percent per year, so about one quarter of employees must be on-boarded year by year. On-boarding contains and is structured by the network operators processes, e.g., change of operator, master data change, calculation of advance payment and others. In the following, this use case is referred to as ”Network Operator”.

7 DATA

The dataset for the use case MaStR comprises 27 documents, and in addition one FAQ document, all of which provide assistance and manuals for MaStR users in various situations, such as the user sign up, registration of installations, or master data management of units. These documents are predominantly step-by-step guides and manuals, such as registration aids. Challenges arose primarily from content and revision tables, which were frequently included in the generated responses. Additional challenges came from screenshot-based guides or documents heavily comprised of images, which are not adequately processed by a language model. Consequently, manual preprocessing involved the removal of content and revision tables, as well as documents predominantly composed of images.

The data provided for the use case Network Operator was the process documentations of the researched network operator processes, of which 19 were investigated. The process descriptions are mostly stored in the organisation’s Confluence-based wiki. However additional documentation in separate files or even as part of Emails or other sources may exist. The respective documents were exported from the companie’s wiki as PDF documents and had an overall size of 34MB. The documents contain typical features of process documentations such as headers and footers with logos, tabular content summaries, outlines, and tabular version histories. All of them are ambiguous inputs for the embedding model as well as the LLM. On the one hand these parts are mostly redundant and thus disturbing for the model. On the other hand they contain important data for the document retrieval step. For our POC these inputs were removed manually. Besides the mentioned overhead information the process documents contained lots of screenshots. For the generation of the embeddings we omitted these screenshots, however in the final answer of the assistant the screenshots where included. All texts were written in German language.

8 EXPERIMENTS

The procedure for conducting our experiments differed between our two use cases MaStR and Network Operator. For the MaStR use case the experiments were done manually, whereas we did both manual and automatic experiments for the Network Operator use case. For both use cases we needed a ground-truth, something that we could compare the results of our experiments with. The following describes what we did on a per use case basis.

For our first use case, *MaStR*, various experimental configurations were tested to determine the accuracy and reliability of the models. As the ground truth for this use case we used a document containing 90 frequently asked questions (FAQs) and their corresponding answers that were checked by domain experts and confirmed to be correct. They were used to compare different configurations of the model to observe consistency of improvements. An experiment run posted all 90 questions to the model and the results were compared to the given answers of the FAQ. We used a binary rating scale where 0 denotes failure (the answer didn't suffice) and 1 denotes pass (the answer mostly reflects the information of the given answer). The result rating of a run was calculated by averaging the binary results of all questions, leading to a value between 0 and 1. Testing different temperature setting for the modell led to the following results. A temperature of 0.5 yielded an average rating of 0.64, compared to 0.57 at a temperature of 0.1. Thus, a temperature of 0.5 yielded most satisfactory results in our setting fostering the decision to further utilize a temperature setting of 0.5 for our experiments to determine the stability of generating good responses.

For the second use case, Network Operator, our objectives were twofold. First, we wanted to assess the quality of the retrieval step of our RAG system. That means we needed to test for its precision and recall concerning the documents, or document chunks to be precise, retrieved from our vector database as those formed the context from which the LLM generated its answers. The second objective was to test for the quality of the generated answers. This test was essential to evaluate whether the LLM would be able to formulate a correct and coherent answer based on the retrieved context given to it in the preceding step. For both of these experiments we used the same document as the ground truth. This document was a catalog containing:

1. 1000 assumed questions from network operator trainees,
2. the names of the documents containing the necessary information for answering these questions, as well as
3. a preformulated correct answer for that given question.

The evaluation for both objectives was based on automatic validation using Promptfoo(Webster, 2024). Leveraging Promptfoo we checked for precision and recall in the retrieval step by first feeding the expected documents from the aforementioned catalog for each question to it. It then queried the sytem with the sample questions and matched the retrieved documents against the expected ones and finally calculated the precision and recall scores.

For the evaluation of the generated answers by the LLM we needed to implement some custom logic that promptfoo used for its checks. To check whether or not generated answers matched our known-good answers from the catalog we could have used a simple string matching (containing steps like trimming and making both texts lower case etc.) but this would have meant that both answers (known-good and generated) would have had to be exactly the same, to the letter. Instead we decided to check for similarity between the two answers. Accordingly, we had our embedding model generate embeddings for each known-good answer as well as each generated answer and calculated cosine similarity of the embeddings to see how similar the generated answer was to the corresponding known-good answer.

9 RESULTS

After describing our system, the used data and the performed experiments, the results will be described along the four initially introduced dimensions.

Performance. We determined the overall performance of our system by measuring the response times from submitting a query to the moment the final respone was returned. This measuring happened in promptfoo which used our systems API. Response times of our system ranged from 3.8 s to 43.5 s with a mean response time of 8.7 s (all values rounded to one decimal place). The measured times are also shown in Figure 2.

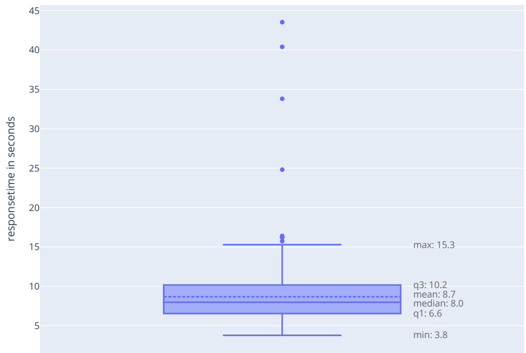

Figure 2: Response times during generation step (500 samples).

The majority of the measured times (75%) were lower than 10.2 s. It is worth noting that for the sake of conducting the correctness tests of our system, it was unnecessary to stream the LLM's responses. However, in our UI, the responses were streamed to the user. This reduced the perceived response times significantly in each case, as the users could immediately see the system's responses as they were being generated.

We also discussed taking measurements for the time it takes the embedding model to create embeddings but quickly found that those times were in the milliseconds and therefore insignificant. Especially during the generation of our vector database steps like reading in the pdf documents took considerably longer than the generation of embeddings which made measuring the latter irrelevant. Additionally, generating the vector database happens only whenever our knowledge base changes which is a.) rare and b.) plannable and can therefore be done during times of low usage leading to low impact on our users. Whilst each query a user submits has to be embedded as well, this too only takes a few milliseconds and does not noticeably impact the user experience.

Correctness: In the case of the Marktstammdatenregister (MaStR) the correctness was measured manually by having domain experts compare the LLM-generated answers to known-good answers in an FAQ document as mentioned in section 8.

Higher temperatures often led to more off-topic responses hence lowering the rating. During the experiment critiques of the FAQs emerged, as they were not always unambiguous. This pattern was also found in the second examined use case underlining that gathering sufficient and correct input data is crucial for provisioning of AI agents.

For the Network Operator case we took measures for the retrieval and the generation steps of the RAG system. For the retrieval we measured:

1. precision of the retrieval of documents (#correctly retrieved documents / #all retrieved documents)
2. recall of the retrieved documents (#correctly retrieved documents / #correctly retrieved documents + #incorrectly not retrieved documents)

To calculate the precision of our RAG system, we needed a ground truth baseline to which we could compare the system's answers. As mentioned in section 8, we used a catalog for this containing a sample set of questions, the documents that should be retrieved for a given question, as well as a known-good answer for each question. Using a randomly chosen subset of 500 samples from the original catalog, our system reaches a 90.8% precision rate during for retrieval step of the RAG-System.

As for the generated answers, when running our tests for semantic similarity between them and our known-good answers, we received test pass rates from $\sim 21.6\%$ to $\sim 40.2\%$ depending on how large we chose the context window size (the amount of text added before and after each retrieved chunk). However, despite these test results, manual revision of the generated answers often showed that even the answers that failed the similarity tests still contained large parts of the correct answer and only seemed to fail the tests due to additional LLM-output that followed the actual answer.

We compiled a short list of example questions, the corresponding known-good answers, as well as the answer generated by the LLM for that question. This is shown below, in Table 4.

It is clear from the data that increasing the number of chunks as well as the context window around those chunks positively impacted answer quality. Both means increase the context that the LLM can work with to generate its answers. The upper limit for the size of the context window is given by the size of each document that the retrieved chunk belongs to. However, while it may seem reasonable to continue increasing the context window around each chunk one has to consider that this may also lead to an increase of information given to the LLM that is irrelevant to answering the original question.

In general, the upper limit of the size of the context given to the LLM is constrained by the respective models context window, which, in our case, was 4000 Tokens. Exceeding this context window will result in truncation and therefore the LLM will not take all the information into account. However, context windows of LLMs have shown to be increasing in size with each new generation of models so this may not be a restraining factor in the future anymore.

Integration: To integrate our system in our users everyday workflows we designed the RAG system as

Table 3: Performance results.

model	avg speed / test run
jphme/Llama-2-13b-chat-german	11,86 s
meta-llama/Meta-Llama-3.1-8B-Instruct	8.7

a chatbot running on a simple, minimalistic website. This website can be opened in an additional browser window or tab on the desktop of the employee so it integrates seamlessly with the software they are running anyway. In addition integrating the RAG system into different legacy systems is ensured as we are leveraging standard technologies like Langchain and established interfaces such as the OpenAI API.

Locality: The whole RAG system is deployed on machines in the local data center of regiocom. Therefore a significant investment in the AI accelerator card was necessary. However, as current research indicates the availability of much smaller models, both for embedding as well as language generation, future investments may be much lower.

We conclude that leveraging locally running embedding models for information retrieval as well as large language models for summarization and answer generation is a suitable means for individual employee coaching and can help relieve the burden on network operators.

10 FUTURE WORK

For the first three of our four main dimensions, there are several tasks for future development. The requirement of **Locality** is fulfilled with the current architecture, and for the moment no future work needs to be identified.

Performance. There are several areas for potential improvement in system performance:

- Alternative Runtime Environments: While our current implementation uses FastChat as the runtime environment, exploring other options such as Ollama or LoraX could potentially yield performance benefits. A comparative analysis of different runtime environments could help identify the most efficient solution for our use case.
- Load Testing: To ensure the system can handle real-world demands, it's crucial to conduct comprehensive load tests simulating scenarios with hundreds or thousands of concurrent users. This will help identify potential bottlenecks and optimize the system's ability to scale under high load conditions.
- Response Time Optimization: Although our current best mean response time of 8.7 seconds with the Llama3.1-8B-Instruct model (as shown in Figure 2) is acceptable for many use cases, there's room for improvement. Investigating ways to reduce latency, as the retrieval process is fast enough in our case exploring more efficient models or model quantization could enhance the user experience, especially for time-sensitive queries.

By addressing these performance-related aspects, we aim to create a more responsive, scalable, and efficient system capable of meeting the demands of a large-scale deployment in a network operator environment.

Correctness. To improve the accuracy and reliability of our system, we have identified the following key areas for future work:

- Enhanced Chunking Strategies: Our current chunking method for document processing can be refined. Developing more sophisticated chunking algorithms that better preserve context and semantic coherence could lead to improved retrieval accuracy. This might involve experimenting with various chunking sizes, overlapping chunks, or semantic-based chunking methods.
- Improved Data Preprocessing: The quality of our input data significantly impacts the system's performance. Future work should focus on developing more robust preprocessing pipelines. This could include better handling of tables, headers, footers, and image captions, as well as improved methods for extracting relevant information from complex document structures.
- Expansion of Expert Evaluation Datasets: To more rigorously assess our system's correctness, we need to expand our collection of expert-curated evaluation datasets. This involves:
  – Collaborating with domain experts to create a larger, more diverse set of question-answer pairs.
  – Developing more comprehensive test scenarios that cover a wider range of use cases and edge cases.
  – Regularly updating these datasets to reflect changes in processes and regulations within the energy sector.
- Refinement of Evaluation Metrics: While our current evaluation methods provide valuable insights, there's room for improvement. Future work could involve:
  – Developing more nuanced metrics that go beyond simple similarity scores, possibly incorporating domain-specific evaluation criteria.

– Implementing automated systems for continuous evaluation and monitoring of the system's correctness over time.

By focusing on these areas, we aim to significantly enhance the correctness and reliability of our RAG system, making it an even more valuable tool for employee coaching and support in the network operator environment.

Integration. Workstation desktops of service center employees usually contain significantly more than one open window. Depending on the environment and the complexity of the respective task, three, four or more applications fill the screen. Thus, an integration into one single application (e.g., an ERP System like SAP) is likely to be incomplete. For now, the solution to provide the assistant as a web-based application is a bearable compromise. As the whole application is tailored in an API-first manner, switching UIs or integrating it into another UI is possible and straightforward.

11 CONCLUSIONS

This paper introduced a locally deployed RAG system for employee coaching in the energy sector, focusing on network operator processes. Our key contributions include the following:

- Development and implementation of an on-premise AI coaching system for the energy sector
- Evaluation of system performance across correctness, performance, integration, as well as data and runtime locality

Our findings show that local embedding models for retrieval, combined with LLMs for answer generation, can effectively support individualized employee coaching. The system achieved a 90.8% precision rate for the retrieval of relevant documents from our knowledge base. While promising, these results need further improvement to be used in a productive environment. The potential impact of this work extends beyond immediate performance metrics. As AI technologies evolve, such systems could significantly enhance knowledge management and employee training in the energy sector. However, challenges remain in data preprocessing, model choices and configurations. In conclusion, our work represents a step towards practical AI application for customer service in the energy sector. While the results are encouraging, further research and refinement are necessary to reach a production-ready quality for our assistant.

ACKNOWLEDGEMENTS

This work was funded by regiocom SE, Germany. The authors would like to express their gratitude to regiocom for their financial support and for providing access to the necessary data and resources that made this research possible.

REFERENCES

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Bundesnetzagentur (2019). Marktstammdatenregister. https://www.marktstammdatenregister.de. © Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen — DL-DE-BY-2.0.

Chase, H. (2022). LangChain.

Chroma (2023). Chromadb. Available at: https://www.trychroma.com/.

Corporation, N. (2024). *NVIDIA Multi-Instance GPU User Guide*. Accessed: 2024-08-16.

Fu, Z., Lam, W., Yu, Q., So, A. M.-C., Hu, S., Liu, Z., and Collier, N. (2023). Decoder-only or encoder-decoder? interpreting language model as a regularized encoder-decoder. https://arxiv.org/abs/2304.04052.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M.-W. (2020). Realm: Retrieval-augmented language model pre-training.

Harries, J. P. https://huggingface.co/jphme/Llama-2-13b-chat-german. Accessed on 2024-08-05.

Joshi, V., He, W., sun Seo, J., and Rajendran, B. (2021). Hybrid in-memory computing architecture for the training of deep neural networks.

Langchain (2024). https://python.langchain.com/v0.2/docs/integrations/vectorstores/. Accessed on 2024-08-03.

Levy, A., Shalom, B. R., and Chalamish, M. (2024). A guide to similarity measures.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.

Liu, J. (2022). LlamaIndex.

lm-sys (2024). FastChat: An open platform for training, serving, and evaluating large language model based chatbots.

Meta (2024). https://llamaimodel.com/commercial-use/. Accessed on 2024-08-01.

Meta (2024). Introducing meta llama 3: The most capable openly available llm to date. https://ai.meta.com/blog/meta-llama-3/. Accessed on 2024-08-01.

Mixedbread.AI (2024). mixedbread-ai/mxbai-embed-large-v1. Accessed on 2024-09-12.

multiNLI (2024). https://huggingface.co/datasets/nyu-mll/multi_nli. Accessed on 2024-08-020.

NVIDIA Corporation (2023a). *CUDA Toolkit Documentation*. Accessed: 2024-08-16.

NVIDIA Corporation (2023b). Nvidia a100 tensor core gpu. Datasheet, NVIDIA Corporation. Accessed on 2024-07-21.

Ollama (2024). Ollama: A platform for running large language models locally. https://github.com/ollama/ollama. Accessed: 2024-08-16.

OpenAI (2024). Openai api. https://platform.openai.com. Accessed: 2024-08-20.

Pakmehr, A., Aßmuth, A., Neumann, C. P., and Pirkl, G. (2023). Security challenges for cloud or fog computing-based ai applications.

Pallets Projects (2010). Flask: Web development, one drop at a time. https://flask.palletsprojects.com/. Accessed: 2024-08-19.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Rakovac Bekeš, E. and Galzina, V. (2023). Exploring the pedagogical use of ai-powered chatbots educational perceptions and practices. In *MIPRO 2023*, pages 636–641, Opatija, Croatia. IEEE.

Streamlit (2022). Streamlit. https://github.com/streamlit/streamlit. Accessed on 2024-08-01.

T-Systems (2024). https://huggingface.co/T-Systems-onsite/cross-en-de-roberta-sentence-transformer. Accessed on 2024-08-01.

Tonmoy, S. M. T. I., Zaman, S. M. M., Jain, V., Rani, A., Rawte, V., Chadha, A., and Das, A. (2024). A comprehensive survey of hallucination mitigation techniques in large language models.

Touvron, H., Martin, L., Stone, K., et al. (2024). The llama 3 herd of models. Available at https://ai.meta.com/research/publications/the-llama-3-herd-of-models/.

Tunstall, L., von Werra, L., and Wolf, T. (2022). *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media.

Turnbull, D., Chugh, R., and Luck, J. (2019). *Learning Management Systems: An Overview*, pages 1–7. Springer International Publishing, Cham.

VagoSolutions (2024). https://huggingface.co/VAGOsolutions/Llama-3-SauerkrautLM-8b-Instruct. Accessed on 2024-08-01.

Webster, I. (2024). promptfoo. https://www.promptfoo.dev/. Accessed: [2024-07-21].

Williams, A., Nangia, N., and Bowman, S. (2018). A broad-coverage challenge corpus for sentence understanding through inference. In Walker, M., Ji, H., and Stent, A., editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Yenduri, G., M, R., G, C. S., Y, S., Srivastava, G., Maddikunta, P. K. R., G, D. R., Jhaveri, R. H., B, P., Wang, W., Vasilakos, A. V., and Gadekallu, T. R. (2023). Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions.

Table 4: Comparison of known-good answers to RAG generated answers.

Question	known-good answer	LLM generated answer
Wann erfolgt ein Betreiberwechsel?	*Ein Betreiberwechsel kommt dann Zustande, wenn eine Einspeiseanlage veräußert oder durch Erbe/Schenkung an eine andere Person übergeben wurde.*	*Ein Betreiberwechsel erfolgt, wenn eine Einspeiseanlage veräußert oder durch Erbe/Schenkung an eine andere Person übergeben wird.*
Wann ist die zusätzliche Pflege als abweichender Rechnungsempfänger zulässig?	*Eine zusätzliche Pflege als abweichender Rechnungs- und Mahnempfänger ist nur bei direkten Netzendkunden (Selbstzahlern) und Einspeisern zulässig, sofern dies vom Kunden gewünscht ist.*	*Nach den bereitgestellten Informationen ist eine zusätzliche Pflege als abweichender Rechnungsempfänger zulässig:* *1. Bei direktem Netzendkunden (Selbstzahlern)* *2. Bei Einspeisern, sofern dies gewünscht ist.* *Bitte beachte, dass diese Informationen auf den bereitgestellten Kontext basieren und sich ändern können, wenn sich die Situation ändert. Wenn du weitere Fragen hast, stehe ich gerne zur Verfügung.*
Was bedeutet Handlungsvollmacht?	*Handlungsvollmacht bedeutet, dass der Bevollmächtigte sowohl Auskünfte erhalten darf als auch selbst Änderungen vornehmen lassen kann.*	*Handlungsvollmacht bedeutet, dass der Bevollmächtigte sowohl Auskünfte erhalten darf als auch selbst Änderungen vornehmen lassen kann.*
Wie gehe ich bei einer schriftlichen Anfrage vor, wenn der Vorgang innerhalb von KK, aber nicht vom aktuellen Servicepartner bearbeitet werden kann?	*Bei einer schriftlichen Anfrage klassifizierst du diese um, so dass diese an den Servicepartner geroutet wird, der den Prozess anschließend bearbeitet.*	*Bei einer schriftlichen Anfrage, die innerhalb von KK, aber nicht vom aktuellen Servicepartner bearbeitet werden kann, musst du den Vorgang umklassifizieren, so dass diese an den korrekten Fachbereich geroutet wird, der den Prozess dann bearbeiten kann.*