O2-FLP group
ALICE collaboration

22 September 2025
Summer student project report
tjas.ajdovec@gmail.com

TECHNICAL NOTE

Chatbot for ALICE Run3 simulation and analysis tasks

Summer student: Tjaš Ajdovec

Supervisors: Matteo Concas, Marco Giacalone and Sandro Wenzel

CERN, Geneva, Switzerland

Keywords: Chatbot, LLM, RAG, LangChain, O^2 framework, simulation, analysis, Mattermost

Summary

In the scope of this project we developed AskALICE, an open-source, local chatbot to assist users with ALICE Run3 simulation and analysis tasks within the O^2 framework. We implemented a RAG pipeline using LangChain, ChromaDB and free models downloaded from HuggingFace. We served the LLMs with llama.cpp on limited AMD hardware. Our knowledge base consists of 400 documents, presentations and transcribed talks. Our evaluation dataset contains 35 question-answer pairs provided by experts. To evaluate the answer grading ability of LLMs, we correlated their inputs to those of Gemini-2.5-Flash API model. We selected four answer correctness metrics (LLM-as-judge score, embedding semantic similarity, ROUGE-L score and BLEU score) and evaluated 7 different models (Qwen2.5, Gemini, Gemma, Mistral, DeepSeek, Gpt, Qwen3) before and after RAG. Our pipeline significantly improved the results and Qwen3-30B-A3B-Instruct performed the best. We also optimized database retrieval parameters and explored the trade-off between performance and response time. We made our chatbot available in Mattermost as user `@askalicebeta` and defined configurable components. We set up a LangFuse server for call tracing and collecting user feedback. In the last Chapter 12 we described our findings, limitations and provided guidelines for future work. Source code and documentation are available at: `https://github.com/ta5946/alice-rag`.

Contents

1 Motivation **3**

2 Existing chatbots **4**

3 Retrieval augmented generation **4**

4 Knowledge base **5**

5 Embedding and reranking models **5**

6 Open-source large language models **6**

7 Text similarity metrics **6**

8 LLM judge evaluation **7**

9 Chatbot evaluation **8**

10 Working examples **11**

11 Call tracing and observability **12**

12 Findings and future work **14**

13 Acknowledgments **15**

A RAG prompts **17**

1 Motivation

Physics analysis at the ALICE experiment relies on Monte Carlo (MC) simulations, which consume significant computing resources, utilizing hundreds of thousands of servers and generating petabytes of data. The MC software ecosystem is vast, with a huge amount of lines of C++ and Python code, shell scripts, and documentation, making user support challenging. This project aims to leverage recent advancements in large language models (LLMs) to train an AI-powered chatbot that aggregates this dispersed information into a centralized resource for automated user support. The student will work on document collection, expert knowledge base development, and chatbot training. Integration into CERN's communication platforms, such as Mattermost, will also be explored, alongside evaluating different deployment options for LLM servers. This project has the potential to significantly enhance user support and serve as a model for AI-driven expert systems across ALICE operations.

2 Existing chatbots

Existing chatbot solutions, such as **ChatGPT**, Google's Gemini and Mistral's Le Chat, have some knowledge of our domain as they were generally pretrained on large-scale web data; including public ALICE GitHub repositories. However, they are not specialized for our use cases and have a knowledge cut-off point at the start of their training. The supporting LLMs are only updated a couple of times per year, which is not enough to ensure up-to-date information. They also do not know about our private repositories or documents, are closed-source and expensive to rely on, especially with increasing request prices. These are the reasons we decided to develop an open-source, local chatbot, specifically designed to assist users with Run3 simulation, data processing and analysis tasks in the context of O^2 framework.

3 Retrieval augmented generation

Retrieval augmented generation (**RAG**) is an engineering technique to adapt any local or remote LLM to our knowledge base, without needing to retrain or fine-tune it. So it works with the base model, domain-specific data is simply passed as part of the context or user message. This way the knowledge base can be updated frequently, even daily or on documentation change. Compared to alternative approaches, RAG is cheap in terms of computing and is appropriate when our number of documents is in the range of thousands. For implementation, it requires 5 key components:

1. **Documents** in any format that can be parsed into text with high accuracy. This includes markdown and word files, presentation slides and PDFs.

2. A **vector database** that can store <text, embedding> pairs and efficiently handle queries based on cosine or some other vector distance. Due to wide support we decided to use ChromaDB.

3. A text **embedding and reranking models** which can order the documents by relevance or similarity for a given search query.

4. **LLM** responsible for generating interim steps and the final chatbot response.

5. And lastly a **user interface**, in our case a dedicated Mattermost chat. It is currently available under the username `@askalicebeta`.

Besides that, we used LangChain, an orchestration framework that contains prebuilt abstractions and helper functions for RAG and other LLM-based pipelines. The system diagram below (Figure 1) shows the interaction between different components and three main stages in the pipeline: document retrieval, context augmentation and answer generation.

Before deploying the chatbot our documents are split into text chunks of a fixed size (1000 characters) and transformed into feature vectors with an embedding model. Documents processed in this way are stored into the vector database. On invocation, the user question also gets transformed into a query vector, which is used to retrieve the (`top_n`) most relevant documents. Both the original question and the retrieved documents are then passed to the

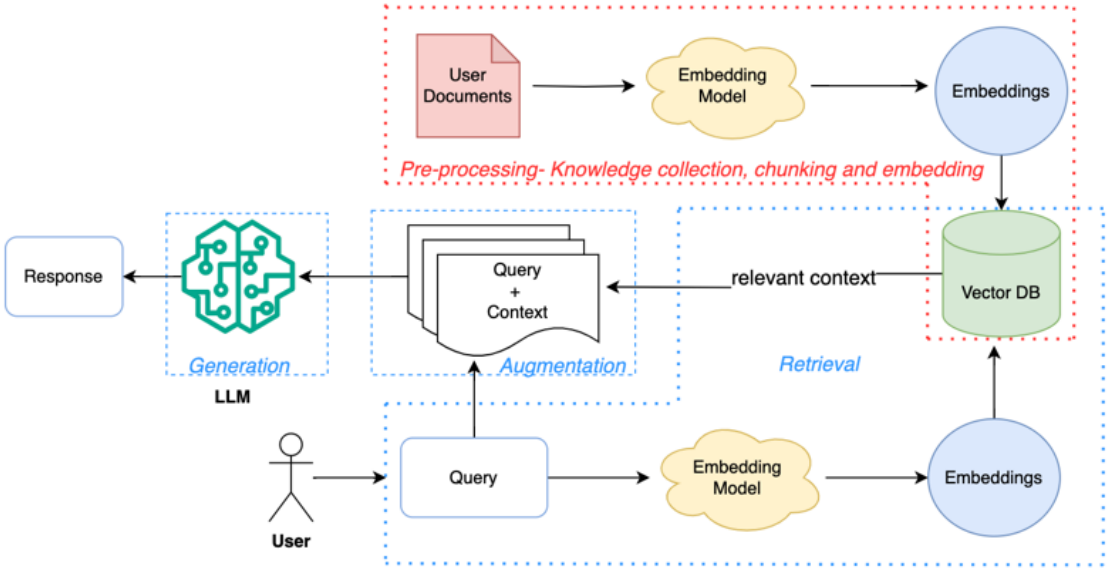

Figure 1: Basic RAG architecture and three main pipeline stages.

LLM, which is instructed to generate an answer. We often prompt the model to only answer based on the provided context to ensure consistency with our knowledge base.

4 Knowledge base

For our use case we wrote a GitHub scraper for ALICE O^2 simulation, data processing and analysis framework repositories, as well as some other script examples. We parsed presentation PDFs and transcribed talks from experts. This totaled to around **400 documents** or 3500 chunks of "training" data.

To capture the problem domain we used an LLM and Mattermost API to collect real user question from the O2 Simulation channel. Most importantly, supervisors provided **35 question-answer pairs** that were used as a reference point in the chatbot evaluation. This enables us to measure the difference between the chatbot response and the gold-standard answer from domain experts.

5 Embedding and reranking models

Models used for document feature extraction and cross ranking are available to download for free on HuggingFace. When selecting the initial text embedding model we considered factors such as: lightweight ($<$ 250 M parameters), number of downloads and average score on the MTEB benchmark. Based on these criteria we selected `BAAI/bge-base-en-v1.5` and the corresponding `BAAI/bge-reranker-base` and loaded them at runtime with the transformers library.

6 Open-source large language models

HuggingFace also offers many free LLMs. At the start of the project we only had access to a single NVIDIA GeForce RTX 2080 GPU with 8 GB of VRAM, which posed a hardware constraint. Therefore, we selected between models smaller than 8 B (billion parameters), while also taking into account the MMLU benchmark, LMArena leaderboard and preliminary tests for our use case. We decided on the `Qwen2.5-7B-Instruct-GGUF:Q6_K` and served it with llama.cpp. This runs the model in a containerized Docker environment and exposes an OpenAI compatible API at a defined URL, by default this is `http://pc-alice-ph01:8080/v1`.

We can easily connect to the LLM from our application using a generic LangChain class:

```python
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="Qwen2.5-7B-Instruct",
    base_url="http://pc-alice-ph01:8080/v1",
    api_key="any"
)

respone = llm.invoke("What is ALICE O2?")
print(response.content)
# ALICE is an acronym for A Large Ion Collider Experiment, which is
   one of the major experiments at CERN...
```

At a later stage in the project we obtained access to a cluster of AMD MI100 GPUs with 32 GB of VRAM each. This enabled us to run larger LLMs of sizes up to 32 B, which is still not close to the most capable open-source models. For example, DeepSeek-R1 uses over 700 B parameters. Table 1 shows the selected medium-sized models and their characteristics. Most of them are served in GGUF format with 6-bit quantization to leave some space for the input and cache tokens.

LLM	Description
Qwen2.5-7B-Instruct	Smaller Qwen model used for chatbot development
gpt-oss-20b	OpenAI's only open-source reasoning model
Qwen3-30B-A3B-Instruct-2507	Qwen model designed for efficient inference
Mistral-Small-3.2-24B-Instruct-2506	Updated Mistral appropriate for our requirements
gemma-3-27b-it	Google's latest open-source model
DeepSeek-R1-Distill-Qwen-32B	Hard reasoning model trained by DeepSeek-R1

Table 1: List of locally served LLMs and their descriptions.

7 Text similarity metrics

Before going into the chatbot evaluation, we first have to define some text similarity metrics, which we used to compare the generated answer to the correct one:

1. **LLM-as-judge score** is a simple metric, where we prompt another LLM to grade the generated answer against the reference on some scale, in our case $\in [1, 5]$. It is based on the learned judgement and reasoning ability of the model and can be useful as it mimics human intuition.

2. **Embedding semantic similarity** is derived from the cosine distance between normalized feature vectors, obtained by passing the text to the previously mentioned embedding model. Cosine similarity is calculated as $1 -$ distance and takes up values $\in [0, 1]$.

3. **ROUGE-L score** is a low-level metric that measures the maximum word sequence overlap between the generated and correct answer. It should not be used as a primary metric as it does not capture semantic meaning and text structure, but operates on a word-level.

4. **BLEU score** is similar, but precision-based n-gram overlaps between the answers. Values for both metrics are in range from 0 (no match) to 1 (exact match).

8 LLM judge evaluation

Not all LLMs are trustworthy when it comes to consistently grading the generated answers and selecting a good judge is not trivial. For this reason we answered 25 questions with 4 different prototype chatbots, such as base `Gemini-2.5-Flash`, base Qwen2.5-7B-Instruct and the same Qwen with our RAG pipeline. We scored these answers with potential LLM judges which resulted in 100 data points $\in \{1, 2, 3, 4, 5\}$ for each. We calculated correlations between all judges using the Pearson coefficient, Spearman coefficient and Root Mean Squared Error (RMSE). To determine the most capable local judge, we observed the **correlations with Gemini**, which is the largest and considered a state-of-the-art model. Based on the evaluation results presented in Table 2, Qwen3-30B-A3B-Instruct grades the answers most similar to Gemini and was therefore selected as the judge for the final chatbot evaluation.

LLM judge	Pearson	Spearman	RMSE
Gemini-2.5-Flash (API)	1.000	1.000	0.000
Gemini-2.5-Flash-Lite (API)	0.766	0.768	0.684
Qwen3-30B-A3B-Instruct	**0.747**	**0.758**	**0.711**
gemma-3-27b-it	0.676	0.682	0.805
DeepSeek-R1-Distill-32B	0.641	0.644	0.848
gpt-oss-20b	0.622	0.604	0.869
Mistral-Small-3.2-24B-Instruct	0.571	0.570	0.926
gemma-2-9b-it	0.558	0.570	0.940
Mistral-7B-Instruct-v0.3	0.546	0.553	0.953
Qwen2.5-7B-Instruct	0.489	0.499	1.011
Meta-Llama-3.1-8B-Instruct	0.315	0.298	1.171

Table 2: Correlation coefficients and RMSE of LLM judges with Gemini-2.5-Flash.

Besides that, the model correlations are consistent across all three metrics. We also plotted a heatmap of Spearman coefficients, which measures answer ranking consistency independent of scale and offset. Pairwise correlations between all the LLM judges are available on Figure 2.

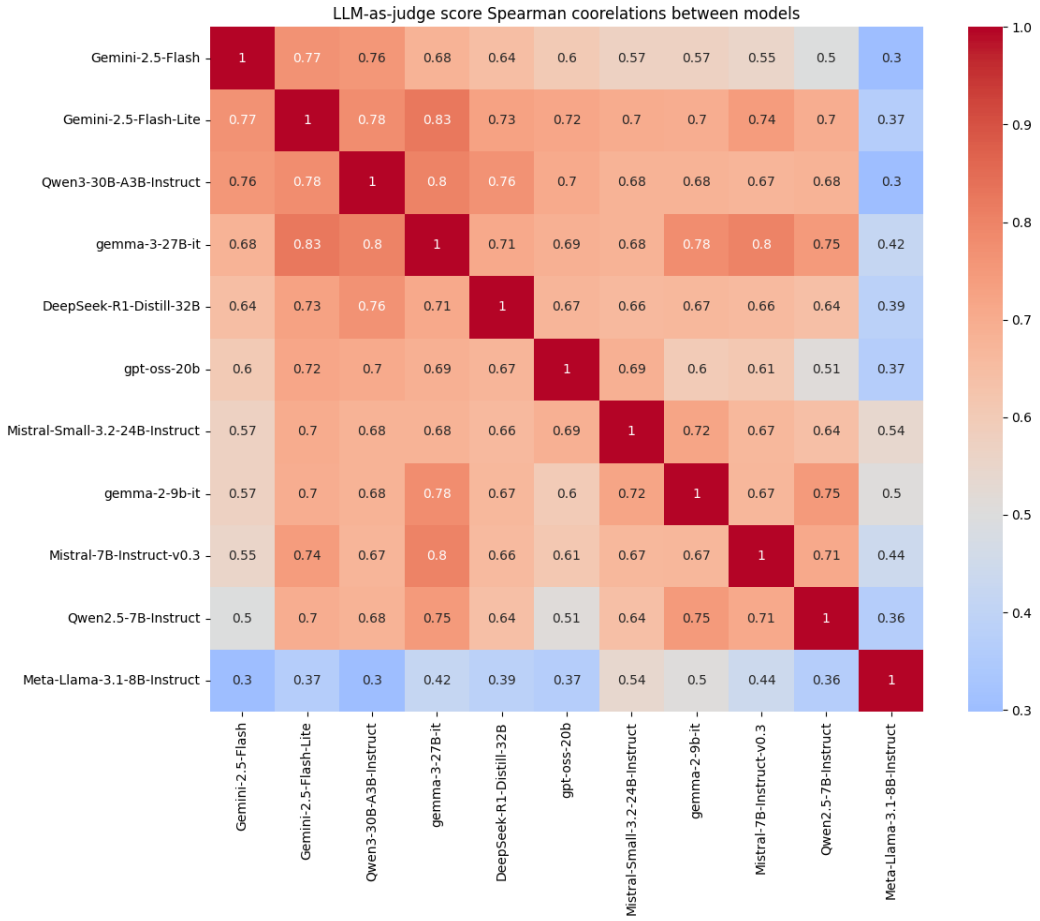

Figure 2: Spearman correlations heatmap between different LLM judges.

9 Chatbot evaluation

At this point, we had a RAG workflow able to produce answers and trustworthy chatbot evaluation metrics. We tested different RAG configurations, mainly models and database retrieval parameters. Due to LLM non-determinism (temperature $\neq 0$), we **sampled multiple answers** for each of the 35 questions in the evaluation dataset. The exact algorithm is described below:

1. Select a text similarity metric,
2. For or each question and correct answer in the evaluation dataset repeat steps 3 - 5,
3. Prompt the chatbot to generate N = 5 answers,

4. Compute the similarities between all generated answers and the correct one,
5. Calculate the mean (μ) and standard deviation (σ) of the similarity scores for this question,
6. Calculate μ and σ as averages across all questions.

This way we can treat the standard deviation as error and claim result significance based on that. First we compared answer correctness for all **language models** in Table 1, before and after using RAG. We only evaluated the base Gemini-2.5-Flash as it was in the free tier rate limit.

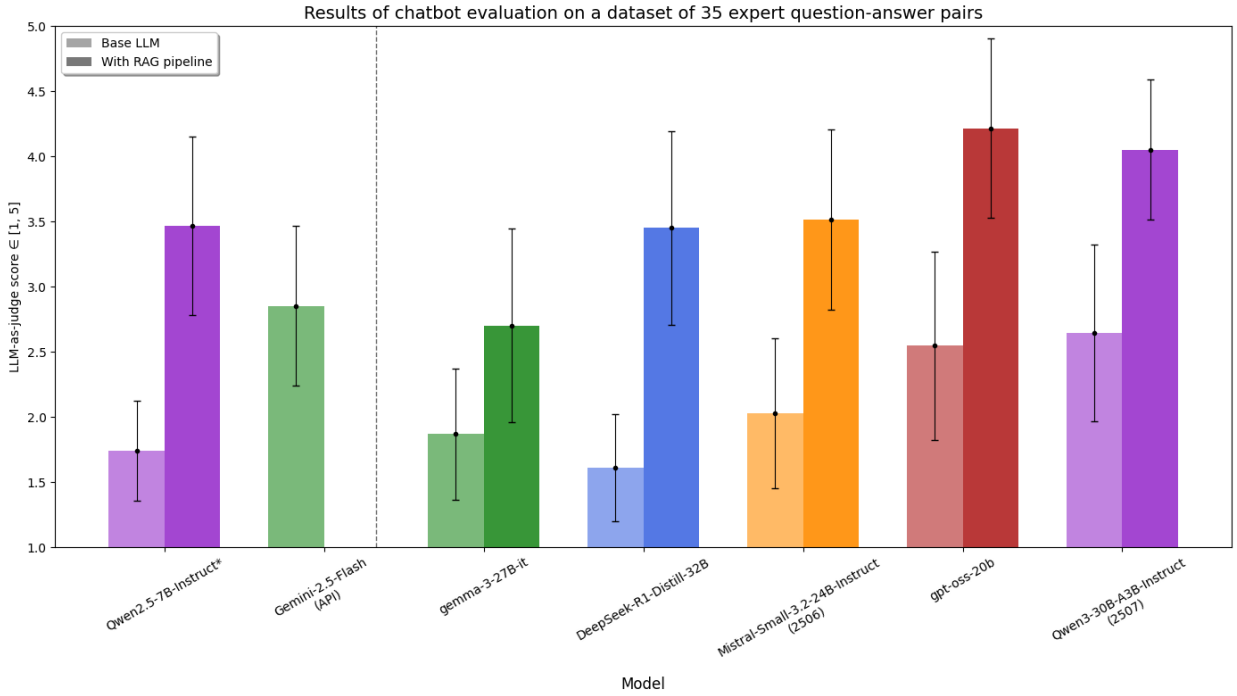

Figure 3: LLM-as-judge scores for different models with and without RAG pipeline.

On Figure 3 we see that by default, Gemini has the most knowledge about O^2 framework, which is expected. Performance of 5 out of 6 other LLMs improved significantly when using the RAG pipeline. Even the 7B Qwen2.5 achieved results comparable to or better than Gemini. We noticed that response time when using RAG actually decreases, likely due to fewer hallucinations and more precise answers. These trends were consistent across multiple metrics presented in Table 3. Overall, Gpt and Qwen3 were the best performing models and considering efficiency we selected Qwen3 as the default chatbot model.

With this LLM, we upscaled embeddings to `BAAI/bge-m3`. This slightly improved the metrics: LLM-as-judge score by 0.1 and semantic similarity by 1% while increasing the latency by 1 second. We also experimented with vector database text chunk sizes (500 to 2000 characters) and splitting strategies (recursive, semantic), but observed no positive effects.

LLM-as-judge score

Model	Base LLM	With RAG
Qwen2.5	1.74 ± 0.38	3.47 ± 0.69
Gemini	2.85 ± 0.61	N/A
Gemma	1.87 ± 0.50	2.70 ± 0.74
DeepSeek	1.61 ± 0.41	3.45 ± 0.74
Mistral	2.03 ± 0.57	3.51 ± 0.69
Gpt	2.55 ± 0.72	4.22 ± 0.69
Qwen3	**2.65 ± 0.68**	**4.05 ± 0.54**

Embedding similarity

Model	Base LLM	With RAG
Qwen2.5	0.71 ± 0.02	0.80 ± 0.03
Gemini	0.73 ± 0.02	N/A
Gemma	0.72 ± 0.02	0.68 ± 0.07
DeepSeek	0.72 ± 0.02	0.79 ± 0.05
Mistral	0.73 ± 0.02	0.76 ± 0.05
Gpt	0.73 ± 0.02	0.79 ± 0.05
Qwen3	**0.73 ± 0.01**	**0.81 ± 0.02**

ROUGE-L score

Model	Base LLM	With RAG
Qwen2.5	0.07 ± 0.01	0.18 ± 0.03
Gemini	0.04 ± 0.01	N/A
Gemma	0.04 ± 0.00	0.10 ± 0.04
DeepSeek	0.07 ± 0.01	0.17 ± 0.05
Mistral	0.08 ± 0.02	0.15 ± 0.04
Gpt	0.05 ± 0.01	0.14 ± 0.03
Qwen3	**0.06 ± 0.01**	**0.15 ± 0.03**

BLEU score

Model	Base LLM	With RAG
Qwen2.5	0.01 ± 0.00	0.07 ± 0.03
Gemini	0.00 ± 0.00	N/A
Gemma	0.00 ± 0.00	0.04 ± 0.02
DeepSeek	0.00 ± 0.00	0.06 ± 0.04
Mistral	0.01 ± 0.01	0.07 ± 0.03
Gpt	0.01 ± 0.00	0.05 ± 0.02
Qwen3	**0.01 ± 0.00**	**0.06 ± 0.03**

Response time (seconds)

Model	Base LLM	With RAG
Qwen2.5	8.45 ± 2.06	4.88 ± 0.88
Gemini	18.34 ± 1.48	N/A
Gemma	88.80 ± 8.08	25.35 ± 4.97
DeepSeek	54.20 ± 14.74	54.09 ± 12.53
Mistral	18.44 ± 6.43	15.93 ± 6.09
Gpt	43.64 ± 10.22	24.83 ± 5.59
Qwen3	**20.68 ± 3.84**	**16.56 ± 2.07**

Table 3: Results of the final chatbot model evaluation. We measured four answer correctness metrics and response times.

Another key RAG component we evaluated is the database configuration. In particular, the **retrieval parameters**, such as the number of documents to rerank `top_k` and retrieve `top_n`. The retrieval similarity threshold can be set arbitrarily, as it is only a first-stage filter. We observed how increasing the information retrieval affects both answer correctness and response time. We noticed that latency scales linearly while performance reaches a plateau at a certain number of documents (Figure 4). In our case, 15 documents was a reasonable cut-off point.

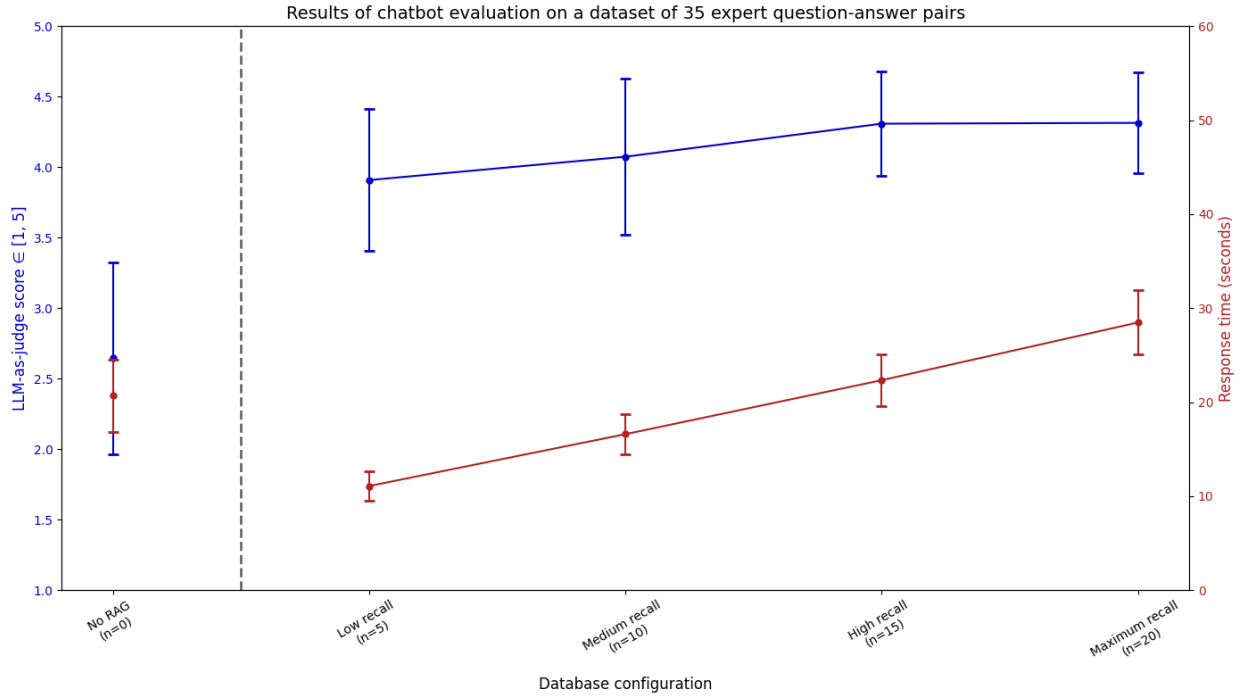

Figure 4: LLM-as-judge scores and response times for different vector database configurations.

10 Working examples

In the previous chapter, we justified the default chatbot model and database configuration, but we still left the option for users to **customize the chatbot**. This can be done by providing flags like `#param:value` at the start of a Mattermost message and changes are thread persistent. Currently, only two key components are configurable, as described in the Table 4.

Parameter	Description	Values	Default
model	Which LLM to use for the chatbot response.	gpt, qwen, mistral, gemma, deepseek	Qwen
db	How many documents to retrieve from the vector database (recall).	low, med, high, max	Medium

Table 4: Configurable chatbot parameters, their descriptions and values.

In Figures 5 and 6, we show a **typical interaction** with the AskALICE chatbot via Mattermost. A user first provides optional flags and asks a question. The query is then sent to the RAG pipeline, while the user observes the chatbot's state through placeholder messages: "Analyzing question", "Generating search query", "Searching for documents", and "Generating response". The first part of the response contains the actual generated answer. Under `Sources` we always append the links of all retrieved documents, enabling the user to verify the information. The response footer always states the used LLM, database

configuration, and asks the user to leave feedback (if the answer was helpful or not). The message history is stored per thread, so to ask a subquestion, we have to use the reply option in Mattermost.

In the first example we used the maximum recall database to get a list of some natively included generators in AliGenO2. The second example used Gpt to find a class and function that can return us MC tracks given an MC label. For questions about running anchored Monte Carlo simulations, we use a separate experimental workflow. It is focused on providing a predefined script template, explaining and setting environment variables.

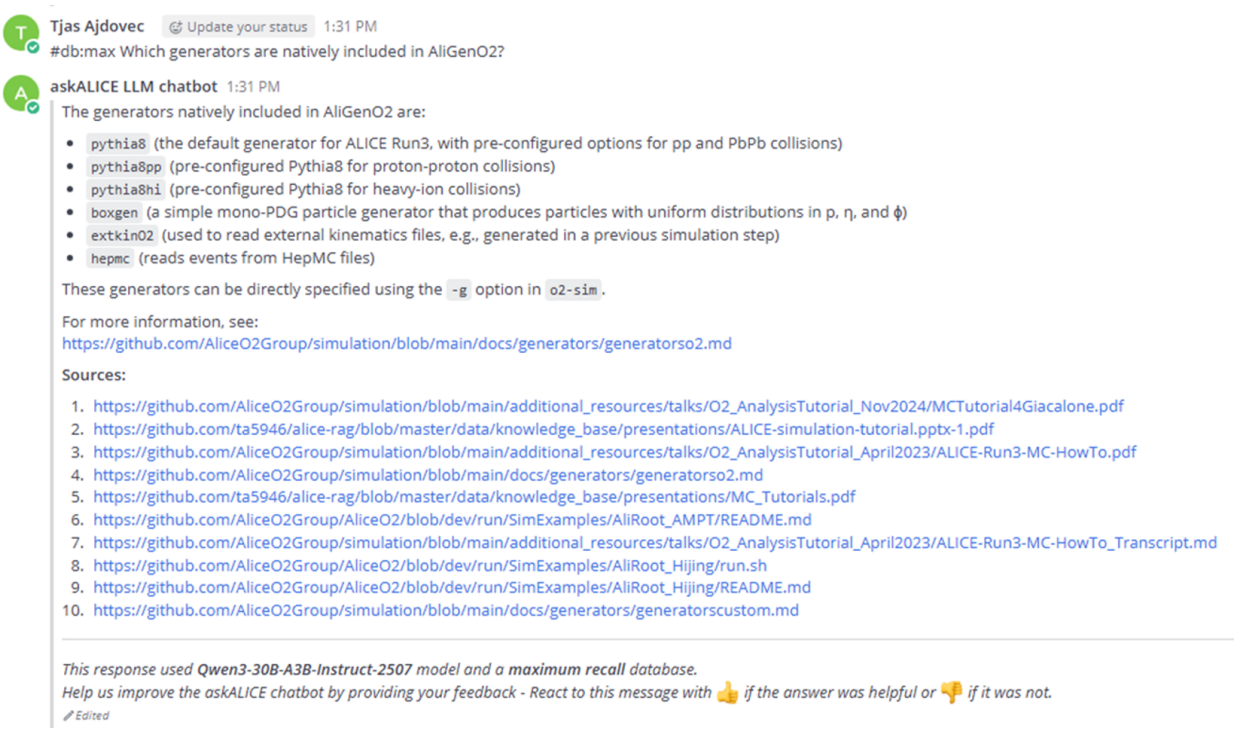

Figure 5: AskALICE chatbot answer to a user question about generators.

11 Call tracing and observability

RAG consists of multiple stages and observability is crucial for both **error and performance analysis**. We configured a local LangFuse server with the user interface hosted at `http://pc-alice-ph01:3000/`. There we can create a new user account and project, where each chatbot call is stored. Figure 7 shows an example trace where the chatbot was asked about the ALICE GRID. We can inspect the input and output at each pipeline stage. We can also see the corresponding RAG configuration, latency, date, API call cost (if applicable) and user feedback. This information can help us with further chatbot development. For example, we could use a dataset of responses with positive user feedback for direct preference optimization.

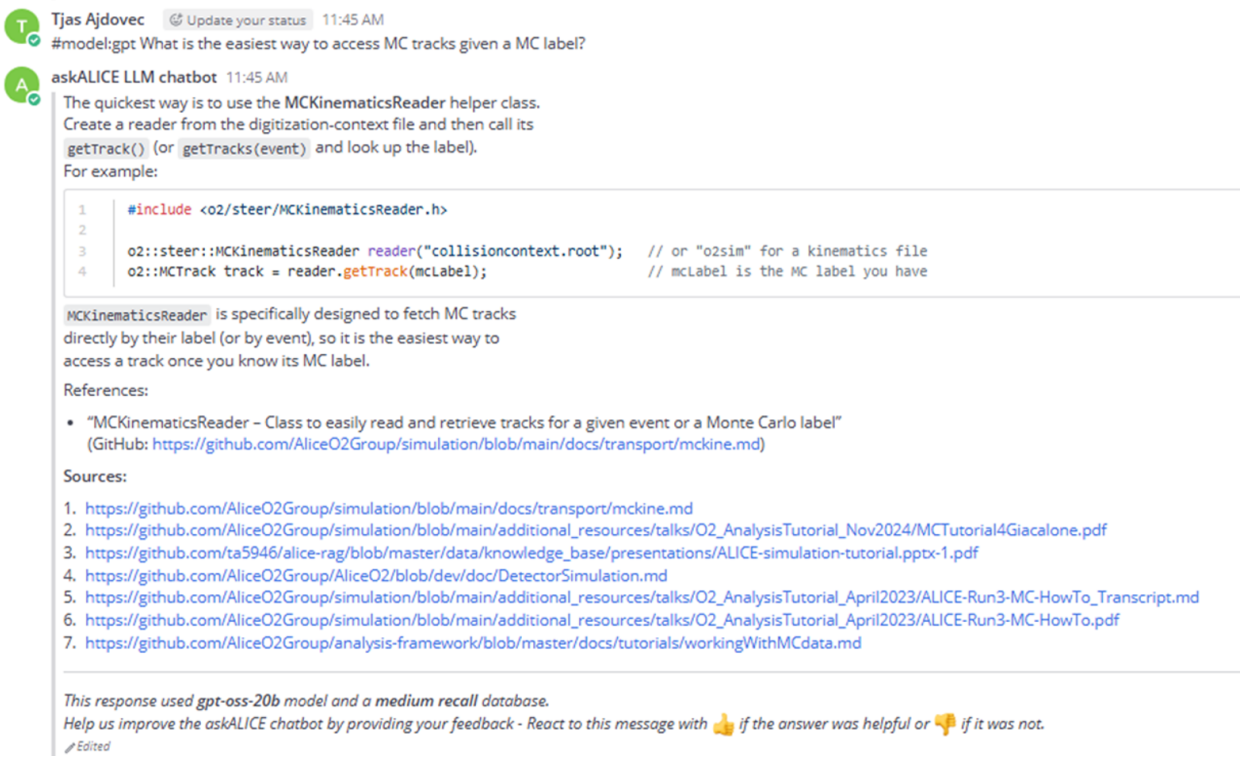

Figure 6: AskALICE chatbot answer to a user question about MC tracks.

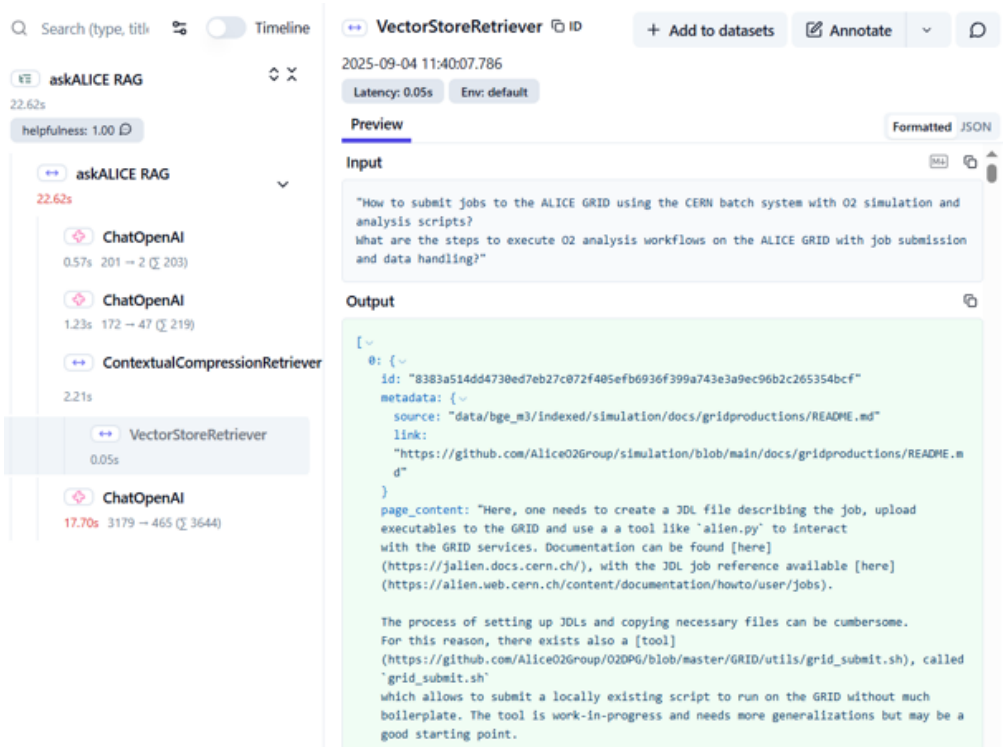

Figure 7: LangFuse trace for a user question about ALICE GRID.

12 Findings and future work

During the course of this pilot project, it turned out that even a 7B LLM can handle a simple RAG application and outperform state-of-the-art API models without it. Prompt engineering reduced hallucinations and latency as a consequence. Scaling up both the LLM and embedding models showed a moderate performance improvement, but not as big as when introducing the domain documents. All four selected answer correctness metrics showed relatively consistent results. Storing both the simulation and analysis data in a single vector database worked well, as the retrieval process was rigorous enough to handle that. RAG returns started diminishing at a certain number of documents to retrieve; this point should always be explored.

Interestingly, a good LLM judge is not necessarily a good conversational model, which was the case for Gemma. We should always evaluate multiple models for different parts of our application. We also experimented with generating artificial data, such as paraphrased documents and synthetic question-answer pairs. Enhancing our knowledge base with it did not improve performance, and a more sophisticated approach is required. The selected LLM tools and frameworks worked well. Besides using a different llama.cpp build, there were no issues switching from NVIDIA to AMD GPUs. All source code and documentation are available at: `https://github.com/ta5946/alice-rag`.

In the future we should encourage O^2 framework users to test out the chatbot and provide feedback. We should also let everyone contribute to writing more documentation, as our 400 document knowledge base is relatively small. Scaling up the models will not help if certain information is not present in the database. As mentioned before, we could enhance our documentation with synthetic data or knowledge graphs. With new documents we can continuously update the RAG database with GitHub actions or daily builds. We could extend the existing AskALICE chatbot to related use cases, such as:

- Analyzing simulation and other output files, even ROOT files.
- Monitoring selected Mattermost channels, for example the O2 simulation channel. The chatbot could join the conversation to answer basic questions and redirect users to other channels or platforms (Jira).
- Providing support for shifters / on-callers.

Alternatively, we could replace RAG with larger, more capable LLM agents. For each user query these agents can decide on which external sources (tools) to use, how to use them and even do multiple steps of reasoning before returning the final answer. This approach relieves the developer workload and produces more creative responses, but comes with a higher computational cost.

13 Acknowledgments

I would first like to thank my supervisors for leading me through the project and taking the time for meetings and discussions. Besides that, I am grateful for the other students who kept me company through the summer and my family, who always support my education.

References

[1] Niklas Muennighoff et al. "Mteb: Massive text embedding benchmark". In: *arXiv preprint arXiv:2210.07316* (2022).

[2] Yubo Wang et al. "Mmlu-pro: A more robust and challenging multi-task language understanding benchmark". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 95266–95290.

[3] Yunfan Gao et al. "Retrieval-augmented generation for large language models: A survey". In: *arXiv preprint arXiv:2312.10997* 2.1 (2023).

[4] Jianlv Chen et al. "Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation". In: *arXiv preprint arXiv:2402.03216* (2024).

[5] An Yang et al. "Qwen3 technical report". In: *arXiv preprint arXiv:2505.09388* (2025).

[6] Lianmin Zheng et al. "Judging llm-as-a-judge with mt-bench and chatbot arena". In: *Advances in neural information processing systems* 36 (2023), pp. 46595–46623.

[7] Anton Alkin et al. "ALICE Run 3 analysis framework". In: *EPJ Web of Conferences*. Vol. 251. EDP Sciences. 2021, p. 03063.

[8] Daniele Dal Santo et al. *chATLAS: An AI Assistant for the ATLAS Collaboration*. Tech. rep. ATL-COM-SOFT-2025-016, 2025.

A RAG prompts

Default message history:

```
This is the start of a new conversation. There is no message history yet.
```

Question classifier system message:

```
You are a question classifier.
Your task is to classify the following user message into one of the
   categories:
1. Question about running ALICE O2 simulations or analysis framework -
   requires retrieving context from the internal documentation.
2. Request for writing / running an anchored Monte Carlo simulation script
    - requires providing a script template and variable definitions.
Note that category 2 is a specific use case that should only be chosen if
   the user explicitly mentions "run anchored MC script template" or
   similar.
For example, "What is anchored MC?" should still be classified as 1.
Respond only with the category number (1 or 2) and nothing else.
```

Question classifier prompt template:

```
(
CONVERSATION HISTORY:
{conversation_history}
)

QUESTION:
{question}

CATEGORY:
```

Search query generator system message:

```
You are a search query generator.
You are provided with a question and conversation history.
Your task is to generate a couple sentences used to retrieve relevant
   documents about ALICE O2 simulations and analysis framework.
The retrieved documents will be used as context to answer the initial user
    question.
Focus on core keywords and topics while trying to avoid general terms such
    as "ALICE", "O2", "simulation", "analysis", "o2-sim", "documentation".
Respond only with the query sentences and nothing else.
```

Search query generator prompt template:

```
(
CONVERSATION HISTORY:
{conversation_history}
)

QUESTION:
{question}

SEARCH QUERY:
```

Response generator system message:

```
Your name is askALICE.
You are a chatbot designed to assist users with ALICE O2 simulation and
   analysis documentation.
Use the provided context to answer the following question:
- If the context contains relevant information, use it to provide a clear
   answer.
- If the context does not contain enough (or any) relevant information,
   say that you do not know the answer.
- Do not explain or mention the context, just use it to directly answer
   the question.
- Do not make up new information.
- You can reference the documents you used to formulate your answer and
   provide their links.
```

Response generator prompt template:

```
(
CONVERSATION HISTORY:
{conversation_history}
)

QUESTION:
{question}

CONTEXT:
{context}

ANSWER:
```