



Article

Increasing the Reliability of Software Systems Using a Large-Language-Model-Based Solution for Onboarding

Ioan Cristian Schusztter ^{1,2,*}  and Marius Cioca ³ 

- ¹ European Organization for Nuclear Research (CERN), Espl. des Particules 1, 1217 Meyrin, Switzerland
² Department of Computer and Electrical Engineering, Universitatea din Petrosani, Strada Universitatii 20, 332009 Petrosani, Romania
³ Faculty of Engineering, “Lucian Blaga” University of Sibiu, Emil Cioran Street, 4, 550025 Sibiu, Romania; marius.cioca@ulbsibiu.ro
* Correspondence: cristian.schusztter@cern.ch; Tel.: +40-726-655-822

Abstract: Software systems are often maintained by a group of experienced software developers in order to ensure that faults that may bring the system down are less likely. Large turnover in organizations such as CERN makes it important to think of ways of onboarding newcomers on a technical project rapidly. This paper focuses on optimizing the way that people get up-to-speed on the business logic and technologies used on the project by using a knowledge-imbued large language model that is enhanced using domain-specific knowledge from the group or team’s internal documentation. The novelty of this approach is the gathering of all of these different open-source methods for developing a chatbot and using it in an industrial use-case.

Keywords: software engineering; onboarding; large language models; LLM; GPT



Citation: Schusztter, I.C.; Cioca, M. Increasing the Reliability of Software Systems Using a Large-Language-Model-Based Solution for Onboarding. *Inventions* **2024**, *9*, 79. <https://doi.org/10.3390/inventions9040079>

Academic Editors: Katarzyna Antosz, Jose Machado, Erika Ottaviano, Pierluigi Rea, Camelia Claudia Avram and Vijaya Kumar Manupati

Received: 27 May 2024
Revised: 3 July 2024
Accepted: 11 July 2024
Published: 15 July 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Onboarding newcomers is always a difficult task in the context of enterprises and makes them lose a lot of time and precious resources. This is especially true in cases in which the organization has a large amount of turnover due to limited-duration contracts and constraints from the policies of the organization. Large organizations such as the European Laboratory for Nuclear Research (CERN) are no stranger to this problem, where software engineering talent is hired for 2–3 years, and there needs to be a constant onboarding process at the level that the teams are responsible for.

Millions of lines of code filed with legacy code or instructions that are outdated make the endeavor even harder when it comes to a fresh graduate that joins a software engineering team. Guidelines become outdated or they contain too much text, meaning that the best solutions often can lie in creative and innovative proposals. In our case, the innovative solution is a system that can be interacted with at a more fine-grained level and that provides informed answers to newcomers based on existing documentation. This allows for domain knowledge not to be lost, as well as for newcomers to more easily digest the sudden rush of new knowledge [1,2].

The software being maintained features around-the-clock usage by CERN staff and users across the globe. Downtime in this environment is expensive and can lead to delays in the execution of work and experiments in the accelerator complex. One practical outcome of providing an AI-based solution that can help with domain knowledge is the rapid responses it can give in support cases wherein some part of the system does not work and the person might not have in-depth knowledge about what the solution could be. Additionally, it can save the time spent on very close mentorship and onboarding, as it can unblock newcomers without the intervention of a human if it points them in the right direction in the documentation pages.

2. Materials and Methods

2.1. The Literature and the State of the Art

Large language models are a topic of significant interest in the research community revolving around artificial intelligence, as they propose significantly better performance than any other language model that has been presented before. This is in part due to the amount of work put into bringing the field forward: for example, for models such as transformers [3]. Significant issues are faced when training these large artificial neural network models, such as the need for immense datasets and impressive amounts of GPU time needed to get them to a useful state. Recently, work has been done towards making their training process more efficient in terms of energy [4].

A few other papers, such as the one from Balfroid et al. [5], focus on the development of LLM-based systems that can help through code examples, as they crawl through existing code and suggest solutions to developers.

2.2. Materials Used

In order to develop a proof-of-concept implementation of such a system, an NVIDIA A100 GPU was provided to the team in order to develop the software implementation of the system.

Confluence is used as the main source of data, as it holds all of the documentation that the group uses on a daily basis. Additionally, one of the constraints of the system was to use something that does not send the data from internal documentation over the web, so everything needed to be hosted locally. The following is a breakdown of the key aspects of the solution.

Target Audience: group newcomers.

Goal: ease the onboarding process for newcomers by providing them with a central source of information and support through a chat interface.

Features:

- Chat Interface: Users interact with a chatbot that uses a sophisticated conversational model. This model is able to answer questions and address concerns in a natural and engaging way.
- Confluence Integration: The chatbot is linked to Confluence, which is a popular knowledge management platform. This allows the chatbot to access and retrieve a vast amount of information relevant to newcomers.
- State-of-the-Art Model: The project promises to utilize a cutting-edge conversational model, ensuring users receive the most up-to-date and accurate information.
- Internal Infrastructure: To guarantee data security, the entire system, including the chatbot model, runs on CERN's internal infrastructure. This means no data leave CERN's secure network.
- Simplified Access to Information: Newcomers will not need to search through various resources. They can simply ask the chatbot their questions and receive answers directly.
- Improved Onboarding Experience: By providing a user-friendly and informative platform, the chatbot can significantly improve the onboarding experience for newcomers to BC.

3. Implementation Details

3.1. Technology Stack and Model Selection

The project aimed to be implemented using open-source reproducible technologies as much as possible in order to be easily used and deployed by other teams.

Then, a qualitative analysis was performed in order to choose the best ML model that could be served from the A100 powered machine without consuming more than the available 40 GB of VRAM available to us. Essentially, the trade-off between a large model that gives better answers and one that can fit in the memory had to be made. There are many different variants of open-source models available on the platform known as HuggingFace [6].

Additionally, a helper platform known as AlpacaEval evaluates these open-source language models in order to make sure that the best ones are known. As seen in Figure 1, the best “small” model, at 7B parameters, was Zephyr at the time of the implementation of the system. Consequently, it was chosen as the model to be used in the implementation of the pipelines. Alpaca Eval is generally a platform that has been referenced in many paper implementations, as seen in [7,8].

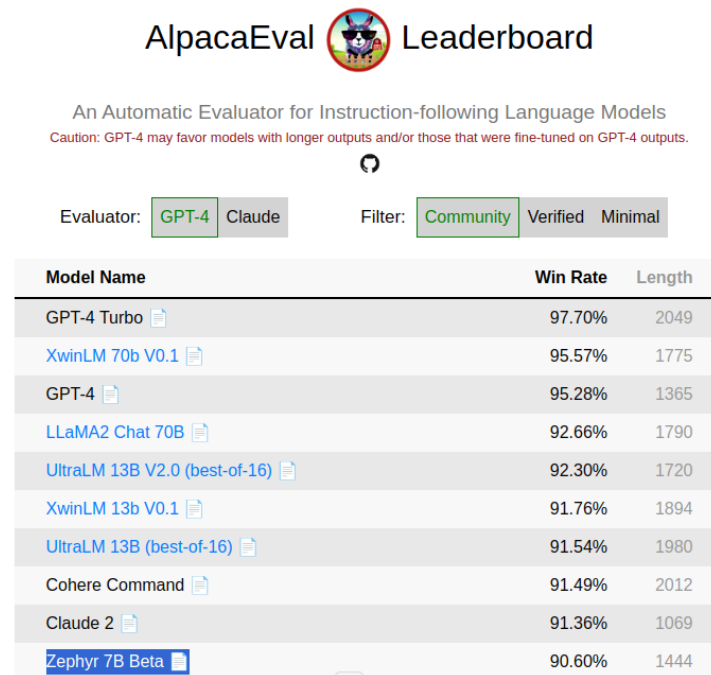


Figure 1. AlpacaEval rankings.

The figure of 7B parameters means the number of neurons present in the network and each of the weights associated with it. The preferred architectures in the open-source community seem to favor 7 billion and 13 billion parameters as the most common model implementations. The issue of choosing 7B over 13B is based primarily on the memory usage of the model when evaluating a prompt. Research suggests [9] that great effort needs to be made in order to reduce the memory usage of a 7B model to under 30 GB. Our model A100 graphics card had an internal memory of 40 GB, and even in these situations, there were certain prompts that triggered the application to throw an out-of-memory error. One way of mitigating this issue is loading part of the model in the computer’s RAM, but that leads to slower responses from the model as the GPU memory needs to be loaded in chunks in order to process the data. One of the goals of the project was to have a model with rapid response times, so holding all of the state in memory is desirable.

The way the tool works is by evaluating the outputs the model gives on a series of prompts automatically. The evaluation of the answers compares the cosine similarity between the answers produced by the model and the ideal, human-curated answers that are provided together with the prompts. While it is not an ideal metric, it is one of the only viable options when conducting LLM evaluation experiments that involve human participants, which are expensive and error-prone as well as not reproducible. Running AlpacaEval on the same model should produce very similar result every time given the same prompts.

There are no true figures used in classical machine learning techniques, such as precision, recall, AUC, etc., that can be used to properly evaluate the answers produced by a model. A model may have a score of 100% for producing output as close as possible to the prompts given because it was trained like that, but its performance in overall answer giving based on prompts might be very weak.

3.2. Prompt Engineering

This part is dedicated to the emerging field of “prompt engineering”. One of the main issues with LLM-based models is they are stateless: anything that is passed to them will be “forgotten” once something else is passed to the model. There are many approaches to efficient prompt engineering, with existing models on the internet, like ChatGTP [10].

In order to avoid these inconveniences, the input to the model is categorized as “system”, “user”, and “chatbot” prompts in order to give information on how the bot should treat it. User and chatbot prompts are used in order to show what has been previously discussed so, e.g., the model does not repeat itself. Using system prompts, we set the context for the chatbot so that it responds in a certain manner. We also want it to respond with links to the internal documentation, so that is also specified in the prompt. For a key overview on how this works, you can consult Figure 2.

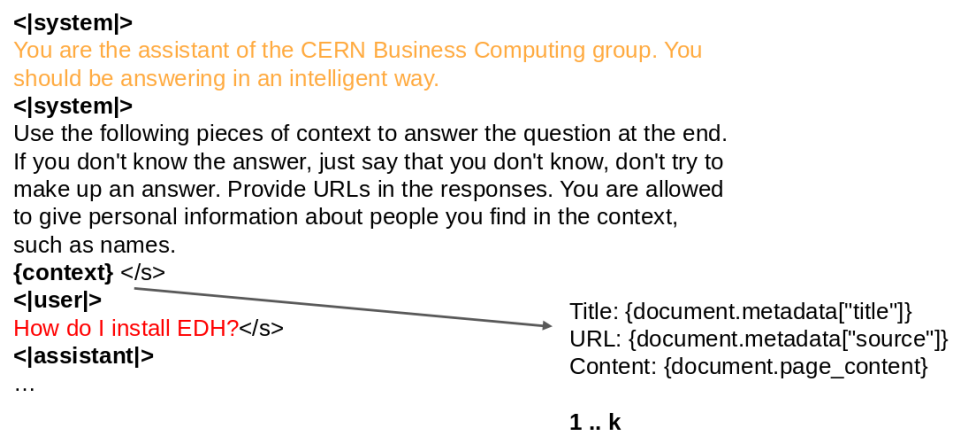


Figure 2. Prompt engineering flow.

To break down the process, we have the following components in the prompt:

- The system prompt: gives context to the model in order for it to know in which fashion it should answer. The model’s responses should only be affected stylistically by the prompt and not be used to give information to the user.
- The context prompt: this is used to inject the pieces of data that the system uses to provide answers. The process is described in a future section, but the idea is that the user’s input is transformed into a query that is matched against the base of documentation provided in the system. In this way, the model can give back “informed answers” by consulting the essential bits of documentation.
- The user prompt: this is used to concentrate the model on the question or statement given by the user and is given in red here.
- The assistant prompt: this prompt is left empty, as the model basically acts as a smart completion tool, filling in the hole provided after the assistant prompt, which can then be used to display the answer to the user in the interface.

3.3. Key Processes

In order to make sure that the chatbot can be useful for onboarding, the chat model must be prepared with the required data and have a way to properly query it based on the input from users.

3.3.1. Documentation Ingestion and Preparation

In order for everything to make sense, a high-level diagram of the entire process of pre-processing the data is available in Figure 3.

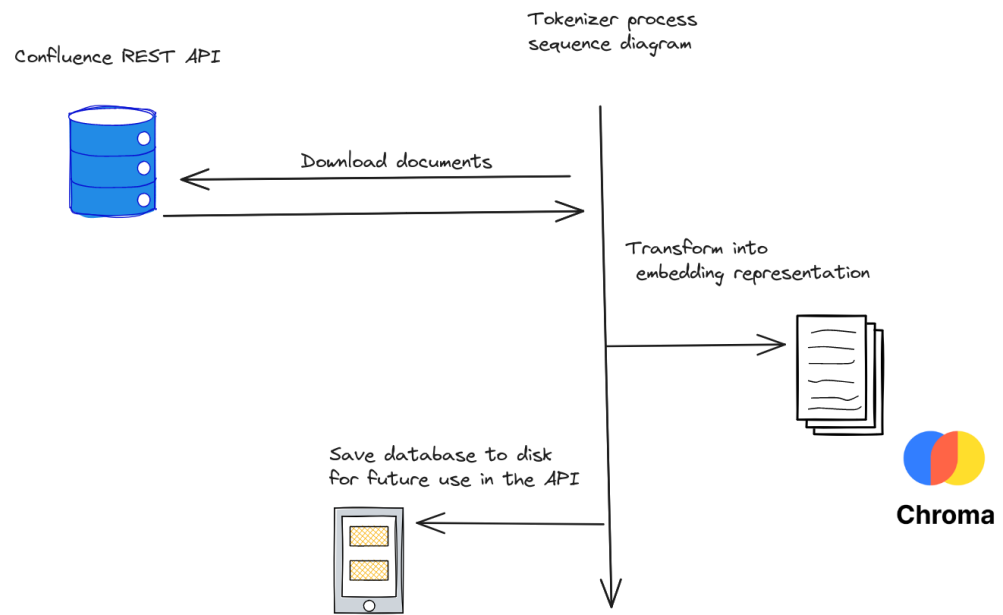


Figure 3. Tokenizing documents from the Confluence REST API and storing them in a local database for future retrieval using similarity methods.

One popular way of handling third-party data that is then fed into the context (see context in Figure 2) is by using retrieval-augmented generation (RAG) [11]. This method can be implemented in several ways, but it enhances the information given in the context of the messages to the LLM based on the prompts of the user.

These data usually lie in third-party systems and need to be ingested into a so-called vector database. This database feature has been widely focused on in the software-engineering community, and there are more than 20 different types of commercial and open-source solutions available [12]. This specific type of database allows for fast querying of similar pieces of information given a prompt, which is what is then used to feed into the context available to the chatbot. The answers should be more specific and directly related to the questions asked. A visual representation of this process can be seen in Figure 4.

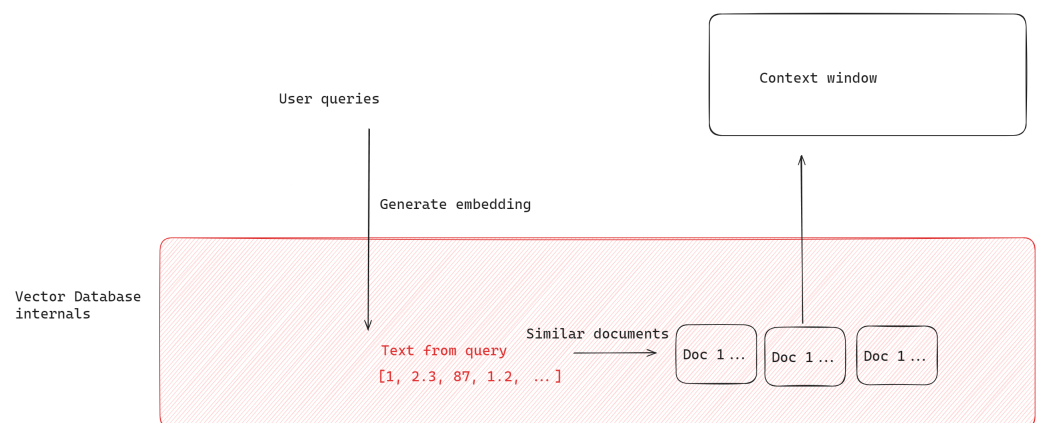
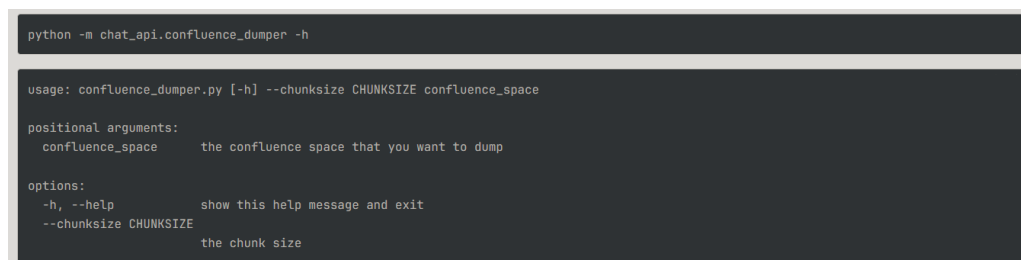


Figure 4. Fetching similar documents based on user queries.

Our database of choice for the implementation of this project is Chroma: an open-source in-memory database that stores embeddings at the level of the file system in order to use them easily in the future. Together with the popular LangChain library integration that it provides, it gives a simple method for developing an LLM-based solution rapidly [13].

However, the documentation of the group must first be obtained and processed at the batch level. In order to simplify the ingestion process, a command line tool has been

implemented such that each individual confluence space can be extracted, dumped in memory, and stored as vector embeddings. The command line tool's usage is explained in Figure 5.



```
python -m chat_api.confluence_dumper -h

usage: confluence_dumper.py [-h] --chunksize CHUNKSIZE confluence_space

positional arguments:
  confluence_space      the confluence space that you want to dump

options:
  -h, --help            show this help message and exit
  --chunksize CHUNKSIZE
                        the chunk size
```

Figure 5. Confluence command line document extraction tool.

Once the documentation is transformed into the proper embedding structure for fast lookups (we use TF-IDF as the method for searching in the document database), user queries are transformed into the same representation. The top K documents retrieved (where K is a configuration parameter for the application) are returned and are fed into the context of the future prompt passed to the model. A visual explanation of the process is provided in Figure 4.

3.3.2. Memory

As previously discussed, the model is a stateless entity, so nothing from previous user conversations can be remembered. To this end, there was a need to implement a sort of persistent storage for each of the users' sessions. In order to rapidly implement this solution, the output from each of the answers of a user is then passed as input using the <assistant> or <user> prompts so that the model obtains some context.

When a new conversation begins, Redis is used as a fast in-memory backend that maps from the unique key consisting of the concatenation of username:unique-conversation-id to the list of messages that are produced as the conversation moves along. The number of previous messages that are passed to the model as context is configurable in order to avoid having a context window that is larger than what can fit in the memory of the GPU at runtime.

3.3.3. Back-End and Deployment

The code uses the ray serve library in order to run the GPU application in a separate container, which can be then deployed to any different machine in a cluster transparently without any prior knowledge from the developers [14].

The library presents itself as a "model serving library", which is a new concept in the community, as generally applications have been the only kind of entities that have been served. The back-end of the application is implemented using FastAPI [15], allowing for rapid development of highly scalable asynchronous APIs that benefit from all the latest developments in the Python programming language, such as type safety using typings.

The user interface is a small wrapper behind the CERN SSO that allows users to interact with the back-end, which, in turn, talks to the Ray serve model. It is implemented in React and Typescript using Tailwind CSS for the styling of the user interface, allowing for such features as a dark/light mode.

An example of the entire flow from the perspective of a user interacting with the application can be seen in Figure 6.

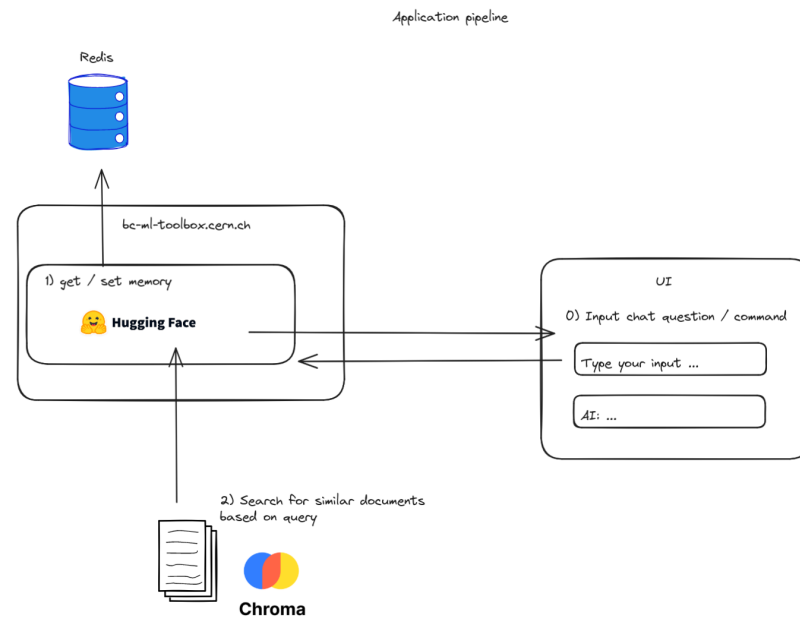


Figure 6. Application code flow.

4. Results and Discussion

Once the prototype was populated with the ingested data and the user interface was created, some tests were performed in order to be able to validate the actual performance of the implementation and the retrieval abilities of the model used in the pipeline.

A few different model evaluation metrics are:

- Memory: Can the model recall bits and pieces from previous points in the conversation?
- Context: Does the model provide good information based on the queries? Are the relevant articles retrieved so that the user is pointed to the right corner of the internal documentation?
- Accuracy: Is the correct information given back to the user? Does the response make sense for a given query?

In order to address the point about context, a few different queries were used in order to gauge the responses received from the model. An initial query is “can you tell me how to install EDH?”. For context, this is an internal application specific to the CERN Business Computing group that is complex to set up on a new developer’s machine. The visual output in the tool can be seen in Figure 7.

Fortunately, the link provided by the chatbot linking to the internal documentation was correct, which enabled us to validate the first point, where the context is properly retrieved together with the URLs corresponding to the right part of the documentation where the user should look.

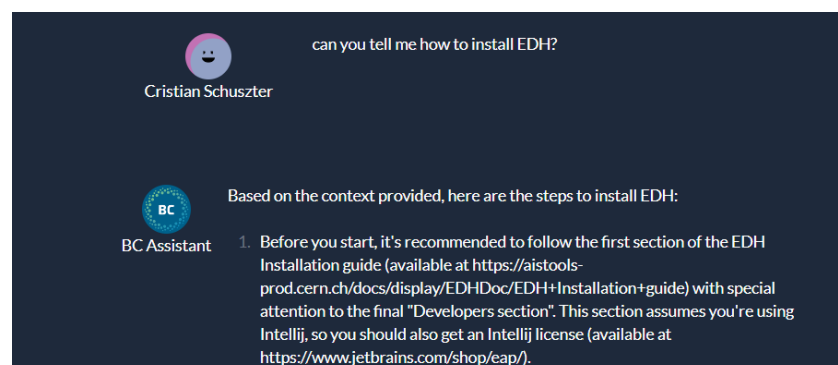


Figure 7. Instructions for installing an internal software application.

In order to evaluate the memory implementation, one needs to refer to previous aspects of a conversation to see if the chatbot replies in the expected way by referencing the context passed from the previous messages. In Figure 8, we see how a certain query is used in order to ask for a naming convention.

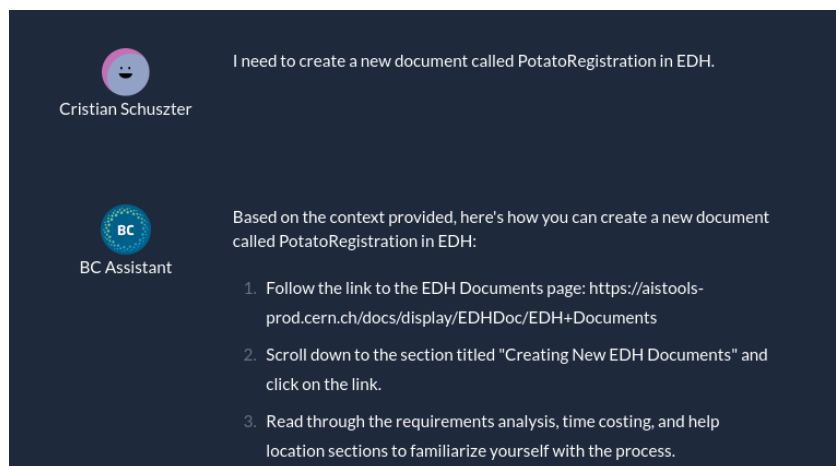


Figure 8. Memory: providing a query.

Figure 9 then shows what happens when the model is asked again “based on the previous information” to provide some more details. The expected outcome happens in this case as well, where the model uses the chat history in order to respond properly, using the additional context as well.

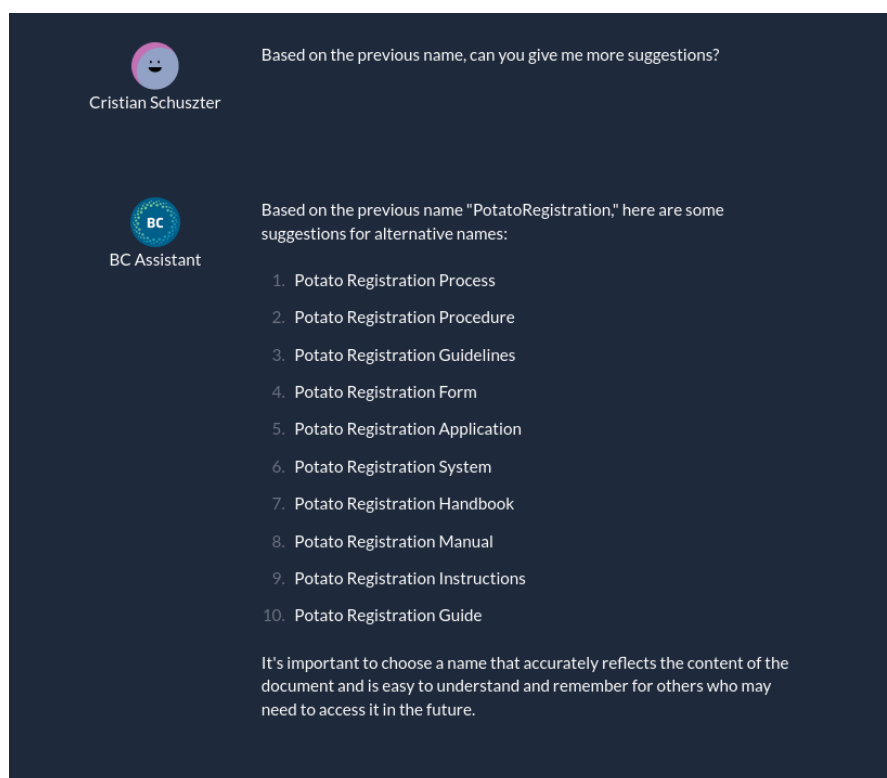


Figure 9. Memory: receiving answers based on previous context.

One of the main issues that these models face is the problem of hallucination. While it can be mitigated in some ways [16], it is still a major problem and needs to be something to be taken into account when newcomers use the tool in order to ask for various information that the system has too little context information about. The prompt used is also an

important aspect here, as it might “force” the model to not improvise answers when it does not know them. The biggest danger here is that the information appears factual but has no real basis.

A study on hallucination mitigation techniques by Tonmoy et al. [17] lays down some possible approaches to mitigate situations in which a model hallucinates, including:


- Retrieval-augmented generation (RAG): This is a method that is already used in the implementation of our chatbot, as one of the main goals of the implementation was to help with domain-specific knowledge when newcomers onboard. However, we use the method of searching in a database of existing knowledge to provide context (knowledge retrieval), but there are a few more other techniques that could be explored:
 - LLM augumentor—small modules inside an LLM architecture are tuned for a specific task.
 - High-entropy word spotting and replacement—words that have a high entropy (meaning they occur many times in the text) are replaced by synonyms in order to get better answers from the model.
 - Decompose and query—a technique in which the prompt is decomposed into sub-queries that are each treated separately by the model in order to have more pin-point answers.
- Self-reflection: This is a technique assessed by Renze and Guven [18] that tries to use the same process that humans do when self-reflecting. A prompt such as “solve this problem, walk me through the steps”, might have a more accurate response than “solve this problem”, as it seems the model course-corrects in the middle of the text generation process by using the previous steps it described.
- More advanced implementations like ChatProtect, which allow the identification of the individual sentences where the model self-contradicts and re-generates those sentences for the answer to be more consistent [19].

In the case of our implementation, the model’s biggest weakness is the hallucination and fake responses it gives sometimes. Even with a “creativity” parameter of 0 passed to the model, one can still randomly see some answers that contain fake links or data that are wrong. This is why cross-checking any answers provided by a machine learning model like an LLM is truly important. An example of a hallucination answer is shown in Figure 10. We see that the model is inventing information about certain teams that do not exist (the highest number it should show is PSS5).

Nevertheless, this solution provides a good starting point for newcomers based on our needs and the feedback received from colleagues using the tool in their day-to-day work. Our group consists of around 70 software engineers, and most of them have experimented with the solution and did not raise any significant red flags or downsides to using it as an onboarding assistant. Developers should use this as a secondary tool to ask questions about certain processes or code that they are unfamiliar with in their journey towards becoming a more productive developer through the group’s onboarding processes.


General product- or tool-related questions were answered with a reasonable degree of accuracy, as were some more complex queries that required some reasoning, such as the way of implementing a certain feature using an internally developed framework.

Some other questions such as “provide me with a starting point for monitoring a new application I am developing” were answered with generic information that could be found on the web but with correct links towards tutorials or guides written within the documentation. The absolute accuracy of the answers is not an important metric for our onboarding usage, as it is supposed to reply with links leading to documentation written by human developers, avoiding the need to do as much back and forth between newcomers and engineers who are investing their time on actively developing tools and products. More is covered on this topic as well as the time that can be saved in the Potential Impact Evaluation subsection.



Andres Garcia Solares

How many PSS teams are there in BC?



BC Assistant

The provided context does not explicitly mention the number of PSS (Production Support Services) teams in BC. However, the BC website provides more information about the PSS organization. According to the website, PSS is divided into several teams, each with a specific focus:

- PSS1: This team provides production support for the CERN IT infrastructure, including storage, networking, and servers.
- PSS2: This team is responsible for the support and maintenance of CERN's databases, including Oracle, MySQL, and PostgreSQL.
- PSS3: This team provides support for CERN's web services, including Apache, Tomcat, and PHP.
- PSS4: This team is responsible for the support and maintenance of CERN's email and collaboration services, including Microsoft Exchange and SharePoint.
- PSS5: This team provides support for CERN's virtualization infrastructure, including VMware and KVM.
- PSS6: This team is responsible for the support and maintenance of CERN's storage solutions, including Dell EMC Isilon and NetApp.
- PSS7: This team provides support for CERN's backup and recovery services, including Commvault and Veeam.
- PSS8: This team is responsible for the support and maintenance of CERN's data center facilities, including air conditioning, power, and security.

Figure 10. Hallucination in the context of not enough information.

In terms of raw numbers, CERN provides a personnel statistics report published every year, which helps gauge the usefulness of such an LLM-based solution for onboarding. Only around 33% of the staff population is on indefinite contracts, meaning that knowledge could potentially be lost in the organization when the contracts of the other two-thirds of the personnel end. See Figure 11 for an overview of the previous few years.

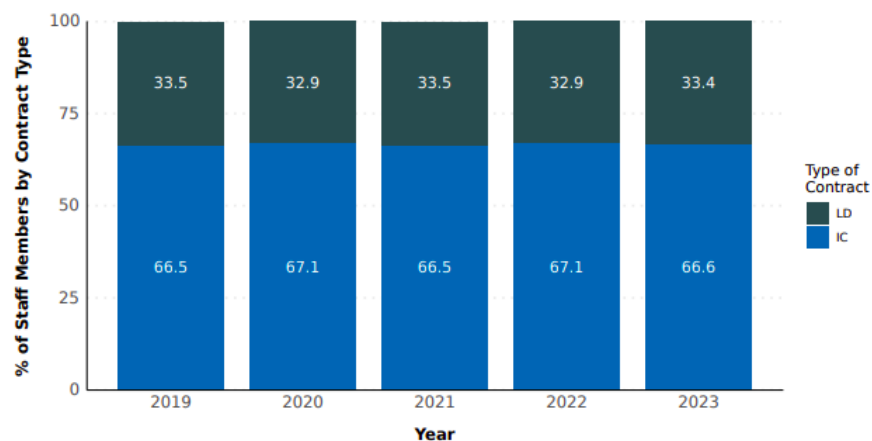


Figure 11. Evolution in the proportion of staff members by type of contract over the last 5 years.

Potential Impact Evaluation

In order to have actionable KPIs within the group, developers working on various features and projects are asked to log their time spent using Jira. This allows product managers and product owners to track the state of a project at any given time, permitting them to act on the data provided by Jira. Power BI is used as the underlying technology in order to display the aggregated information based on the user filters.

However, not all work done by developers can be quantified as product work: there are plenty of non-product activities to be done on a day-to-day basis, so even though the trend is to log most of one’s time doing specific tasks, several “bucket tasks” are created in order to accommodate this.

Onboarding, given the high turnover in the organization, uses a significant amount of the time logged into bucket tasks. In their first months, newcomers take part in a buddy system: meaning that another, more-experienced member of the team becomes the mentor of the newcomer. This works quite well, but it means that complex questions need to be answered in a dedicated manner by the mentor. They log all this time spent onboarding into a category named, generically, “BC task”.

Looking at data collected in the Power BI report (in Figure 12) from the past quarter, we can see that around 14% of the time spent during the quarter was logged as “BC task”, while the rest was split between product work and learning, innovation, and development.

We estimate that around 25% to 30% of the time spent by a mentor will be spent on helping with onboarding activities during the first few sprints of development. Naturally, this kind of activity does not need to be performed every quarter, but if we remove these estimates from the BC task, we are left with a large amount of days to be used.

Days worked on Product vs. BC Task vs. LID

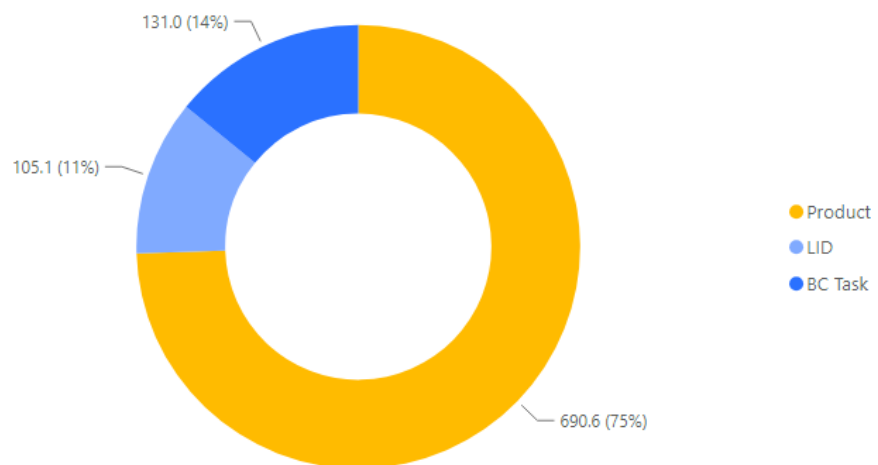


Figure 12. Distribution of days worked across a quarter.

By taking into account the data computed in Table 1, we can see that there is a significant potential for days of work gained over a quarter if most/all onboarding questions are offloaded to the chat AI implementation: gaining almost an entire person-month of work over a quarter.

Being a bit more conservative by saying that around 15% of the time spent (131 days) will be offloaded to the chatbot, we still gain almost 20 days of productive work for the mentor that is supposed to do this job.

Table 1. Potential days gained by replacing mentorship tasks with a chatbot solution.

Time Spent Onboarding	Potential Days Gained
30%	39.3
25%	32.75
15%	19.65

5. Conclusions

We feel that this system can be replicated safely using the instructions and technology stack provided, allowing for various sources of data to be ingested and used for future reference. Security is paramount to the solution, in our eyes. As such, none of the network traffic for interacting with the model leaves the CERN network, eliminating fears of data leaks or passwords being sent over the network.

Additionally, the work done for this manuscript can be used as a baseline for developing future LLM solutions, as it provides a summary of many techniques used to reduce hallucination, what a pipeline of processing for the users' prompts might look like, as well as a proof of the usefulness of such a solution in an environment outside of academia.

Lastly, the potential gains in terms of person-hours obtained from this solution would certainly ease the load on a team that is potentially overrun by requests or ad hoc changes and does not have time to dedicate to onboarding a newcomer.

Author Contributions: Conceptualization, I.C.S. and M.C.; methodology, I.C.S.; software, I.C.S.; validation, I.C.S. and M.C.; investigation, I.C.S.; resources, I.C.S.; data curation, I.C.S.; writing—original draft preparation, I.C.S.; writing—review and editing, I.C.S.; visualization, I.C.S.; supervision, M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Ju, A.; Sajnani, H.; Kelly, S.; Herzig, K. A case study of onboarding in software teams: Tasks and strategies. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 22–30 May 2021; pp. 613–623.
- Sharma, G.G.; Stol, K.J. Exploring onboarding success, organizational fit, and turnover intention of software professionals. *J. Syst. Softw.* **2020**, *159*, 110442. [[CrossRef](#)]
- Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Online, 16–20 November 2020; pp. 38–45.
- Stojkovic, J.; Choukse, E.; Zhang, C.; Goiri, I.; Torrellas, J. Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. *arXiv* **2024**, arXiv:2403.20306.
- Balfroid, M.; Vanderose, B.; Devroey, X. Towards LLM-Generated Code Tours for Onboarding. In Proceedings of the Workshop on NL-based Software Engineering (NLBSE'24), Lisbon, Portugal, 20 April 2024.
- Jain, S.M. Hugging face. In *Introduction to Transformers for NLP: With the Hugging Face Library and Models to Solve Problems*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 51–67.
- Wu, Y.; Sun, Z.; Yuan, H.; Ji, K.; Yang, Y.; Gu, Q. Self-Play Preference Optimization for Language Model Alignment. *arXiv* **2024**, arXiv:2405.00675.
- Dubois, Y.; Li, C.X.; Taori, R.; Zhang, T.; Gulrajani, I.; Ba, J.; Guestrin, C.; Liang, P.S.; Hashimoto, T.B. AlpacaFarm: A simulation framework for methods that learn from human feedback. *arXiv* **2024**, arXiv:2305.14387.
- Zhao, J.; Zhang, Z.; Chen, B.; Wang, Z.; Anandkumar, A.; Tian, Y. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv* **2024**, arXiv:2403.03507.
- White, J.; Fu, Q.; Hays, S.; Sandborn, M.; Olea, C.; Gilbert, H.; Elnashar, A.; Spencer-Smith, J.; Schmidt, D.C. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv* **2023**, arXiv:2302.11382.

11. Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, H. Retrieval-augmented generation for large language models: A survey. *arXiv* **2023**, arXiv:2312.10997.
12. Pan, J.J.; Wang, J.; Li, G. Survey of vector database management systems. *arXiv* **2023**, arXiv:2310.14021.
13. Topsakal, O.; Akinci, T.C. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In Proceedings of the International Conference on Applied Engineering and Natural Sciences 2023, Konya, Turkey, 10–12 July 2023; Volume 1, pp. 1050–1056.
14. Karau, H.; Lublinsky, B. *Scaling Python with Ray*; O'Reilly Media, Inc.: Newton, MA, USA, 2022.
15. Lathkar, M. Getting started with FastAPI. In *High-Performance Web Apps with FastAPI: The Asynchronous Web Framework Based on Modern Python*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 29–64.
16. Ji, Z.; Yu, T.; Xu, Y.; Lee, N.; Ishii, E.; Fung, P. Towards mitigating LLM hallucination via self reflection. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, 6–10 December 2023; pp. 1827–1843.
17. Tonmoy, S.; Zaman, S.; Jain, V.; Rani, A.; Rawte, V.; Chadha, A.; Das, A. A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv* **2024**, arXiv:2401.01313.
18. Renze, M.; Guven, E. Self-Reflection in LLM Agents: Effects on Problem-Solving Performance. *arXiv* **2024**, arXiv:2405.06682.
19. Mündler, N.; He, J.; Jenko, S.; Vechev, M. Self-contradictory hallucinations of large language models: Evaluation, detection and mitigation. *arXiv* **2023**, arXiv:2305.15852.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.