[Skip to content](https://mlops.community/how-to-build-a-knowledge-assistant-at-scale/" \l "bodycontent)

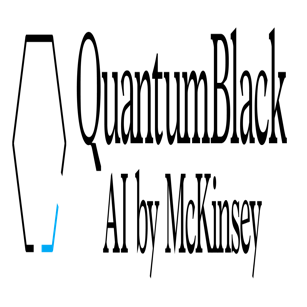
[MLOps Community](https://mlops.community/)

* [**Join**](https://mlops.community/join/)
* [**Learn**](https://learn.mlops.community/)
* [**Tools**](https://mlops.community/learn/)
* [**Blog**](https://mlops.community/blog/)
* Events
* [**Videos**](https://home.mlops.community/public/content)
* [**Partner**](https://mlops.community/partnerships/)

December 22, 2023

**How to Build a Knowledge Assistant at Scale**





[QuantumBlack Team Jannik Wiedenhaupt, Roman Drapeko, Mohamed Abusaid, Nayur Khan](https://mlops.community/author/quantumblack-team/)

QuantumBlack, AI by McKinsey unlocks the power of artificial intelligence to help organizations blend AI and cutting-edge solutions with strategic thinking and domain expertise.

[www.mckinsey.com/capabilities/quantumblack/](https://www.mckinsey.com/capabilities/quantumblack/)

**Introduction**

The discussion about the myriad applications of Large Language Models (LLMs) is extensive and well-trodden in tech circles1. These models have opened many use cases, reshaping sectors from customer service to content creation. However, an often-overlooked aspect in this discourse is the practicality of productizing and scaling these use cases to support tens of thousands of users2. The challenges range from expanding server capacity, tweaking algorithms, ensuring robustness and reliability, and maintaining privacy and security.

In this article, we describe some of the considerations necessary when developing an enterprise-level knowledge assistant (KA) and introduce a scalable architecture.

**Foundational Architecture Principles**

A well-designed KA must offer round-the-clock operation in a demanding enterprise environment and embody more than just cutting-edge AI capabilities. It should be fine-tuned for quality and speed and structured for continuous improvement, allowing for seamless integration and evolution of new functionalities. These operational imperatives set the stage for this proposed architectural design.

To achieve these high standards of operational excellence, the KA is built upon five foundational architecture principles. Each principle plays a critical role in ensuring that the KA meets the immediate needs of a large user base and remains a versatile and forward-looking solution, ready to adapt and grow with the changing landscape of enterprise requirements.

1. **Scalability**: Addressing the high volume of interactions and the expansion needs of AI applications.
2. **Security**: Ensuring data safety and strict access control in a world where information security is paramount.
3. **Transparency**: Offering clear insights into system operations, usage metrics, and cost implications.
4. **Modularity**: Facilitating easy upgrades and modifications to stay abreast of technological advancements.
5. **Reusability**: Promoting efficiency and collaboration by designing components that can be used across various projects.

These foundational architecture principles are intricately woven into every aspect of the suggested design, forming the backbone of a retrieval-augmented generation (RAG) architecture.

**Early-Stage Decisions Enhancing KA’s Foundational Principles**

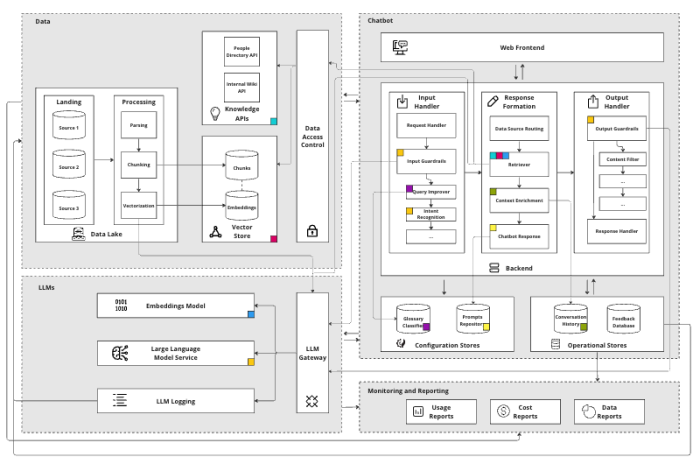
1. **Quality Over Cost**: We know quality matters. This foundational choice means accepting more significant upfront expenses linked to token usage and infrastructure. This decision is worthwhile as better performance and reliability from these quality investments bring tangible savings.
2. **Service-Based LLMs**: Another critical early decision adopts a service-based approach to LLMs. This choice underscores the need for flexibility and scalability in a KA’s language-processing capabilities. By integrating state-of-the-art service-based LLMs, any KA is equipped to rapidly adapt to changing conditions and technological advances, positioning it as a cutting-edge solution in this technology realm.
3. **LLM-Agnosticism:**As the space of generative AI develops, and new players and models enter the space regularly, it is essential that a KA is future-proofed by offering the option to switch the underlying LLM(s) easily.

These early-stage decisions shape the design of a KA into a robust, adaptable, and high-performing enterprise KA. As we explore the multi-layered architecture of such a KA in the following sections, we’ll see how these enhanced principles drive the design and functionality of each layer in the system.

**A RAG architecture**

At the core of the KA is a carefully crafted architecture segmented into four essential layers, each with its unique function and set of challenges. This multi-layered approach forms the KA’s structural and operational framework, grounded by the foundational principles described above.

1. **Data Layer**: The foundation, where vast amounts of data are processed and prepared for retrieval. It is crucial for the KA’s enterprise-specific intelligence.
2. **LLM Layer**: The general-purpose intelligence and processing center for all language model requests, ensuring contextually accurate and relevant responses.
3. **Reporting Layer**: The analytical segment, which provides usage, cost, and performance metrics insights.
4. **Application Layer**: The user-facing interface and backend with business logic – a key layer of the KA that navigates logic for forming responses to end-users.



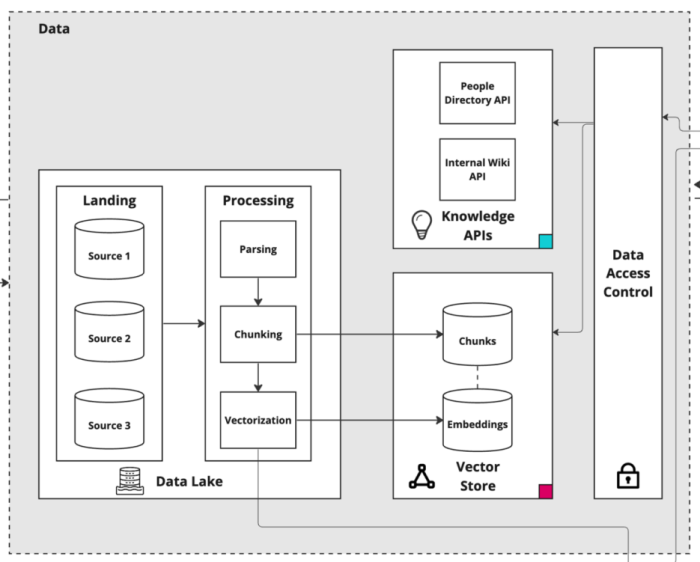
As we embark on a detailed journey through the layers, we will briefly examine the Data, LLM, and Reporting layers, highlighting their roles and significance. The spotlight will then shift to an in-depth look at the Application Layer, where the KA’s functionalities come to life, directly interacting with, and serving, the end-users.

**1. Data Layer**

The Data Layer of the KA is integral to its “enterprise-specific” intelligence, anchored by a **vector store** that stores documents in chunks, along with their embeddings and metadata4. This vector store is essential for facilitating search using semantic similarity on a large scale, ensuring performance remains robust even as data volumes expand.

LLMs have limits on how much data they can accept and process at one time (also known as token limits), making it hard to process long documents simultaneously. We recommend use of a well-known “chunking” technique to break documents into smaller parts to solve this. This enables search across the whole document in steps, avoiding an LLM’s token limit. Metadata stored alongside chunks enables us to associate information found during searches with source documents. Custom pipelines enrich the data with as much relevant metadata as possible to improve search results. This capability maintains context and relevance in the KA’s responses.

Selecting the suitable vector database and the appropriate chunking strategy is critical. Different chunking strategies, such as syntactic versus semantic, variable versus fixed, play distinct roles in how data is processed and retrieved5.



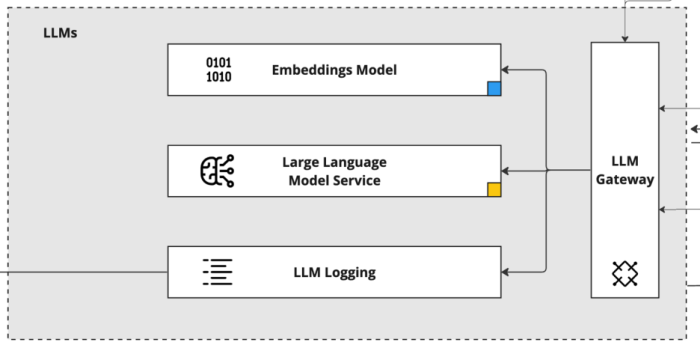
To handle the vast amounts of data, we recommend a **Data Lake with data processing pipelines,**implemented with a framework like the open-source Python-based Kedro6. These pipelines should be tasked with parsing, chunking, metadata enrichment, and vectorizing data chunks, subsequently populating the vector databases. These pipelines need well-structured and indexed data storage, so it is crucial to have healthy data quality and governance in place.

Additionally, the Data Layer can provide access to various **knowledge APIs**, like a People Directory and a Wiki, to further enrich the KA’s responses. These APIs may offer additional context and relevant information, enhancing the capability to deliver tailored and intelligent responses.

Finally, it’s important to control who can access what. If a user can’t see a document, the KA should not take it into response formation. The **data access control** component should be decoupled from the KA itself. This approach not only fortifies security and ensures compliance but also elegantly paves the way for seamless scalability across multiple KAs.

**2. LLM Layer**

The LLM Layer in the KA’s architecture serves as the central unit of processing. This layer is uniquely designed to handle the complexities and demands of processing language model requests, playing a critical role in the functionality of the KA.



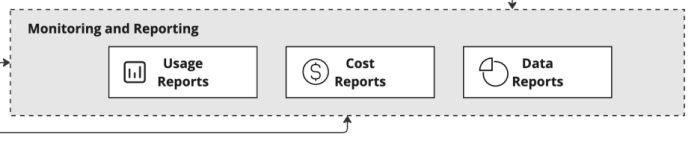
A key component of the LLM Layer is the **LLM API Gateway**. This gateway is the conduit through which all requests pass, acting as a centralized processing point. Its design includes scalable-on-demand integrations with multiple LLM vendors, offering the flexibility to easily switch services as needed. This versatility is crucial in maintaining operational efficiency and adapting to various requirements or changes in vendor capabilities.

An important function of the LLM API Gateway is its ability to track the costs associated with using LLMs (e.g. tokens generated, subscriptions). This feature is vital for managing the operational budget and optimizing resource allocation. Additionally, the gateway logs all interactions in a **logging platform**. This logging is not just about keeping a record; it’s a treasure trove of data that can be analyzed for improvements, troubleshooting, and understanding usage patterns.

Within this layer, there is direct access to both **LLM models** and **Embedding models**. The LLM models are the backbone of the KA’s language understanding and generation capabilities. Meanwhile, the Embeddings models, which are also used by the Data Layer for vectorizing document chunks, play a critical role in enhancing the semantic search capabilities of the KA.

**3. Reporting Layer**

The Reporting Layer in any KA’s architecture is essential for providing transparency on several critical fronts: costs, usage, and data analytics. This layer is intricately designed to capture and present a comprehensive view of the KA’s operational dynamics, making it an invaluable tool for both management and continuous improvement.



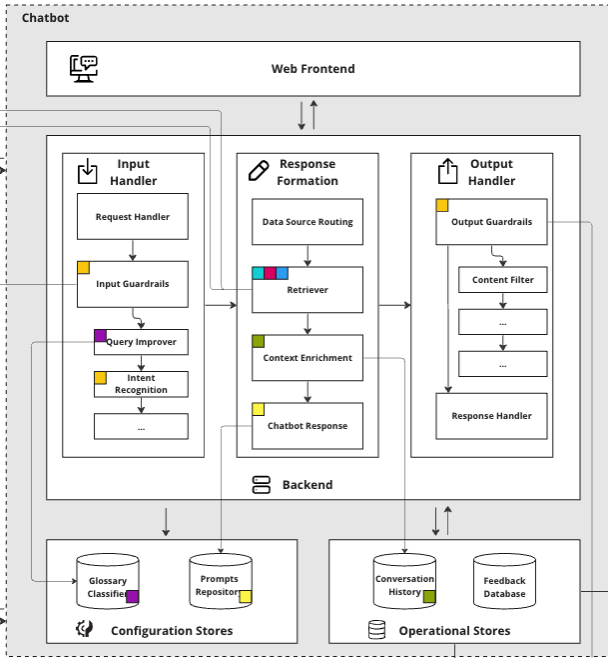
One of the primary functions of the Reporting Layer is **cost analysis** to track and analyze all expenses related to the operations of the KA. This includes costs associated with token consumption by LLMs, data processing, and other computational resources. By offering detailed insights into these expenditures, the Reporting Layer enables effective budget management and helps identify opportunities for cost optimization.

Another crucial aspect of this layer is **usage monitoring**. It keeps a close watch on how the KA is being used across the organization. This monitoring covers various metrics, such as the number of user interactions, peak usage times, and the types of queries being processed. Understanding these usage patterns is vital for scaling the KA effectively and ensuring it meets the evolving needs of the enterprise.

Additionally, the Reporting Layer delves into **data analytics**, providing an in-depth look at the performance and effectiveness of the KA. This includes analyzing response accuracy, user satisfaction, and the overall efficiency of the KA’s operations. Such analytics are instrumental in guiding future improvements, ensuring the KA remains a cutting-edge tool for the enterprise.

**4. Application Layer**

The Application Layer is where the functionality of the KA comes to the forefront, directly engaging with users. This layer is where user queries are generated, received, processed, and responded to, encompassing the end-to-end interaction that defines the user experience.



The Application Layer comprises of four main components:

* **Frontend**: This is the user interface of the KA, where users interact and input their queries.
* **Operational Stores**: These are databases that store the KA’s conversational history and user feedback.
* **Configuration Stores**: This component contains glossaries for query improvement and prompts from response generation.
* **Backend**: The backend processes API requests from frontend, handling the intricate task of understanding and generating responses integrating to services from LLMs and Data Layers

**Frontend**

The frontend of the KA should be a straightforward web interface, typically crafted using React and JavaScript. Design should consider ease of use, for users to simply ask questions, receive answers, and access guidelines for effective interaction with the KA. This interface design may consider inclusion of a feature for users to provide feedback, essential for refining the KA’s performance.

Responses to user queries should be supported by clearly cited sources to offer a reliable reference for the shared information. Additionally, answers may include links to relevant enterprise micro-sites or suggest contacts within the organization who can offer further assistance on the topic. This approach adds a layer of practical utility to each response, directing users to additional resources or personnel that can provide more in-depth support or information.

The modular design of the KA architecture plays a key role here. It allows for the possibility of substituting a frontend with alternative interfaces in the future, such as a mobile app or an instant messaging platform. This flexibility comes about because the backend interactions occur through APIs, enabling seamless integration with various frontends while maintaining consistent functionality and user experience.

**Operational Stores**

Operational stores form the backend persistence layer, responsible for the storage of conversation history, user settings, feedback, and other critical operational data essential for the KA to be functional. Conversation history is particularly important for providing historic context to the LLM, enhancing the relevance of responses in ongoing interactions.

Additionally, the information gathered in operational stores is crucial for the continuous improvement of the KA. This data is analyzed within the Data Layer to identify trends and areas for enhancement, directly influencing the KA’s development and refinement. Insights derived from this analysis are then presented in the Reporting Layer, providing a comprehensive view of the KA’s interactions and effectiveness, which is vital for its ongoing optimization and success.

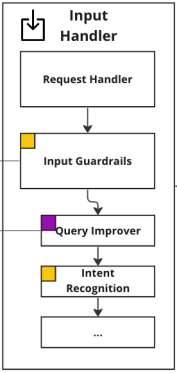
**Backend**

The backend is where the core business logic of the KA resides. It’s structured as a set of components, each with a single responsibility, working together to process user interactions efficiently. At a high-level, it is an orchestration of different decisions and LLM operations.

It handles critical functions such as accessing the Data Layer and LLMs, analyzing incoming requests, formulating messages, and delivering responses. Each component is designed to perform its specific task effectively, ensuring that the entire process from query intake to response delivery is smooth and precise.

The following section traces a user query through the complete backend architecture.

**Input Handler**



**Request Handler**

The Application Layer of the KA activates upon receiving a user query through an API. The Request Handler manages chat interactions and retrieves the last few messages in a conversation from the Conversation History Store. Additionally, the Request Handler loads the current configurations for the LLMs used in the application.

**Input Guardrails**

Once the necessary database operations are completed, the *Input Guardrails* apply. In any specific context, input guardrails encompass a selection of policies, business rules, and validations, and are designed to ensure that incoming requests meet predefined criteria and may proceed. The primary objective of these guardrails is to prevent users from using the system in ways that deviate from its intended purpose. For instance, in the scenario of a flight booking app, customers should not have the capability to inquire about other passengers beyond the scope of their valid booking.

Guardrails are essentially a stack of functions arranged in a predetermined order. Each function evaluates the incoming request input and its metadata and takes one of three possible actions. The possible actions include “pass” indicating that the guardrail approves the request without any issues; “update” this suggests the request requires modification before being allowed to pass; and “reject” signaling that the request failed the guardrail and cannot continue for processing, this terminates the process and returns a rejection reason to the requester. This approach ensures that requests that fail the guardrails are rejected early and if requests require modifications before being shared further then this is handled appropriately, this not only ensures adherences to intended use cases but also efficiently processes incoming requests for maximum reliability.

One such update-guardrail is the Query Improver*.* This component is crucial for adapting domain-specific terminology to enhance later retrieval processes. In many industries and business, queries include niche jargon, abbreviations, and phrases unique to the industry. These can be obscure or have different meanings in general language. To address this, the implementation should include a comprehensive glossary of company-specific terms and abbreviations. This glossary “translates” and modifies user queries for optimal retrieval. For instance, it could remove trailing punctuation and expand acronyms in the queries (e.g. “MVP” is reformulated as “MVP (Minimum Viable Product)”). Such alterations significantly boost the retrieval effectiveness on proprietary data.

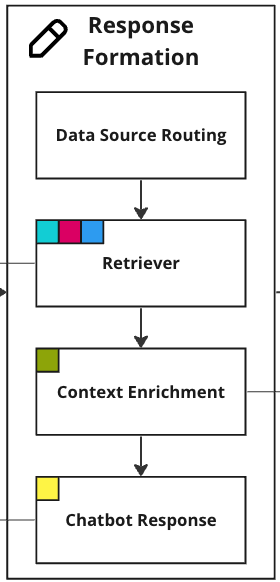
Eliminating punctuation aids in aligning the query’s semantic similarity with a corpus, which predominantly consists of statements rather than questions. Expanding abbreviations is doubly beneficial: it increases the prominence of key terms in the retrieval process, ensures coverage of content that may only use the expanded form, and aids the chat model in accurately interpreting the user’s intent. Such refinements are instrumental in enhancing the overall performance and accuracy of a KA.

The next step in the process is the Intent Recognition module, a common feature in LLM applications designed to bring structure to the typically unstructured nature of LLMs. This module’s function is to categorize each user query into one of several pre-defined intents. The identified intent plays a dual role: it guides the subsequent control flow within the application and enhances the effectiveness of the knowledge retrieval system.

The most reliable method for intent recognition isn’t a highly specialized machine learning model but rather an LLM. To improve the LLM’s accuracy, we suggest a few-shot prompting technique with balanced examples for each intent.

For instance, if we have five intents and three examples per intent to ensure accurate classification, then every intent is represented with three examples. This gives us a total of 15 examples in the prompt. This method is highly effective for setups with fewer than ten intents, achieving over 90% accuracy. However, it’s important to note that this approach has its limitations. As the number of intents increases, adding more examples becomes less practical, and distinguishing between intents becomes more challenging.

**Response Formation**



**Data Source Routing**

The Data Source Routingmodule determines where the KA receives its knowledge based on the user’s intent. With the user’s intent, the KA picks between three primary data sources, each accessible through a custom search algorithm or external APIs:

1. **Vector Store**: Text documents, like PDFs, PowerPoints, and Word documents. All chunks have metadata that enables filtering like title, abstract, authors etc.
2. **People Directory API**: Personnel information, specific skills, and contact details.
3. **Internal Wiki API**: Company-related information, IT instructions, HR documents and more.

The real advantage of intent recognition lies in its flexibility to incorporate additional data sources as needed. Beyond enhancing the control over the KA’s outputs, selective querying of data sources offers another significant benefit. While many solutions emphasize vector stores and semantic similarity search, not all data types are equally suited for such methods. For example, a people directory, with its distinct data, doesn’t fit as seamlessly into an embedding database as long documents do. In a standard similarity search, even well-detailed people profiles might not rank high enough to be included in the top results. Intent recognition circumvents this issue through clearly defined control flow. This behavior can be implemented using different chains for different intents, as in the example below.

INTENT\_CHAIN\_MAPPING = {

IntentType.KNOWLEDGE: KnowledgeChain,

IntentType.PEOPLE: PeopleChain,

IntentType.SUPPORT: WikiChain,

IntentType.CHAT: ChatChain,

IntentType.DOMAIN\_KNOWLEDGE: DomainKnowledgeChain,

}

def get\_response(question: str, conversation\_history: List[Message]):

chain = intent\_chain\_mapping[intent]

llm = AIGateway(model=’gpt-4’)

llm\_task = asyncio.create\_task(chain.acall(question, llm, conversation\_history))

# ...

response = await llm\_task

return response

### ---------------- ###

class KnowledgeChain(Chain):

# ...

class DomainKnowledgeChain(Chain):

# ...

class ChatChain(Chain):

# ...

class PeopleChain(Chain):

# ...

**Retriever**

The question is then passed to the Retriever*.*Depending on the targeted search the retriever will either embed the question through LLM Gateway and perform a semantic similarity search, use it for a keyword search, or pass it to an external API that handles the retrieval.

The Retriever should be tailored to manage different types of data effectively. Each data source not only varies in content but also in the optimal amount of information to retrieve. For example, the breadth and depth of data needed from a people directory differs significantly from the same required from a knowledge base.

To address this, the retrieval logic needs customization. For people-related queries, the retriever is configured to return a concise list of the top five most relevant contacts from the directory. In contrast, a search for knowledge yields a broader set, pulling up to 20 chunks of information to provide a more comprehensive context.

This approach, however, is not rigid. For specific intents where a more integrated perspective is beneficial, a Retriever should combine data from multiple sources. For instance, a user seeking guidance in a specific domain will receive information both from the vector store and the wiki as a mix is likely to be most useful to the user.

Fine-tuning the process means definition of a search algorithm that can combine different semantic similarity search algorithms, exact keyword matching and metadata filtering. Additionally, it may require manual tuning in how many items to retrieve from each source for each intent.

Continuous feedback loops with early-access users are crucial in the optimization process, to iteratively refine the retrieval strategies until the balance of quantity, quality, and source diversity is just right.

**Context Enrichment**

A KA’s context enrichment phase requires crafting effective prompts. These prompts must harmoniously blend instructions from the assistant, retrieved context, and the chat history. This process is heavily influenced by the detected intent and come with varying levels of difficulty in consolidating data from different sources. A significant challenge may typically arise in ensuring the relevance and conceptual cohesion for queries seeking pure knowledge. To mitigate reliance on semantic search and enhance accuracy of the final chat completion, there should be consideration given to the strategy inspired by the **MapReduceChain** in LangChain7 (see KnowledgeChain example below).

This method involves deploying parallel LLM calls for each information chunk, instructing the model to not just evaluate but also synthesize information across these chunks. This approach is pivotal in ensuring that source citations are accurate. Instead of depending on the LLM to reproduce source links – a method prone to inaccuracies – you should embed source referencing directly into code logic.

Furthermore, you should integrate recent conversation history into this enrichment process, enhancing the KA’s ability to provide contextually relevant responses. One strategy uses a common buffer window approach, focusing on the last 3-5 exchanges. The approach not only ensures relevance and continuity in conversations but also conserves tokens, proving more efficient than longer memory spans or more complex methodologies.

class KnowledgeChain(Chain):

# ...

async def acall(

self,

question: str,

llm: LanguageModel,

conversation\_history: List[Message],

retrieve\_k: int = 20,

filter\_k: int = 5,

):

documents = self.retrieve(question, k=retrieve\_k)

filtered\_documents = self.filter(documents, k=filter\_k) # Map step

llm\_response = self.answer(filtered\_documents, question, conversation\_history) # Reduce step

answer = self.postprocess(llm\_response, filtered\_documents)

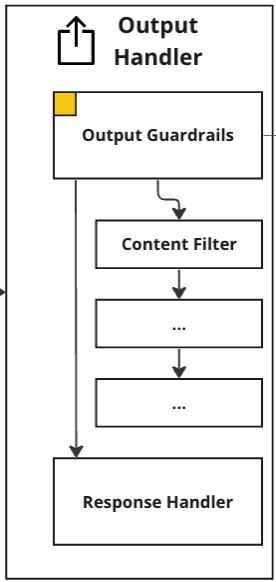
return answer

**KA Response**

After all the steps above to collect relevant information, the next step answers the user’s question using an LLM. As a reminder, the LLM prompt needs to include data from the databases (see Data Source Routingand Retriever*)*and conversation history. Additionally, it’s necessary to give clear instructions to the LLM and format the input data correctly, to determine its behavior, tone, and adherence to the given data.

Precise prompt engineering becomes a real challenge, as the outputs need to be accurate and reliable for critical decision-making. The diversity of topics, spanning hundreds of complex subjects, presents another layer of complexity. To ensure the quality and relevance of responses, there should be a set of early users to test the experience, and subject matter experts from various fields to test the correctness of the KA’s responses. Their insights will be invaluable in refining the KA’s instructions and guiding the selection of source documents.

**Output Handler**



**Output Guardrails**

After receiving the final chat completion from the LLM, the next step post processes it in the Output Handler*.* It is typical to find that no matter how carefully you engineer a prompt and steps in advance of the model, there always remains a final risk of hallucinations, and undesirable information being shown to users8. To mitigate this risk, there should be a set of Output Guardrails in place. These are a set of asynchronously executed checks on the model’s response that include a content filter and a hallucination detector. The content filter detects and removes biased and harmful language as well as removing any personal identifiable information (PII). The hallucination detector checks whether there is any information in the response that is not given in the retrieved context. Both guardrails are based on LLMs. Besides mitigating risk, they also inform future development and troubleshooting efforts.

**Response Handler**

Afterward, if the chat completion passes the output guardrails, the final step formats the response, and sends it to the front end for the user to read.

**Summary of Considerations**

We summarize some of the considerations covered earlier in this article.

* **Chunking:** The selection of a chunking strategy significantly impacts the performance of semantic similarity search, the use of context, and the understanding of specific knowledge topics by the language model.
* **Guardrails:** Implementing guardrails for input/output is crucial to mitigate risks and ensure the reputation of AI applications in enterprise settings. These guardrails can be customized and developed according to the organization’s risk requirements.
* **Configuration Database:** Maintaining a database table to track LLM configurations allows for efficient monitoring, potential rollback capabilities, and the association of specific model versions with user feedback and errors.
* **Search:** Fine-tuning the search algorithm involves combining semantic similarity search algorithms, exact keyword matching, and metadata filtering, while continuously optimizing retrieval strategies based on user feedback to achieve the right balance of quantity, quality, and source diversity.
* **Prompt Engineering:** Effective prompting is key to the success of an application and can be collaboratively done with users and/or experts.
* **Controlling LLMs:** Introducing intent recognition or a similar deterministic split enhances control flow and provides developers with more control over the behavior of LLM applications.
* **Making Data LLM-ready:** Cleaning unstructured data from artifacts (e.g., footers in the middle of chunks) and adding relevant metadata (e.g., titles) to chunks allows LLMs to effectively understand different data types.
* **Separating Data Sources:** While it may be tempting to mix all types of data in a vector store and use semantic similarity search, different data types have different requirements, and querying them separately yields much better results.
* **Domain Knowledge:** Incorporating specific knowledge through glossaries, prompt engineering, or fine-tuning is essential for LLMs to understand industry or company-specific knowledge.

**Conclusion**

In the realm of corporate technology, the integration and application of LLMs offer intriguing insights into the evolving landscape of data management, system architecture, and organizational transformation. This article aims to shed light on these aspects, with an emphasis on the broader application of LLMs within corporate settings.

Our discussion is just the beginning of a deeper exploration into the various layers, including data handling, LLM optimization, and impact assessment, essential for deploying advanced LLM applications.

Future articles will delve into the general infrastructure requirements and best practices for implementing LLMs in a corporate environment, along with exploring diverse AI use cases. For those interested in the expanding field of LLMs and their scalable applications, we invite suggestions on topics of interest.

Don’t forget to subscribe to the MLOps Community Newsletter to ensure you don’t miss our upcoming content.

**Footnotes**:  
1.<https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier>

2.<https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/what-every-ceo-should-know-about-generative-ai>

3.<https://arxiv.org/abs/2005.11401>

4.<https://vercel.com/guides/vector-databases>

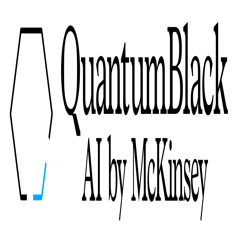
5.<https://www.pinecone.io/learn/chunking-strategies/>

6.<https://kedro.org/>

7.<https://python.langchain.com/docs/modules/chains/document/map_reduce>

8.<https://www.nytimes.com/2023/07/27/business/ai-chatgpt-safety-research.html>

**Authors**

* 

[QuantumBlack Team Jannik Wiedenhaupt, Roman Drapeko, Mohamed Abusaid, Nayur Khan](https://mlops.community/author/quantumblack-team/)

QuantumBlack, AI by McKinsey unlocks the power of artificial intelligence to help organizations blend AI and cutting-edge solutions with strategic thinking and domain expertise.

[View all posts](https://mlops.community/author/quantumblack-team/)

* 

[Jannik Wiedenhaupt](https://mlops.community/author/jannik-wiedenhaupt/)

Data Scientist at [McKinsey](https://www.mckinsey.com/capabilities/quantumblack/how-we-help-clients) || M.S. Columbia || CDTM || TU Munich

[View all posts](https://mlops.community/author/jannik-wiedenhaupt/)

* 

[Roman Drapeko](https://mlops.community/author/roman-drapeko/)

Distinguished Data Engineer (Snr. Director) at [QuantumBlack, AI by McKinsey](https://www.mckinsey.com/capabilities/quantumblack/how-we-help-clients" \t "_blank)

[View all posts](https://mlops.community/author/roman-drapeko/)

* 

[Mohamed Abusaid](https://mlops.community/author/mohamed-abusaid/)

Associate Partner at [QuantumBlack, McKinsey & Co](https://www.mckinsey.com/capabilities/quantumblack/how-we-help-clients" \t "_blank).

[View all posts](https://mlops.community/author/mohamed-abusaid/)

* 

[Nayur Khan](https://mlops.community/author/nayur-khan/)

Partner - [QuantumBlack, AI by McKinsey](https://www.mckinsey.com/capabilities/quantumblack/how-we-help-clients" \t "_blank)🔹DataIQ 100 - 2023🔸Keynote Speaker🔹Scaling AI🔸D&I Lead🔹MLOps🔸Responsible AI🔹Software Engineering

[View all posts](https://mlops.community/author/nayur-khan/)

**Related posts:**

[**[](https://mlops.community/empowering-language-model-applications-understanding-and-evaluating-vector-databases-in-production/)Empowering Language Model Applications: Understanding and Evaluating Vector Databases in Production**](https://mlops.community/empowering-language-model-applications-understanding-and-evaluating-vector-databases-in-production/) [**[](https://mlops.community/%f0%9f%a4%b8%e2%9a%95%ef%b8%8funleashing-the-power-of-large-language-models-in-healthcare-and-wellness-practical-context-providing-in-healthcare-and-wellness-with-mistral-7b/)🤸⚕️Unleashing the Power of Large Language Models in Healthcare and Wellness: Practical Context Providing in Healthcare and Wellness with Mistral 7B**](https://mlops.community/%f0%9f%a4%b8%e2%9a%95%ef%b8%8funleashing-the-power-of-large-language-models-in-healthcare-and-wellness-practical-context-providing-in-healthcare-and-wellness-with-mistral-7b/) [**[](https://mlops.community/building-neoways-ml-platform-with-a-team-first-approach-and-product-thinking/)Building Neoway’s ML Platform with a Team-First Approach and Product Thinking**](https://mlops.community/building-neoways-ml-platform-with-a-team-first-approach-and-product-thinking/) [**[](https://mlops.community/what-i-learned-building-platforms-at-stitch-fix/)What I Learned Building Platforms at Stitch Fix**](https://mlops.community/what-i-learned-building-platforms-at-stitch-fix/) [**[](https://mlops.community/vector-similarity-search-from-basics-to-production/)Vector Similarity Search: From Basics to Production**](https://mlops.community/vector-similarity-search-from-basics-to-production/)

Tags: [Knowledge Assistant](https://mlops.community/tag/knowledge-assistant/), [LLMs](https://mlops.community/tag/llms/), [MLops](https://mlops.community/tag/mlops/)

[Privacy Policy](https://mlops.community/privacy-policy/)  
  
©2025 MLOps Community. All rights reserved unless states. Images provided by [Unsplash.com](https://unsplash.com/) and [pexels.com](https://pexels.com/" \t "_blank).Made with ♥, tea and biscuits.

* [Join](https://mlops.community/join/)
* [Learn](https://learn.mlops.community/)
* [Tools](https://mlops.community/learn/)
* [Blog](https://mlops.community/blog/)
* Events
* [Videos](https://home.mlops.community/public/content)
* [Partner](https://mlops.community/partnerships/)
* [Slack](https://go.mlops.community/slack)
* [Youtube](https://go.mlops.community/youtube)
* [Medium](https://go.mlops.community/medium)
* [Twitter](https://go.mlops.community/twitter)
* [Linkedin](https://go.mlops.community/linkedin)