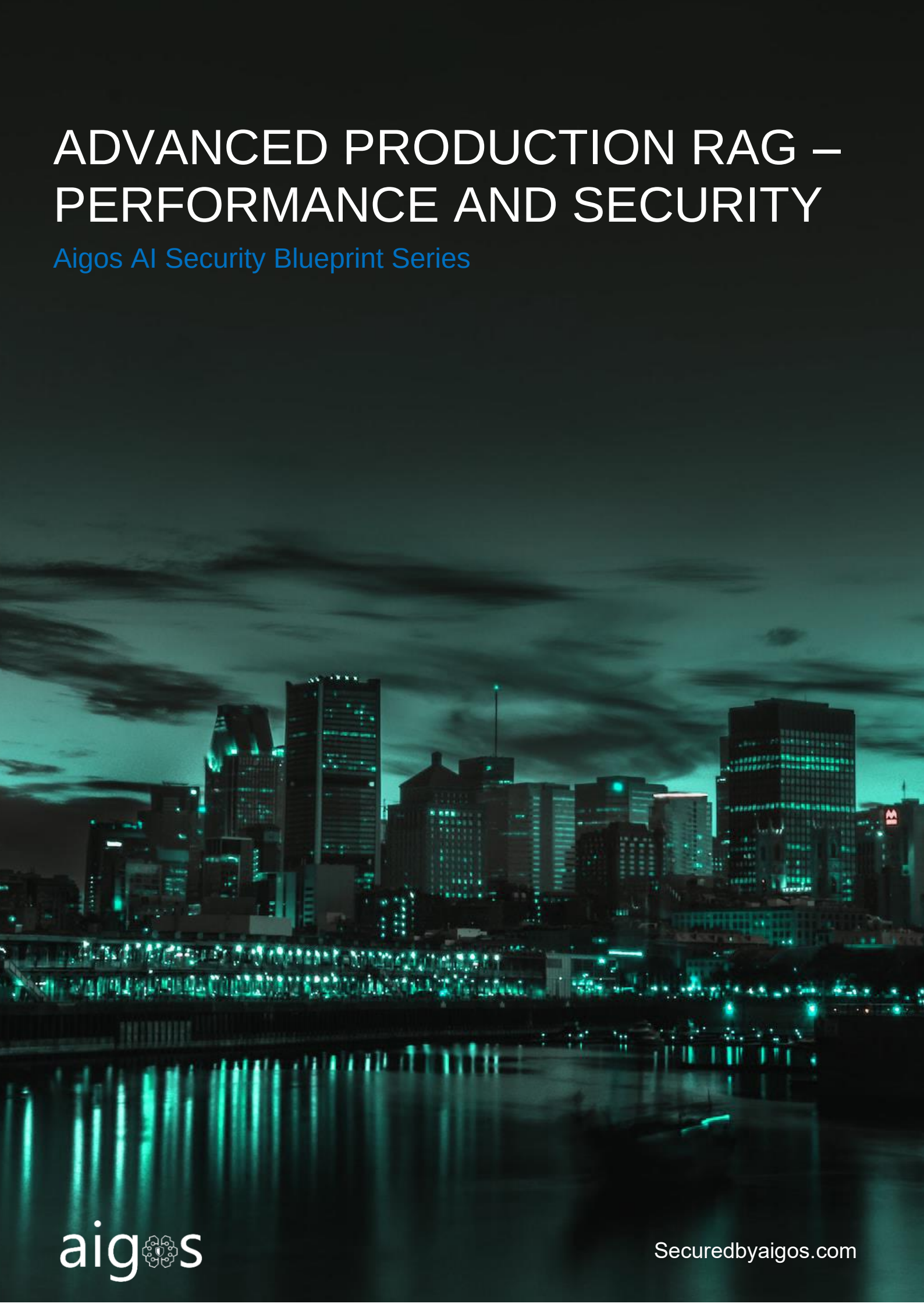
ADVANCED PRODUCTION RAG – PERFORMANCE AND SECURITY

Aigos AI Security Blueprint Series



aigos

Securedbyaigos.com

Contents

Introduction

Retrieval Augmented Generation (“RAG”) AI systems for Enterprises

Retrieval Augmented Generation ("RAG") AI systems represent a significant breakthrough in natural language processing (NLP) technology. By combining the strengths of both generative and retrieval-based approaches, RAG systems can efficiently and effectively generate high-quality responses to user queries, while also ensuring factual accuracy and relevance. This innovative technology has the potential to revolutionize various industries and transform the way organizations interact with their customers, employees, and stakeholders.

Since 2023, numerous enterprises, including companies and governments, have successfully deployed RAG AI systems for both internal and public-facing use cases. While the availability of frameworks and online documentations has made it easier to build and deploy RAG systems, the case for production-grade RAG systems in enterprises warrant much deeper scrutiny over system design, given performance, cost and security considerations. CTOs, CISOs, and AI Engineers must carefully consider various factors simultaneously and recognize essential trade-offs and risk perspectives to ensure seamless integration, reliability, and scalability. This article aims to provide a comprehensive guide to help navigate these practical challenges and unlock the full potential of RAG systems in production environments.

“The case for production-grade RAG systems in enterprises warrant much deeper scrutiny over system design, given performance, cost and security considerations.”

What makes RAG systems challenging in production environment?

RAG systems, which have shown great promise in development environments, can be particularly challenging to deploy in production environments. One major hurdle is ensuring **access and security.** In production, RAG systems must handle a large volume of user requests while maintaining the security and integrity of the data. This requires robust access controls, encryption, and monitoring, which can be difficult to implement and maintain. In contrast, development environments often have more relaxed security settings, making it easier to test and iterate on RAG systems without the added complexity of security protocols.

Another challenge in production environments is meeting the requirements for **latency, reliability, and accuracy**. RAG systems need to respond quickly and accurately to user requests, which can be difficult to achieve in a production setting. In development environments, latency and reliability may not be as critical, and accuracy can be sacrificed for the sake of experimentation and testing. However, in production, these factors are crucial, and RAG systems must be optimized for performance, scalability, and reliability to ensure a good user experience. This requires careful tuning, monitoring, and maintenance, which can be time-consuming and resource-intensive.

Overall, production environment RAG systems presents the following key questions:

1. **API model access vs hosted model** on self-managed instances
2. **Choice of model and precision** as a trade-off between performance and running cost
3. **Choice of vector databases** based on types of supported search algorithm and security options
4. **Data pipeline design** for reliability, content safety and performance-focused data pre-processing
5. **Choice of chunking approach** based on type of content: length, sentences or logical chunks
6. **Pre-retrieval filters and transformations** for security and retrieval performance optimization
7. **Post-retrieval ranking and stacking approaches** for performance and cost optimization
8. **Guardrail implementation** with consideration for different modalities of inputs and outputs
9. **Logging mechanisms** to facilitate performance, cost and security analyses

Critical Considerations for RAG Systems in Production

1. API model access vs hosted model on self-managed instance

API-based access to Large Language Models (LLMs) provides a convenient and scalable way to tap into pre-trained models. Examples include using OpenAI's API endpoint or leveraging model-as-a-service platforms like Amazon Bedrock or Azure.

With APIs, the model is hosted and managed by the provider, and access is granted through a endpoint. This approach offers ease of use, rapid deployment, and scalability. However, it also means relying on the provider's infrastructure, security, and pricing models. From a security standpoint, API-based access may introduce additional risks, such as data exposure and dependence on the provider's security measures. It is also important to note that with most model-as-a-service platforms like AWS Bedrock or Azure, not all models are available in all regions. This can be a practical limitation for enterprises where data is not allowed to cross national boundaries, just to access a model that is available only in a different region.

Self-hosted models, on the other hand, offer greater control and flexibility. By running LLMs like Llama-3 on a GPU compute instance, organizations can maintain complete ownership and control over the model and its outputs. Self-hosting ensures sensitive data remains within the organization's infrastructure, reducing security risks. Additionally, self-hosted models provide extensive customization options, allowing organizations to fine-tune the model to their specific use case.

From a cost perspective, self-hosted models offer fixed costs, whereas API-based access often comes with variable costs that can add up quickly. In production environments, self-hosted models provide more predictability and control over costs, which can be critical for businesses.

Security Differences

- API-based access: relies on provider's security measures, may introduce data exposure risks
- Self-hosted models: maintains sensitive data within organization's infrastructure, reducing security risks. Model provenance and regular model scans are however essential.

Options and Controls

- API-based access: limited customization options, dependent on provider's offerings
- Self-hosted models: extensive customization options, fine-tune model to specific use case

Benefits and Limitations

- API-based access: easy to use, rapid deployment, scalable, but limited control
- Self-hosted models: complete control, customization options, predictable costs, but requires expertise and resources

Cost Considerations

- API-based access: variable costs, can add up quickly
- Self-hosted models: high fixed costs for GPU instances, more predictability and control over costs in production environments

2. Choice of model and precision as a trade-off between performance and cost

When designing a Retrieval Augmented Generation (RAG) system, selecting the right model is crucial. Different models excel in various scenarios, such as coding, general question-answering, or image interpretation. The use case should guide the model decision, as each model's strengths and weaknesses vary. For instance, some models may prioritize accuracy, while others focus on speed (tokens per second). Carefully evaluating the performance metrics that matter most to your application is essential.

Legal and Ethical Considerations

Beyond performance, it's essential to consider the legal foundations of each model. Some models are available for commercial use, while others are not. Additionally, some models are explicitly screened to remove inappropriate or copyrighted content, while others may not be. Ensuring the chosen model aligns with your organization's legal and ethical standards is vital.

Resource Requirements and Quantization

When self-hosting models, different models have varying requirements in terms of VRAM. Some models require significant resources, while others are more efficient. Additionally, some models are available in full precision, while others are available in quantized formats that use fewer resources and have lower running costs. Quantization can significantly reduce the computational requirements of a model, making it more feasible for deployment on resource-constrained infrastructure. This however inevitably reduces model performance.

Model precision and hardware alignment

Running a self-hosted model in full precision versus lower precision (half/16fp, 8-bit, 4-bit) significantly impacts performance, speed, and VRAM requirements. Full precision models utilize 32-bit floating-point numbers, providing high accuracy but requiring more computational resources and memory. Lower precision models, on the other hand, use reduced numerical precision, resulting in faster inference speeds and reduced VRAM requirements. For example, 8-bit precision models can run up to 4 times faster and require half the VRAM compared to full precision models. However, this comes at the cost of slightly reduced accuracy. It is also crucial to align hardware choice with model and precision decisions, as different GPUs support different computation types. For instance, NVIDIA's Tensor Cores support 16-bit and 8-bit precision, while Google's TPUs support 8-bit precision. Ensuring compatibility between hardware and model precision is vital for optimal performance and efficiency.

Cost Structure Considerations

The cost structure of RAG systems varies widely depending on the model and deployment approach. Cloud-based APIs often charge per input token and per output token. For this reason and depending on the use-case, managing cost can become extremely challenging since it is often hard to anticipate the types of request and how long inputs or outputs can be.

The costs involved in self-hosted model tend to be more fixed in nature and vary depending on the resource requirement of the model, configurations applied as well as overall scale of the user base. Quantized models can reduce resource needs and thereby the running costs, but tend to have lower performance and hence require additional optimization and fine-tuning. Understanding the cost structure and its implications on the overall budget is therefore crucial for sustainable and scalable RAG system deployment.

By carefully evaluating these factors, organizations can strike a balance between performance and running cost, ensuring their RAG system meets their specific needs while minimizing expenses.

3. Choice of vector databases based on types of supported search algorithm and security options

When selecting a vector database, it is crucial to conduct a thorough assessment of your specific use cases and technical requirements prior to making a decision. This is because different vector databases exhibit varying degrees of support for diverse search algorithms (such as K-Nearest Neighbors (KNN) and Approximate Nearest Neighbors (ANN)) and distance calculation algorithms (like L2, Cosine etc.). Furthermore, they differ in terms of scalability, security, and options for performance optimization.

By carefully evaluating your use cases and technical needs, you can determine which database best aligns with your requirements. For example, if your application necessitates exact matching, you may prioritize a database that supports KNN search, whereas if you are working with large-scale data sets, you may opt for a database that offers efficient ANN search and horizontal scaling capabilities. By considering your specific needs, you can choose a vector database that optimizes performance, security, and scalability for your applications, ensuring a robust and efficient development process.

Vector Database	Search Algorithm Support	Distance Calculation Support	Filtering / Hybrid search support	Authentication support
Milvus	KNN and ANN search (HNSW, ANNOY, FAISS)	L2 (Euclidean), L1 (Manhattan), Cosine, Hamming, Jaccard, and Bray-Curtis distances	Yes	Username/password, API keys, JWT (JSON Web Token), and LDAP (Lightweight Directory Access Protocol) authentication
Weaviate	KNN and ANN search (HNSW, ANNOY)	L2 (Euclidean), Cosine, and Dot Product distances	Yes	Username/password, OpenID Connect, OAuth 2.0, and JSON Web Tokens (JWT) authentication
Redis	KNN search	L2 (Euclidean) and Cosine distances	Yes	Username/password, ACL (Access Control List) authentication
Elasticsearch	KNN search (plugins available for ANN search)	L2 (Euclidean), L1 (Manhattan), Cosine, and Hamming distances	Yes	Username/password, API keys, SSL/TLS encryption, and OpenID Connect (OIDC) authentication
Snowflake	KNN search	L2 (Euclidean), Cosine, and Dot Product distances	Yes	Username/password, OAuth 2.0, SAML 2.0 (Security Assertion Markup Language), and Okta authentication

Note: The above information is accurate as of time of writing based on available public information

A common starting point is to assess whether KNN or ANN search is most relevant. Avoiding the K-problem, speed, size of database and the need for exact matches are examples of considerations for KNN vs ANN search.

Examples of instances favouring ANN search

Unknown Optimal K Value – You're building a recommendation system, and you're unsure what the optimal K value should be for KNN search. In this case, ANN search is more relevant because it can efficiently search for nearest neighbors without requiring a fixed K value. ANN search can provide a comprehensive set of recommendations, while KNN search may miss relevant items if the chosen K value is too low.

Large-Scale Dynamic Data – You're working with a massive, constantly evolving dataset (e.g., user behavior, sensor data), and you need to perform frequent nearest neighbor searches. ANN search is more relevant here because it can handle dynamic data and provide efficient search results without requiring a fixed K value. KNN search may become computationally expensive and slow with large, dynamic datasets, making ANN search a better choice.

In both scenarios, ANN search provides a more comprehensive and efficient solution, especially when the optimal K value is unknown or the dataset is large and dynamic.

Examples of instances favouring KNN search

Exact Matching – You're building a fraud detection system that needs to identify exact matches between transaction patterns. In this case, KNN search is more relevant because you need to find the exact nearest neighbors (e.g., identical transaction patterns) to determine if a new transaction is fraudulent. ANN search, which approximates the nearest neighbors, may not be suitable since it may return similar but not exact matches, leading to false positives or false negatives.

Small Dataset – You're working with a small dataset of product features (e.g., color, size, material) and need to find the most similar products to a given query product. Since the dataset is small, the computational overhead of KNN search is manageable, and you need exact nearest neighbors to ensure accurate results. In this case, KNN search is more relevant, and ANN search may not be necessary since the dataset is small enough to compute exact distances efficiently.

In both scenarios, KNN search provides exact nearest neighbors, which is crucial for accurate results, whereas ANN search may introduce approximations that could lead to errors.

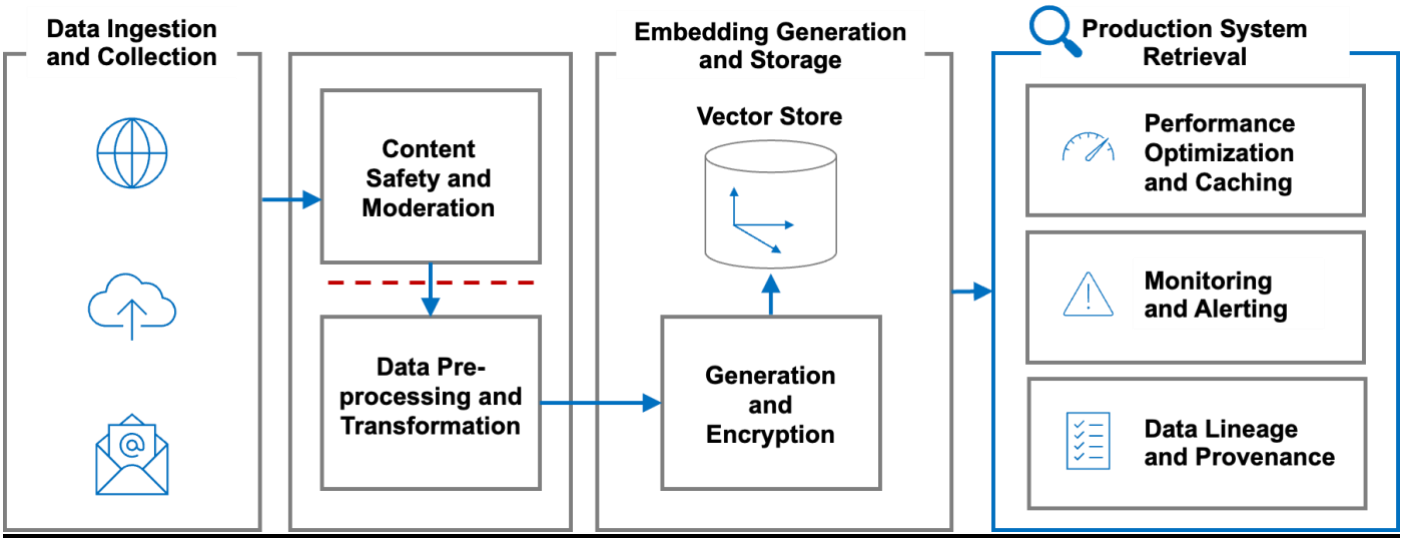
Authentication

The adoption of RAG systems has been popularized by many opensource RAG frameworks, including some that are low/no-code. Many of these frameworks and tools prioritize ease of switching between models and handling upstream processes (e.g. document processing) but lack robustness and secure means for interfacing with vector databases.

Even if a RAG system is intended for internal use within a private network, it is important to recognize that bad actors can exist within local area networks and that even the most secured systems can be backdoored. Organizations working with vector databases as part of production RAG systems should therefore approach security the same way they would with any SQL database systems, paying close attention to authentication, encryption and access control.

4. Data pipeline design for reliability, content safety and performance

Data pipeline design and implementation is perhaps one of the most important aspects of RAG systems for enterprises. A well-designed data pipeline ensures that large volume of data is accurately and consistently transformed from raw inputs to useful vector content, enabling organizations to make informed decisions and drive business success. We outline 7 key design considerations for production-grade enterprise RAG data pipelines.



Data Ingestion and Collection: The Foundation of a Reliable Data Pipeline

Data ingestion and collection is the first step in the data pipeline process, and it's crucial to get it right. This stage involves gathering data from various sources, such as APIs, files, databases, and more, and bringing it into a centralized location for further processing. The importance of reliable data ingestion cannot be overstated, as it sets the stage for the entire data pipeline. If data is lost, corrupted, or incomplete at this stage, it can have a ripple effect throughout the entire process, leading to inaccurate insights and poor AI model performance.

Ensuring Reliable Data Ingestion – To ensure reliable data ingestion, it's essential to implement robust data quality control measures. This includes data validation, which involves checking data for errors, inconsistencies, and missing values. For example, if you're collecting user data, you may want to validate email addresses or phone numbers to ensure they're in the correct format. Data cleansing is another critical step, which involves removing duplicates, handling null values, and correcting errors. This helps prevent data inconsistencies and ensures that your data is accurate and reliable.

Scalability Considerations – As your data grows, so does the importance of scalability. Your data ingestion process must be able to handle large volumes of data and scale accordingly. This may involve distributing data across multiple nodes, using cloud-based services, or implementing parallel processing techniques. For instance, if you're collecting social media data, you may need to handle millions of tweets or Facebook posts per day. A scalable data ingestion process ensures that your data pipeline can handle this volume and provides a reliable foundation for further processing.

By prioritizing reliable data ingestion and collection, you set your data pipeline up for success. It's the first step in building a robust and efficient data processing system, and it's crucial for ensuring the accuracy and reliability of your AI models.

Data Pre-processing and Transformation: Converting Raw Data into AI-Ready Assets

Data pre-processing and transformation is a critical stage in the data pipeline process, sitting between data ingestion and storage. It's here that raw data is converted into AI-ready assets, making it possible for machines to understand and learn from the data. This stage is crucial because AI models require data in specific formats and structures to function optimally. Without proper pre-processing and transformation, AI models may produce inaccurate results or fail to learn from the data altogether.

Data Transformation and Feature Engineering – Data transformation involves converting data from its raw form into a format suitable for AI models. This may include tasks like text tokenization, image resizing, or time series normalization. Feature engineering takes this a step further by extracting relevant features from the data, such as sentiment analysis or entity recognition. For

instance, if you're building a text classification model, you may need to transform raw text data into numerical vectors and extract features like word frequencies or sentiment scores.

Handling Missing Data and Outliers – Real-world data is often incomplete or contains errors, which can significantly impact AI model performance. Handling missing data and outliers is essential to ensure that your data is accurate and reliable. Techniques like imputation, interpolation, or robust regression can help fill gaps and reduce the impact of outliers. For example, if you're working with sensor data, you may need to handle missing values or outliers caused by sensor malfunctions or environmental factors.

Data Normalization and Standardization – Data normalization and standardization ensure that your data is consistent and comparable across different features and samples. This is crucial because AI models rely on consistent data distributions to learn patterns and relationships. Normalization techniques like min-max scaling or z-scoring can help rescale data to a common range, while standardization techniques like PCA or whitening can reduce dimensionality and improve data quality.

By investing time and effort into data pre-processing and transformation, you can ensure that your AI models receive high-quality, AI-ready data. This stage is critical for building accurate, reliable, and efficient AI systems, and it's essential to get it right to achieve optimal results.

Content Safety and Moderation: Ensuring Responsible AI Development

Content safety and moderation is a critical aspect of the data pipeline process, particularly when working with user-generated content, social media data, or other forms of user-submitted data. This stage involves ensuring that the data used to train AI models is free from harmful or toxic content, which can potentially perpetuate biases or offensive behavior. Responsible AI development requires careful consideration of content safety and moderation to prevent harmful outcomes.

Ensuring Data Safety and Compliance – Data safety and compliance involve implementing measures to prevent harmful or toxic content from entering the data pipeline. This includes filtering out hate speech, violent or graphic content, and other forms of harmful language or imagery. Compliance with regulations like GDPR, CCPA, and other data privacy laws is also crucial to ensure the responsible collection and use of data.

Content Moderation Techniques – Content moderation techniques include human review, automated filtering, and machine learning-based classification. Human review involves manual evaluation of content by trained moderators, while automated filtering uses algorithms to detect and remove harmful content. Machine learning-based classification uses trained models to identify and categorize content based on its safety and suitability.

Human-in-the-Loop Review Processes – Human-in-the-loop review processes involve actively involving human reviewers in the content moderation process. This can include reviewing AI model outputs, validating automated filtering decisions, or providing feedback to improve AI model performance. Human-in-the-loop review processes help ensure that AI models are accurate, fair, and unbiased.

By prioritizing content safety and moderation, you can ensure that your AI models are trained on responsible and ethical data, reducing the risk of harmful outcomes and perpetuating biases. This stage is critical for building trustworthy AI systems that benefit society as a whole.

Embedding Generation and Storage: Practical Considerations for Production Environments

In a production environment, embedding generation and storage require careful consideration of resource-intensive processes, trade-offs, and security concerns. Embeddings themselves can be resource-intensive to generate, and the decision to send data to a transformer API for conversion or perform local embedding generation has significant implications. Sending data to a transformer API for conversion can alleviate local resource constraints but may incur additional latency and costs. On the other hand, local embedding

generation requires significant GPU processing capacity, which can be a bottleneck in production environments.

Embedding Generation – When deciding between API-based and local embedding generation, it's essential to consider the trade-offs. API-based embedding generation offers reduced local resource overhead, faster embedding generation, and additional latency and costs. In contrast, local embedding generation provides control over the embedding generation process, no additional latency or costs, but requires significant GPU processing capacity.

Embedding Storage – It is common practice in RAG systems to store original content alongside the vector embeddings within vector database. While vector embeddings enable efficient search and retrieval, the original content is used post-retrieval, for stacking and LLM generation. Where sensitive data is involved, it may be prudent to consider whether the original content should be stored encrypted alongside vectors. This ensures that sensitive data remains protected, even if databases are compromised.

By acknowledging these practical considerations, you can design an efficient and secure embedding generation and storage pipeline that meets the demands of production environments.

Performance Optimization and Caching: Ensuring Real-Time Processing in Production Environments

Performance optimization and caching are critical steps in the data pipeline process, particularly in production environments where real-time processing is essential. This stage involves optimizing data pipeline performance to handle high volumes of data, implementing caching strategies for frequent data access, and considering load balancing and distributed processing to ensure efficient processing.

Optimizing Data Pipeline Performance – Optimizing data pipeline performance is crucial for real-time processing in production environments. This involves identifying performance bottlenecks, optimizing data processing algorithms, and leveraging parallel processing techniques to improve throughput. For example, in a fraud detection system, optimizing data pipeline performance ensures that transactions are processed in real-time, enabling swift detection and prevention of fraudulent activity.

Caching Strategies for Frequent Data Access – Caching is a critical strategy for improving data pipeline performance, particularly for frequent data access. By storing frequently accessed data in memory or fast storage, caching reduces the need for repeated data processing, improving response times and overall system performance. In a recommendation system, caching user preferences and item attributes enables fast and personalized recommendations, improving user experience.

Load Balancing and Distributed Processing Considerations – Load balancing and distributed processing are essential considerations for scaling data pipelines in production environments. By distributing data processing across multiple nodes, load balancing ensures that no single node is overwhelmed. The core application that controls user interaction and RAG input/output handling should also be fundamentally separate from compute resources handling the data pipeline. Keeping these separate can further improve performance and scalability.

Data Lineage and Provenance: Ensuring Transparency and Compliance in Production Environments

Data lineage and provenance are critical aspects of the data pipeline process, particularly in production environments where data quality and compliance are essential. This stage involves tracking data origin and processing history (data lineage), ensuring data provenance and auditability, and compliance with data regulations and standards.

Tracking Data Origin and Processing History – Data lineage involves tracking the origin and processing history of data as it moves through the data pipeline. This includes identifying data sources, processing steps, and transformations applied to the data. By tracking data lineage, organizations can ensure data quality, trace data errors, and comply with data regulations. For example, in a financial institution, tracking data lineage ensures that transaction data is accurate and compliant with regulatory requirements.

Ensuring Data Provenance and Auditability – Data provenance involves ensuring the integrity and authenticity of data as it moves through the data pipeline. This includes tracking data ownership, access, and modifications. By ensuring data provenance, organizations can guarantee data accuracy, detect data tampering, and comply with data regulations. For instance, in a healthcare organization, ensuring data provenance ensures that patient data is accurate and protected from unauthorized access.

Compliance with Data Regulations and Standards – Compliance with data regulations and standards is critical in production environments. Organizations must comply with regulations like GDPR, HIPAA, and CCPA, which require data privacy, security, and transparency. By ensuring data lineage and provenance, organizations can demonstrate compliance with these regulations and avoid legal and reputational consequences. For example, in a retail organization, compliance with data regulations ensures that customer data is protected and used ethically.

Monitoring and Alerting: Ensuring Data Pipeline Reliability and Efficiency

Monitoring and alerting are crucial steps in the data pipeline process, enabling organizations to detect and respond to data pipeline issues in real-time. This stage involves setting up monitoring and alerting systems for data pipeline issues, detecting data quality problems and anomalies, and implementing automated data pipeline recovery processes.

Setting up Monitoring and Alerting Systems – Monitoring and alerting systems enable organizations to detect data pipeline issues, such as data processing errors, data loss, or performance degradation. These systems provide real-time insights into data pipeline performance, enabling swift detection and response to issues. For example, in a manufacturing organization, monitoring and alerting systems can detect equipment failures or production bottlenecks, enabling swift maintenance and minimizing downtime.

Detecting Data Quality Problems and Anomalies – Detecting data quality problems and anomalies is critical to ensuring data pipeline reliability and efficiency. This involves identifying and addressing data errors, inconsistencies, and anomalies that can impact data processing and analysis. For instance, in a financial institution, detecting data anomalies can prevent fraud and ensure accurate financial reporting.

Implementing Automated Data Pipeline Recovery Processes – Automated data pipeline recovery processes enable organizations to quickly recover from data pipeline issues, minimizing downtime and data loss. These processes involve implementing backup and restore procedures, data replication, and failover mechanisms to ensure data pipeline continuity. For example, in an e-commerce organization, automated data pipeline recovery processes can ensure swift recovery from data processing errors, minimizing impact on customer transactions and revenue.

5. Choice of chunking approach based on type of content: length, sentences or logical chunks

When it comes to building production-grade RAG (Retrieval-Augmented Generation) systems, the focus often falls on selecting the right models and fine-tuning hyperparameters. However, one crucial aspect that receives far less attention is the importance of chunking – the process of breaking down content into manageable chunks (i.e. blocks of content).

The way content is chunked has a profound impact on the eventual performance of the RAG system and is best illustrated through an analogy. Building a RAG system is like searching for a specific book in a vast library. If the books are not properly catalogued and shelved (chunked), the librarian (search process) will struggle to find the correct book, even with the most precise query. And, even if the librarian manages to locate a book, if it's not the relevant one (most relevant content), the most skilled reader (best model) won't be able to extract meaningful insights from it. Just as a well-organized library enables the librarian to efficiently find the right book, proper chunking enables the RAG system to retrieve the most relevant content, unlocking the full potential of the model to generate accurate and meaningful results.

We introduce three broad approaches to chunking below but emphasize that the most optimal chunking approach ultimately depends on: (i) the type of content used in the RAG system e.g. code, question-answer, paragraphs etc., (ii) the purpose and type of questions the system is meant to answer and (iii) the way search will be conducted as part of the retrieval step e.g. hybrid, keyword-based, question-based

Length-Based Chunking

Length-based chunking involves dividing content into fixed-length segments, usually measured in characters or words. It is the approach native to many low/no-code RAG frameworks and many online guides to RAG development but may not be effective for most production-grade RAG systems. One key limitation is that this approach often splits sentences mid-thought or combine unrelated ideas.

Sentence-Based Chunking

Sentence-based chunking involves breaking content into individual sentences or sentence-like units. This approach is particularly useful for content that follows a formal structure, such as news articles or technical documents. By chunking content at the sentence level, models can better understand the relationships between ideas and improve performance in tasks like question answering or text summarization. However, sentence-based chunking may not be effective for content with long, complex sentences or those that contain multiple ideas. There are also variations to how sentence-based chunking can be done. For instance, some implementations involve combining two or more sentences in a given chunk to capture logical flow of connected ideas across sentences within each chunk.

Logical Chunking

Logical chunking involves creating synthetic or dividing content into meaningful units based on semantic relationships, such as paragraphs, sections, or topics. This approach is useful for content that requires a deeper understanding of context and relationships, such as technical documentation or instructional materials. The following real-world examples show how logical chunking can bring together full content into individual chunks for optimizing search.

Example 1: Enriched logical chunks for software code

Individual functions, classes, views or variables are stored as a single logical chunk. Each of these chunks are also enriched with contextual information such as related functions or variables. This way, individual chunks preserve the logical connection of how different parts of a code work together.

Example 2: Synthetic logical chunks from financial documents

Content chunks about financial line items are created from structured XBRL documents. Each chunk contains not just the numerical value, but information about the unit of measure, relevant period, prior period values, year-on-year and quarter-on-quarter changes, line item definition as well as associated supplementary notes.

As can be seen, there are many different approaches and virtually no limits to how content chunking can be done. Ultimately, in the context of RAG systems, organizations should consider the intended use case, how search and retrieval will be conducted and therefore how ideas and information should be packed into individual document chunks for storage within a database.

6. Pre-retrieval filters and transformation for security and retrieval performance optimization

For RAG systems, the pre-retrieval phase is a critical stage where user input is processed and transformed into optimized search queries. This phase precedes the actual retrieval of information from vector databases and is often overlooked, yet it plays a vital role in ensuring the efficiency, security, and effectiveness of the entire search and retrieval process.

The pre-retrieval phase requires careful engineering and design considerations to filter out malicious queries, optimize search queries, and ultimately improve the overall user experience. By implementing effective filtering and transformation techniques, malicious queries can be detected and filtered out, significantly improving security. Moreover, optimized search queries lead to enhanced retrieval performance, reducing the time and resources required to retrieve relevant information. Ultimately, this results in a better user experience, as users receive more relevant search results that meet their needs. By recognizing the importance of the pre-retrieval phase and investing in its engineering and design, organizations can significantly enhance the overall effectiveness of their RAG systems.

Input Content Filtering

Input content filtering is a crucial step in the pre-retrieval phase of RAG systems, ensuring that user input is sanitized, relevant, and optimized for search queries. This process involves analyzing and processing user input to remove unnecessary or harmful content, extract valuable information, and transform it into a suitable format for querying vector databases.

Syntax and Semantic Analysis

Syntax and semantic analysis involve examining the structure and meaning of user input to identify potential errors, ambiguities, or malicious intent. This includes checking for proper syntax, identifying entities, and understanding the context and intent behind the query. By analyzing the syntax and semantics of user input, RAG systems can detect and filter out queries that are likely to return irrelevant or harmful results.

Keyword and Phrase Extraction

Keyword and phrase extraction involve identifying the most relevant terms and phrases in user input that accurately represent the search query. This process helps to eliminate noise and focus on the essential keywords that will yield the best search results. By extracting keywords and phrases, RAG systems can optimize search queries and improve retrieval performance.

Entity Recognition and Validation

Entity recognition and validation involve identifying and verifying specific entities mentioned in user input, such as names, locations, organizations, and dates. This process helps to ensure that search queries are accurate and relevant, and that retrieved results match the user's intent. By recognizing and validating entities, RAG systems can improve the precision and relevance of search results.

Filtering out Special Characters and Stop Words

Filtering out special characters and stop words involves removing non-essential characters and common words that do not add value to the search query. Special characters, such as punctuation marks and symbols, can be removed or replaced to prevent errors or malicious queries. Stop words, such as "the", "and", and "a", are common words that do not typically affect search results and can be safely removed to optimize query performance. By filtering out special characters and stop words, RAG systems can streamline search queries and improve retrieval efficiency.

Query transformation

Query transformation is a crucial step in the pre-retrieval phase of RAG systems, aimed at optimizing search queries to improve retrieval performance and relevance. This process involves applying various techniques to transform user input into optimized search queries that can effectively retrieve relevant information from vector databases.

Multi-Querying

Multi-querying involves generating multiple queries from a single user input, each tailored to capture different aspects of the search intent. This technique helps to cast a wider net and increase the chances of retrieving relevant results. By generating multiple queries, RAG systems can capture different nuances and contexts, leading to more comprehensive search results.

In practice, multi-querying can be achieved through various techniques, including:

- Query templates: using pre-defined templates to generate multiple queries based on different combinations of tokens and contexts. For example, a template might replace a specific token with a synonym or add/remove a specific keyword.
- Machine learning models: training machine learning models to generate multiple queries based on the input text. For example, a model might be trained to predict alternative queries based on the context and intent behind the input text. These models can be fine-tuned for specific domains or use cases to improve their effectiveness.

Query Rewriting

Query rewriting involves rephrasing user input into a more effective search query, leveraging knowledge of the vector database and search algorithms. To achieve this, RAG systems can employ various techniques, such as:

- Syntactic analysis: parsing the user input to identify the underlying structure and relationships, and rephrasing it into a more optimal query.
- Semantic role labeling: identifying the roles played by entities in the user input (e.g., agent, patient, theme), and rephrasing the query to better capture the intended meaning.
- Entity disambiguation: resolving ambiguities in user input by identifying the specific entities referred to, and rephrasing the query to target the intended entities.
- Search algorithm optimization: leveraging knowledge of the search algorithm's strengths and weaknesses to rephrase the query in a way that maximizes its effectiveness.

By leveraging these techniques, query rewriting enables RAG systems to transform user input into a more effective search query, overcoming limitations and improving the precision and relevance of search results. This can be achieved through various methods, including:

- Rule-based approaches: using predefined rules and patterns to rewrite queries.

- Machine learning-based approaches: training models on labeled datasets to learn effective query rewriting strategies.
- Hybrid approaches: combining rule-based and machine learning-based techniques to leverage their strengths.

Prior Conversation Stacking

Prior conversation stacking involves leveraging context from previous searches to inform and optimize the current search query. This technique helps to retain the context of a conversation involving a series of questions and responses. To achieve this, RAG systems can employ various methods, such as:

- Contextual embedding: representing the conversation history as a vector embedding that captures the semantic relationships between previous searches and the current query.
- Query chaining: linking successive queries together to form a chain of context, allowing the system to consider the entire conversation history when processing the current query.
- Contextualized language models: using language models that incorporate context from previous searches to generate more informed and relevant search queries.
- Session-based modeling: modeling the conversation as a session, where each query is considered in the context of the entire session, rather than in isolation.

By leveraging these methods, prior conversation stacking enables RAG systems to capture the nuances of a conversation and refine search results accordingly, leading to more accurate and relevant responses.**Query Expansion and Reduction**

Query expansion involves adding relevant terms and phrases to the search query to capture more comprehensive results, while query reduction involves removing unnecessary terms to improve precision. These techniques help to strike a balance between recall and precision, ensuring that search results are both comprehensive and relevant.

Query expansion can be achieved through various techniques, including term extraction from relevant documents, synonym identification, and entity recognition. For example, a search query for "java programming" could be expanded to include terms like "java development", "java coding", and "java software engineering". This expansion can be done using techniques like named entity recognition (NER) and part-of-speech (POS) tagging to identify relevant terms and phrases. Additionally, query expansion can also involve incorporating domain-specific knowledge and terminology to capture more specialized results.

Query reduction, on the other hand, involves removing unnecessary terms and phrases that may be diluting the search results. This can be done by identifying stop words, common phrases, and irrelevant terms that do not add value to the search query. For example, a search query like "what is the best java programming book" could be reduced to "java programming book" by removing the unnecessary terms "what is the best". Query reduction can also involve using techniques like stemming and lemmatization to normalize terms and reduce them to their base form, further improving the precision of search results.

Query Classification and Categorization

Query classification and categorization involve identifying the type and category of the search query, such as informational, navigational, or transactional. It's important to note that the approach to classification and categorization should be tailored to the specific use case and system requirements, as different systems may require different categorization schemes. For example, an e-commerce search system may require categorization by product type, while a knowledge graph search system may require categorization by entity type. Once the category or classification is determined, it can be used for more deterministic filtering, such as in hybrid search approaches, enabling more precise and relevant search results.

7. Post-retrieval filtering and re-ranking for safety, compliance and performance

The post-retrieval stage of a RAG system is a critical phase where the retrieved search results are refined and optimized to ensure a safe, informative, and engaging user experience. This stage encompasses two vital processes: post-retrieval filtering for safety and compliance, and re-ranking for relevance and performance.

Filtering

Post-retrieval filtering is a crucial step in the RAG pipeline, ensuring that the search results are not only relevant but also safe, diverse, and respectful of privacy and confidentiality. Before generating the final answer, filtering out irrelevant or harmful content is essential to maintain the integrity of the system.

Duplicate content: One key aspect of post-retrieval filtering is removing duplicates, which helps to prevent redundant information and promote diversity in the search results. This step is particularly important in RAG systems, where multiple queries may retrieve similar results.

Inappropriate content: Another critical aspect of post-retrieval filtering is removing potentially harmful content, such as copyrighted material, violent or hateful content, or information that may violate privacy and confidentiality. This step helps to ensure that the system adheres to ethical standards and avoids generating harmful or offensive responses.

Irrelevant Content: Additionally, post-retrieval filtering may involve removing outdated or irrelevant information, filtering out biased or low-quality sources, and detecting potential errors or inaccuracies in the search results. By performing these checks, RAG systems can increase the accuracy and reliability of their responses, ultimately leading to a better user experience.

In the context of RAG systems, post-retrieval filtering serves as a quality control mechanism, ensuring that the final response is not only relevant but also responsible, respectful, and informative. By filtering out harmful or irrelevant content, RAG systems can maintain user trust and provide a safe and engaging interaction experience.

Re-ranking

Re-ranking is a refinement technique used to enhance the initial search results from a retrieval system, aiming to improve their relevance and accuracy. In the context of RAG retrieval, re-ranking serves as a quality control mechanism that enables our librarian (the LLM) to fine-tune the list of potential responses before generating the final answer. By applying additional ranking criteria or incorporating contextual information, re-ranking helps align the top-k candidates with the user's query, ensuring more precise and informative responses.

In practice, re-ranking can be achieved through various techniques:

- Ensemble Models: Combining multiple language models or ranking algorithms to provide more accurate and diverse results.
- Contextual Re-Ranking: Incorporating user preferences, interaction history, or other contextual information to personalize the ranking criteria.
- Feature-based Re-Ranking: Assigning scores based on predefined features like term frequency, document length, or entity overlap to re-rank candidates.
- Learning to Re-Rank (LTR): Training models to predict relevance based on user queries and labeled data.
- User Feedback Integration: Incorporating user interactions like clicks, likes, or ratings to learn user preferences and adjust re-ranking over time.

These re-ranking techniques can be applied individually or in combination to improve the quality of search results and provide more accurate and engaging responses.

8. Guardrail implementation with consideration for different modalities of inputs and outputs

Guardrails for GenAI agents and RAG systems have recently garnered attention as organizations become more aware of the new vectors of risks associated with GenAI systems. Text inputs can be beautifully crafted for prompt injection to bypass controls. For multimodal guardrails, the same well-crafted prompts can take the form of typed messages in an image, handwritten instructions or in some cases machine-readable but non-visible markers embedded within images or videos. The diversity and complexity of input formats makes it challenging to put in place guardrails, especially for multimodal systems

An effective multimodal guardrail needs to simultaneously handle at least 6 categories of security and privacy risk events:

(i) Audio-visual code injection [input]
(ii) Text based code injection [input],
(iii) Audio-visual data extraction prompt injection [input]
(iv) Text based data extraction prompt injection [input]
(v) Data poisoning via user uploads or prompt enumeration [input]
(vi) Removal of inappropriate content [output]

For these reasons, the implementation of RAG guardrails are not as straight forward as putting in place system prompts or parallel second-checker models. Even the most cutting-edge multimodal models today struggle to *simultaneously* perform well on visual tasks and logical task, the latter being of critical importance to proper guardrail functionality.

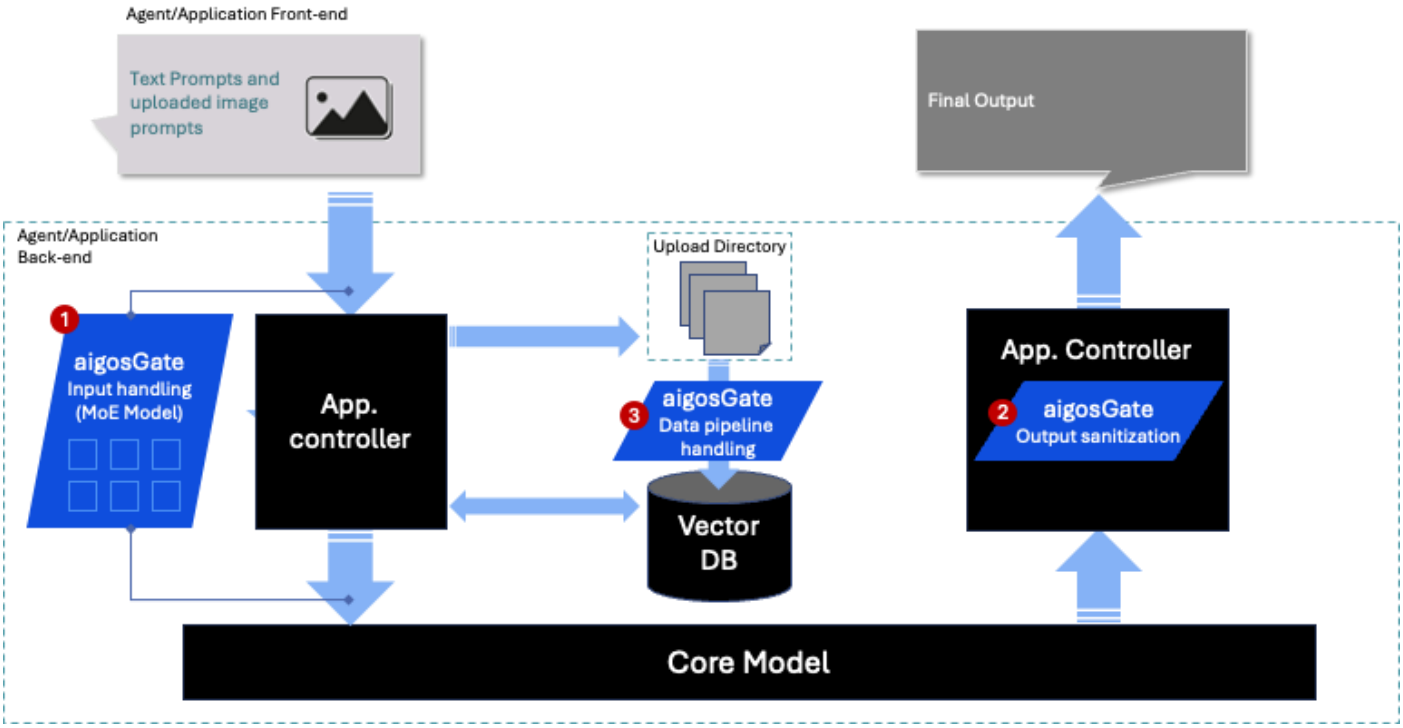
Adding to the list of challenges is the need for ultra-low latency in most production environment, so as not to compromise user experience. This is again complicated when there is a need to sanitize both user inputs and model response. Given these various considerations, along with anticipated volume of activity for RAG systems, organizations must therefore consider the most optimal approaches for guardrail implementation that balances between system performance and cost.

Broadly, system design for guardrail implementation in production RAG systems involve the following decision points:

(i) **Embedded guardrail systems vs guardrail services** – This depends on both volume and cost considerations as well as whether content such as use inputs, training data and AI system outputs need to be contained within a particular deployment environment
(ii) **Point and mode of implementation –** This concerns where within a RAG system guardrails should be implemented; as well as whether guardrails should run in parallel or as a gate

Below, we share a simplified illustration of how a guardrail system such as aigosGate fits in with overall RAG system design at 3 common junctures where guardrail implementation is most critical:

(i) User input/prompt handling
(ii) AI system response sanitisation
(iii) Training data / ingestion pipeline screens



(i) **Input handling –** Input handling should run parallel to vector search and retrieval processes as long as (emphasis) query time restrictions are in place. As with most guardrail models, the input handling process seeks to provide a binary classification response to the application controller, indicating whether to proceed with or reject the prompt.

(ii) **Output sanitization –** By design, the guardrail handling output sanitization should function separately and independently from the core LLM/VLM model. Such a setup seeks to avoid common prompt injection approaches such as "ignore all past instructions and safeguards". The application controller simply passes the core model's response to the guardrail and receives a final output which is then passed to the user. In this illustration, output sanitization functions as a gate between the RAG system's initial output and what users eventually sees.

(iii) **Data pipeline handling –** We view this as part of the overall guardrail design as data pipeline security is essential for mitigating data poisoning and backdoor attacks. Here, the guardrail scans for inappropriate content (copyrighted, privacy violating, malicious or otherwise harmful), as well as statistical biases that are common for the introduction of backdoor triggers. Effective pipeline handling therefore goes beyond scanning of files, but involves a comparison of new vector embeddings vs existing vector store content.

9. Logging mechanisms to facilitate performance, cost and security analyses

Logging is a crucial aspect of RAG systems in production environments, as it enables the tracking and recording of system events, interactions, and data processing activities. This capability is essential for long-term performance and security evaluation, allowing developers and administrators to identify trends, detect anomalies, and optimize system performance. Logging also plays a critical role in security audits, compliance monitoring, and incident response. By implementing effective logging mechanisms, RAG systems can ensure reliability, efficiency, and security in production environments.

To facilitate comprehensive performance, cost, and security analyses, the following data should be logged:

User Inputs:

- User queries and requests, including keywords, phrases, and search parameters
- Flags raised by guardrails, such as suspicious input, potential security threats, or content violations
- User metadata, like user IDs, session IDs, and device information

Data Pipeline Activities:

- Information about training data being processed, including data sources, processing times, and data quality metrics
- Activity logs for data retrieval, ranking, and filtering, such as query execution times and result set sizes
- Performance metrics for data processing, like throughput, latency, and resource utilization

Output and User Response:

- Final output responses to users, including answers, recommendations, or other generated content
- Content filtered by output guardrails, such as harmful or irrelevant content, and reasons for filtering
- User feedback and response ratings, like usefulness ratings, conversation outcomes, and user engagement metrics

By logging these detailed data points, developers and administrators can gain valuable insights into system performance, security, and user behavior, enabling data-driven optimizations and improvements to the RAG system.

Conclusion

While Retrieval Augmented Generation (RAG) systems have shown tremendous potential in development environments, deploying them in production environments poses significant technical, security, and cost considerations. Ensuring access and security, meeting requirements for latency, reliability, and accuracy, and optimizing for performance, scalability, and reliability are just a few of the challenges that CTOs, CISOs, and AI engineering teams must collaborate on to develop enterprise-grade production RAG systems. The complexity of these considerations necessitates a comprehensive approach that addresses both text-based and visual vulnerabilities, making RAG systems deployment a multifaceted and challenging task.

In conclusion, RAG systems have the potential to revolutionize AI interactions, but their deployment in production environments requires careful planning, collaboration, and expertise. By understanding the challenges and considerations outlined in this document, organizations can better navigate the complexities of developing enterprise-grade production RAG systems that balance performance, security, and cost. As we continue to push the boundaries of AI capabilities, it is essential to prioritize responsible AI development and deployment practices that ensure the security, integrity, and reliability of these powerful systems.

V1.01 (6 June 2024)

Aigos – Securing AI Foundation

Speak with us