ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Natural Language Processing

AGENTIC AND PRIVACY-PRESERVING RETRIEVAL-AUGMENTED GENERATION: ARCHITECTURE AND EMPIRICAL EVALUATION

CANDIDATE
Mohammed Oubia

SUPERVISOR
Prof. Paolo Torroni

CO-SUPERVISORS
Simone Sensidoni, company tutor
Donati Nicolò

Academic year 2024–2025
Session 5th

Abstract

In recent years, large language models (LLMs) have demonstrated strong capabilities in natural language understanding and generation. Trained primarily on publicly available data, these models are unable to answer queries related to private and sensitive organizational data, as they lack access to private knowledge sources. To overcome this limitation, two main approaches are used: fine-tuning models for specific domains or employing Retrieval-Augmented Generation (RAG) to retrieve relevant information from private knowledge bases. Fine-tuning requires high computational resources and expertise, often beyond the capabilities of small and medium-sized companies, making RAG a more practical solution. This thesis investigates the use of RAG and agentic solutions to enable LLMs to operate over domain-specific data while preserving sensitive information. It also examines the design of a multi-agent system capable of autonomously executing user tasks. The proposed AI framework was developed and evaluated in collaboration with a transportation company and an Italian banking institution, involving system design, implementation, and experimental evaluation of retrieval performance and privacy preservation strategies.

Contents

1 Introduction 1

2 Background and Related Work 4

2.1 Foundations of Natural Language Processing and Large Language Models . . . 4

2.1.1 Classical NLP and Statistical Language Models 4

2.1.2 Neural Language Models and Pretraining Paradigms . 6

2.2 Neural Architectures for Language Modeling 7

2.2.1 Recurrent Neural Networks (RNNs) 7

2.2.2 Transformer Architecture 9

2.2.3 Scaling Laws and Training of Large Language Models 14

2.3 The LLaMA Family of Models 16

2.3.1 Architectural Characteristics 16

2.4 Retrieval-Augmented Generation 19

2.4.1 Motivation and Conceptual Framework 19

2.4.2 Dense Retrieval, Embedding Models, and Chunking . 20

2.4.3 Limitations and Failure Modes of RAG 25

2.5 Privacy and Security in Retrieval-Augmented Systems 27

2.5.1 Data Leakage Risks 27

3 Materials and Methods **29**

3.1 Data sources 29

3.1.1 Historical Help Desk Records (CSV) 29

3.1.2 Data Pipeline toward Vector Storage 30
3.2 Sensitive data handling 31
3.2.1 Anonymization Pipeline 31
3.2.2 Comparative Analysis of Protected vs. Unprotected Retrieval 32
3.3 Embedding generation 33
3.3.1 Vector Semantics and the Distributional Hypothesis . . 33
3.3.2 Implementation via Sentence-BERT (SBERT) 34
3.4 Retriever setup 35
3.4.1 Query Processing and Embedding 36
3.5 Testing methodology 39
3.5.1 Datasets and Ground Truth 39
3.5.2 Retrieval Evaluation Protocol 40

4 Architecture **43**
4.1 System Overview 43
4.2 Architectural Design Principles 45
4.3 Data Acquisition and Indexing Layers 45
4.4 Retrieval and Reasoning Layer (The General Path) 46
4.5 Application Orchestration Layer (The Agentic Path) 46
4.5.1 MCP Client and Tool Governance 46
4.5.2 Tool Server Execution 47
4.6 Synthesis and Final Resolution 47
4.7 System Deployment View 47

5 Implementation **49**
5.1 Agent Orchestration Architecture 49
5.1.1 Implementation Objectives and Constraints 49
5.2 Development Environment and Framework Stack 50
5.2.1 LangGraph State and Workflow Construction 51

5.2.2 MCP Integration and Tool Governance 51
5.3 Safety and Privacy Enforcement 52

6 Evaluation 54
6.1 Metrics 54
6.1.1 Retrieval Accuracy Metrics 54
6.1.2 Privacy and Safety Metrics 55
6.1.3 Efficiency Metrics 55
6.2 Baselines 56
6.3 Experiments 56
6.3.1 Experiment 1: Embedding Model, LLM, and Chunk Size Sweep 56
6.3.2 Experiment 2: Retriever Type Comparison 57
6.3.3 Experiment 3: RAGAS Retrieval Configuration Search 57
6.3.4 Experiment 4: Privacy Pipeline Evaluation 58
6.4 Results and analysis 58
6.4.1 Experiment 1: Embedding Model, LLM, and Chunk Size Sweep 58
6.4.2 Experiment 2: Retriever Type Comparison 59
6.4.3 Experiment 3: RAGAS Configuration Search 60
6.4.4 Experiment 4: Privacy Pipeline Evaluation 61
6.4.5 Retrieval Strategy and Distance Function Analysis . . 61
6.4.6 Final Production Configuration 62
6.5 Discussion 62

7 Conclusion and Future Work 64
7.1 Summary of contributions 64
7.2 Limitations 65
7.3 Future directions 65

Bibliography 67

List of Figures

2.1 A simplified view of a classical NLP workflow based on hand-crafted features and statistical models. 5
2.2 An illustration of tokenization and two common text representations: one-hot vectors vs dense embeddings. 6
2.3 A simple RNN processes tokens sequentially, updating a hidden state at each step. 7
2.4 Reference RNN architecture diagram (input x, hidden state h with a one-step delay z^{-1}, and output y) [8]. 8
2.5 Reference LSTM architecture diagram highlighting gates (σ), element-wise operations ($\times$, $+$), and the cell state flow [8]. . . 9
2.6 An LSTM augments an RNN with a cell state and gates to control information flow over time. 9
2.7 Reference Transformer architecture [9]. 10
2.8 Self-Attention detailed architecture [10]. 11
2.9 Multi-Head Attention [11]. 12
2.10 Encoder-Decoder Architecture [9]. 14
2.11 LLaMA-Architecture [12]. 16
2.12 Rotary Positional Embeddings (RoPE) [13]. 18
2.13 RAG overview main component [15]. 19
2.14 Embedding illustration and vector space representation [17]. . 22
2.15 Chunks Representation 24

3.1 Data Anonymization Pipeline 31

3.2 Retrieval process architecture [16]. 36

4.1 End-to-end workflow of the Agentic RAG platform. 44

List of Tables

3.1 Comparison of Retrieval Performance and Privacy Security Levels 32

5.1 Attributes of the LangGraph State Definition 51
5.2 Implementation Details of the LangGraph Nodes 52
5.3 Tool Configuration and Execution Sequence 52

6.1 Top retrieval accuracy results from the configuration sweep. Score = correctly answered queries, Total = evaluation set size. 59
6.2 Latency summary per retriever. 59
6.3 Table 6.3: Round 1 RAGAS configuration exploration results. 60
6.4 Round 2 chunk-model coupling results. Different embeddings perform optimally at different chunk sizes. 60
6.5 Privacy and efficiency comparison (unprotected vs protected index). 61
6.6 Final production configuration with rationale and impact assessment. 63

Chapter 1

Introduction

The rapid advancement of Artificial Intelligence and large language models (LL Ms) has transformed the way organizations and companies approach automation and information handling. Although these models have shown broad capabilities across different sectors, they remain limited when applied to private enterprise data. Since most of the LLMs are trained on the publicly available data, they cannot directly access private companies' knowledge without additional architectural support.

In industrial contexts, adapting LLMs to domain-specific environments is typically done either by fine-tuning or Retrieval-Augmented Generation (RAG). Fine-tuning requires large computational resources, technical expertise and ongoing maintenance, making it an impractical choice for many small and medium-sized enterprises. Therefore, RAG has emerged as a more efficient and flexible alternative, enabling models to dynamically retrieve relevant information from private knowledge bases without requiring modifications to the underlying model parameters.

This thesis was written during my internship at Bitapp, a software development company based in Bologna that has invested in building AI solutions for its clients. I was responsible for researching and testing different architectural approaches and frameworks in order to identify the most effective solution for enterprise deployment. The outcome of this work was the design

and implementation of an Agentic RAG platform: a web-based multi-agent system capable of retrieving domain-specific knowledge while autonomously executing user-oriented tasks through structured workflows.

The project focused not only on building a functional system, but also on evaluating different retrieval strategies and architectural configurations, suitable for real-world industrial environments.

While RAG improves factual grounding, it introduces new privacy and security challenges. When private or sensitive information is indexed within a retrieval system, injected prompts may reveal confidential data during retrieval.

This issue is especially critical in enterprise contexts where knowledge bases may contain personally identifiable information (PII), financial data, or internal business records. Therefore, it is important to handle sensitive data during indexing, embedding, and retrieval.

Evaluating RAG systems in industrial environments is much more complex than academic evaluation based on standardized benchmarks. For this reason, we tested and introduced different evaluation approaches, such as human-in-the-loop evaluation and systematic benchmarking under real-world constraints.

This thesis focuses on the design, implementation, and evaluation of a privacy-aware, Agentic RAG platform. The research addresses two primary dimensions:

1. **Retriever Evaluation:** Comparative analysis of retrieval configurations, including vector-based search methods and embedding models using Qdrant as the vector database.

2. **Sensitive Data Handling:** Investigation of privacy-preserving strategies within the RAG pipeline, specifically comparing tokenization or redaction of sensitive fields prior to embedding against post-retrieval filtering approaches.

In addition, this work explores an *agentic architecture* in which multiple agents coordinate through structured workflows. Using LangChain and LangGraph, the system dynamically orchestrates tasks such as retrieval, reasoning, ticket generation, and route planning. [32] The Model Context Protocol (MCP) is integrated to manage tool invocation and structured API interactions. [31]

The framework is implemented as a web-based multi-agent RAG system deployed on cloud infrastructure, enabling integration of LLM services and enterprise knowledge bases. It was developed and evaluated in collaboration with two real-world organizations: a transportation company and an Italian banking institution. Experimental analysis focuses on retrieval performance, response quality, and latency considerations.

The remainder of this thesis is structured as follows:

Chapter 2 presents background and related work, including classical Natural Language processing methods, transformer-based LLMs, Retrieval- Augmented Generation paradigms and privacy risks in retrieval systems architectures.

Chapter 3 outlines the materials and methods used to construct each component of the RAG system, from data storage and manipulation to retrieval and testing methods.

Chapter 4 describes the system architecture and the integration of different components to build a robust system.

Chapter 5 outlines the implementation of different technologies and pre-existing solutions used to enhance and deploy the system.

Chapter 6 discusses the results and metrics, implications for secure RAG deployment, limitations of the current approach, and potential directions for future research.

Chapter 2

Background and Related Work

2.1 Foundations of Natural Language Processing and Large Language Models

2.1.1 Classical NLP and Statistical Language Models

Classical NLP is heavily based on *statistical models* that work on top of simple and explicit representation of text. After **Normalization** of the text(*lowercasing, removing noise and handling abbreviations*) then it follows Two key steps, **tokenization** and **feature extraction**.

Tokenization is a technique that splits raw text into basic units called *tokens* (often words or sub-words). For example, the sentence ”I like NLP” becomes the token sequence $\langle$I, like, NLP$\rangle$. Tokenization is a very important technique because most of the statistical methods require the input to be formatted as a list of individual symbols.

Feature extraction it converts tokens into number. Common choices for feature extraction are:

- *Bag-of-Words (BoW)*: counts how many times each word appears (order is ignored).
  Example: for “I like NLP” over the vocabulary {I, like, NLP}: $\langle$1, 1,

$1\rangle$.

- *n-grams*: fixed-length sequences of n consecutive tokens used to track which words typically appear together.
  Example bigrams ($n = 2$) for "I like NLP": (I like), (like NLP).
  Example trigrams ($n = 3$): (I like NLP).

For many classification tasks. these features are simple and fast to compute. they work well as long as enough labeled data is available.

Statistical language models A model that estimates how likely a word sequence is. the classic *n-gram language model* is based on assuming that the next word depends only on the previous $n - 1$ words:

$$P(w_t \mid w_{1:t-1}) \approx P(w_t \mid w_{t-n+1:t-1}).$$

This makes training and inference efficient, but it struggles with *data sparsity* (many n-grams are rare) and it cannot capture long-range dependencies.

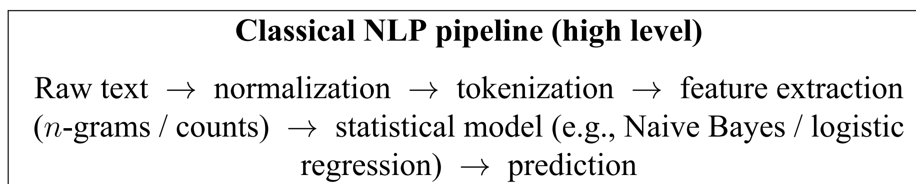

Figure 2.1: A simplified view of a classical NLP workflow based on handcrafted features and statistical models.

Tokenization and Text Representation. Tokenization defines *what a model sees as a unit ⟨I, like NLP⟩* . Common choices are:

- **Word tokenization**: split by spaces/punctuation (simple, but struggles with unknown words that it has not seen before in the data).

- **Subword tokenization** (e.g., BPE/WordPiece): split words into frequent pieces (handles rare words better). By breaking rare or complex words into common components. Effective to address the **Out-of-Vocabulary(OOV)** problem.

- **Character tokenization**: split into characters (very flexible, but sequences become long).*[”I”, ” ”, ”l”, ”i”, ”k”, ”e”, ” ”, ”N”, ”L”, ”P”]*

After tokenization, each token should be converted into numbers. In the classical NLP, this was often done with **sparse representations *one-hot vectors*** or ***count-based vectors*** (BoW / TF–IDF). In neural models, tokens are usually mapped to **dense embeddings** where it forms a vector-space representation of the tokens so similar words tend to have vectors close to each other.

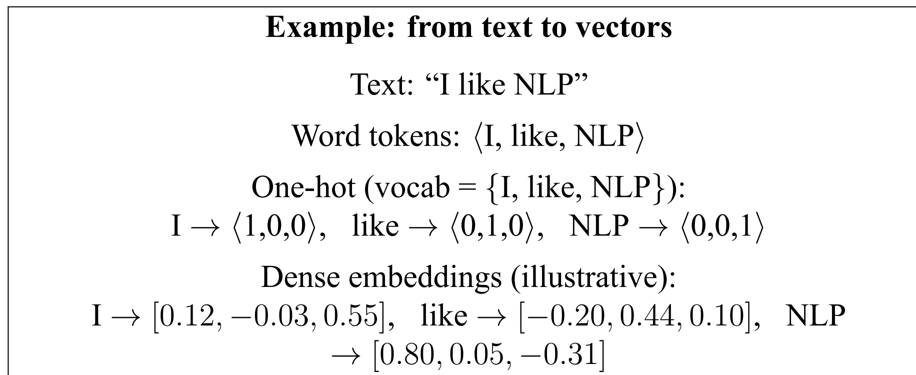

Figure 2.2: An illustration of tokenization and two common text representations: one-hot vectors vs dense embeddings.

2.1.2 Neural Language Models and Pretraining Paradigms

Modern neural language models replace hand-crafted features with **learned representations**. Instead of counting words or n-grams, the model learns internal features directly from data, which helps capture meaning and longer contexts.

Pretraining (*pre-train-then-fine-tune*) is a key paradigm for modern LLMs: a model is trained on a massive corpus, allowing it to learn general linguistic patterns before it is adapted to specific downstream tasks.

- **Autoregressive (causal) modeling**: the model learns to predict the next token given previous tokens (common in decoder-only models, e.g.,

GPT).

- **Masked language modeling**: predict missing or hidden tokens in a sentence (common in encoder-based models, e.g., BERT).

After the pre-training, the models are **fine-tuned** on a labeled data or **instruction-tuned** to follow specific prompts.

2.2 Neural Architectures for Language Modeling

2.2.1 Recurrent Neural Networks (RNNs)

In the introduction of big data and deep learning, the handling of natural language had been evolved and new neural network architecture had been introduced by scientist in order to treat text sequences where information is processed *step by step* through different layers of network. at each time step t the network reads the current token embedding x_t and updates a hidden state h_t that summarizes the past:

$$h_t = f(W_x x_t + W_h h_{t-1}),$$

$f(\cdot)$ is the activation function, and setting it to tanh (hyperbolic tangent) is the standard engine that controls how the neural network processes information.

h_t is then used to predict the next token.

Example ("I like NLP"):

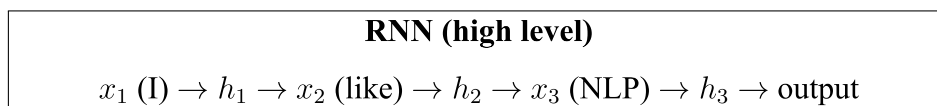

Figure 2.3: A simple RNN processes tokens sequentially, updating a hidden state at each step.

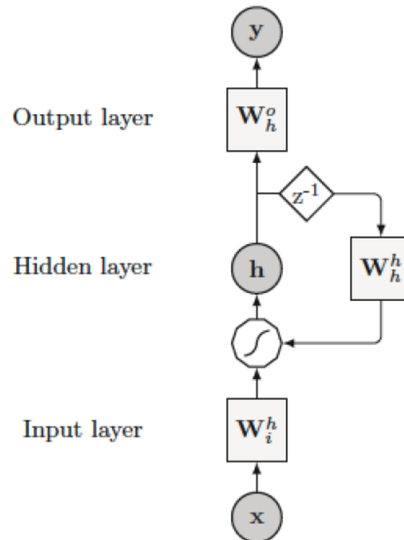

Figure 2.4: Reference RNN architecture diagram (input x, hidden state h with a one-step delay z^{-1}, and output y) [8].

RNN may have difficulty in preserving information over many steps; this issue is known as *vanishing/exploding gradients*. which motivated the use of LSTM that is a gated variant.

Long Short-Term Memory (LSTM). Long Short-Term Memory (LSTM) networks is an extend architecture of the recurrent neural networks by introducing an cell state (memory variable) and a multiplicative gates that regulate information flow. This architecture addresses the optimization difficulties of plain RNNs by providing a more stable pathway for propagating salient signals across many time steps.

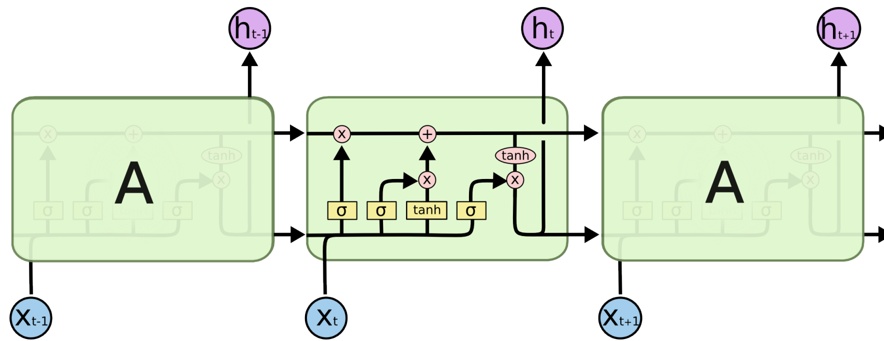

Figure 2.5: Reference LSTM architecture diagram highlighting gates (σ), element-wise operations ($\times$, $+$), and the cell state flow [8].

An LSTM maintains a hidden state h_t and a cell state c_t. And three sigmoid gates—*forget*, *input*, and *output*, The first sigmoid controls which components of c_{t-1} are retained, The second sigmoid decides what new content is written to memory, and The third decides which portion of the updated memory is exposed as h_t. In practice, this gating mechanism Maintain the vanishing gradients and improves modeling of longer-range dependencies.

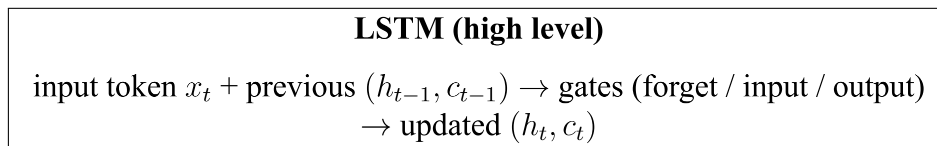

Figure 2.6: An LSTM augments an RNN with a cell state and gates to control information flow over time.

2.2.2 Transformer Architecture

Transformer is a new neural network that was introduced as an improved architecture for natural language processing. As mentioned before LSTM is improved upon the standard RNN by having the *cell state* in order to carry long-term memory. it still has some limitation in sequential processing. The model process a sentence in order. this makes it slow to be trained and it still loses information for a long documents.

The transformer solves this issue by processing the entire sequence in parallel through the architecture in the following.

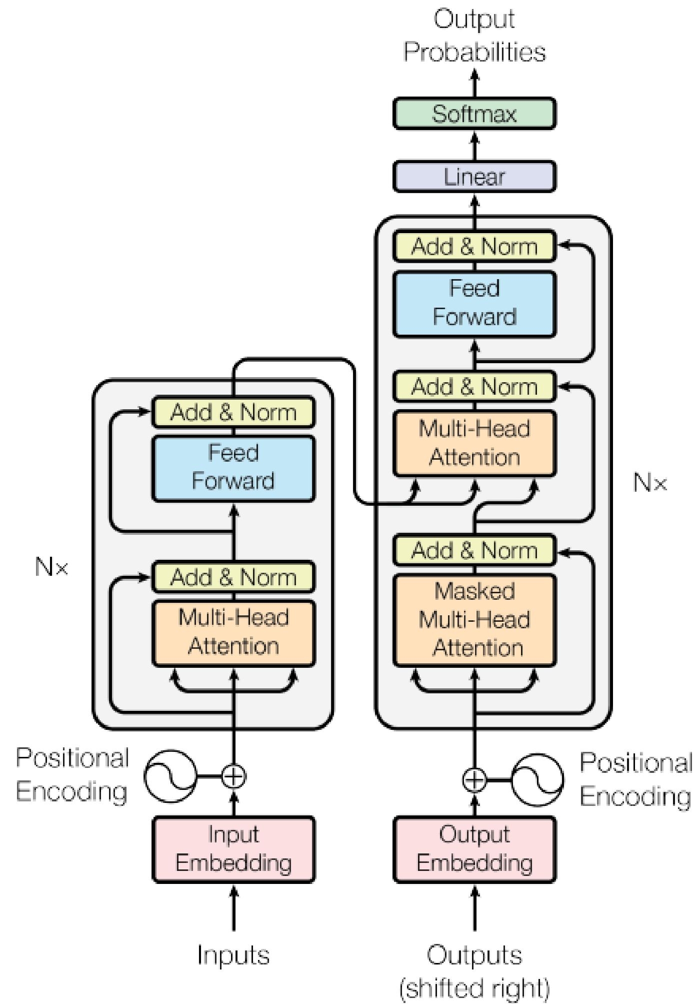

Figure 2.7: Reference Transformer architecture [9].

To Demonstrate the mechanism of transform architecture, we consider the sequence 'I like NLP' as I (x_1) like (x_2) NLP (x_3). while x_1, x_2 and x_3 are embedding vectors.

Due to Multi-Head Attention and Positional Encoding the model creates a mathematical link between any two words regardless of their position. where a link can be created between "I","like" and"NLP" ensuring that the information does not fade in case of long documents.

Self-Attention Mechanism. Self-Attention is a main-core element that allows the Transformer to determine the relevance of words in a sequence

simultaneously.

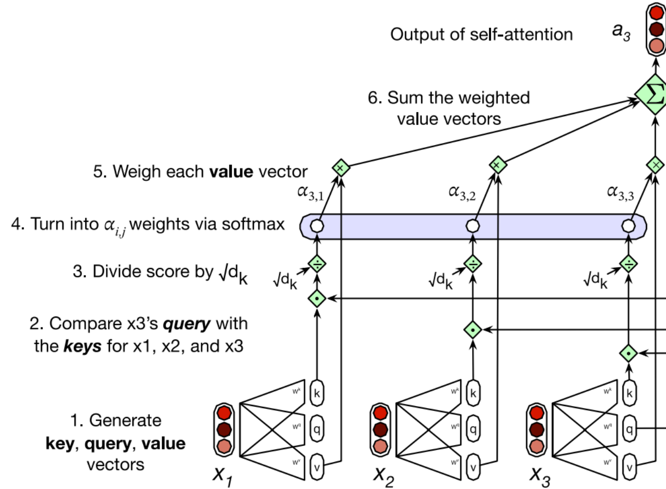

Figure 2.8: Self-Attention detailed architecture [10].

Query ($q_i = x_i W^Q$): It is used as reference point for attention. For x_2 query q_2 looks for the other tokens in the sequence that provide necessary context for x_2.

Keys ($k_i = x_i W^K$): Used as values to compare against the query. the k_1 for x_1 contains a label that identifying it in this case "I" is a subject pronoun.

Values ($v_i = x_i W^V$): Used to represent the final output.

The model on step 3 it calculates the scores between the query x_2 and the keys of all tokens.

$$\text{score}(x_i, x_j) = \frac{q_i k_j^T}{\sqrt{d_k}}$$

then the scores are normalized using the **softmax** function in order to determine the attention weights.

$$\alpha_{i,j} = \text{softmax}_j([\text{score}(x_i, x_1), \ldots, \text{score}(x_i, x_T)])$$

The final output for a token is the weighted sum of the values of every token in the sequence:

$$a_i = \sum_t \alpha_{i,t} v_t$$

If the model decides that x_1 is very important to understand the word x_2 it gives to it a high score $\alpha_{2,1}$ to that relationship. Because of this high score, x_2 will get a lot of the meaning and information from x_1.

Finally, Because all this is math and it might change the size of the data, the Model applies a final filter W^O to resize the information back to its original shape so it can easily move to the next layer of the transformer.

Multi-Head Attention. While Self-Attention allows words to interact, Multi-Head Attention allows the model to perform the search multiple times in parallel.

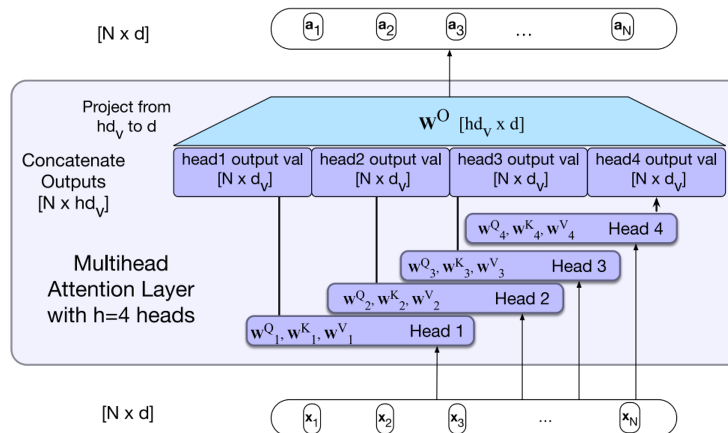

Figure 2.9: Multi-Head Attention [11].

Instead of having just one set of Q,K,V vectors for *'I like NLP'*, The model splits the data into multiple heads, where each head focuses on a different linguistic relationship(grammar, vocabulary, context, etc.).

It splits the original input into h heads, Then it calculates Self-Attention for each h independently. After that all the different output values are concatenated again in order to project them in the final steps to unify the information back into a single vector.

Feed-Forward Networks and Residual Connections.

While attention mechanism focuses on the relationships between tokens. the **Feed-Forward Network (FFN)** and **Residual Connection (Add Norm)** . It is responsible for ensuring the model remains stable during training.

Feed-Forward Networks: Is a small independent neural network applied to each position separately and identically. So Self-Attention is collecting information other words("like" gathering context from "I"), The FFN is about extracting higher-level feature information. In *"I like NLP"* after *"like"* (x_2) has attended to *"I"* (x_1), the FFN processes this combined representation to conclude that the relationship represents a *first-person preference*.

Residual Connection (Add): This mechanism introduced skipping connection that adds the original input back to the output of the layer via skipping connection. This prevents the *vanishing gradient* problem, ensuring that the input *"I like NLP"* stays the same as it goes through deep layers.

Layer Normalization (Norm): It re-scales numerical values to a standard range. It prevents mathematical values from exploding or becoming too small, to result a faster and stable training.

Positional Encoding.

Positional Encoding is added to the input embedding at the bottom of the architecture as mentioned above. In order to generate for each token x_i a unique vector based on it's position (e.g., $pos = 1$ for "I", $pos = 3$ for "NLP").

These positional vectors are added directly to the word embedding so that the resulted input contains both the word and its location in the sequence. Typically, these encoding use a sine and cosine function of different frequencies to allow the model to easily learn and attend to relative positions.

Encoder–Decoder and Decoder-Only Architectures. The full architecture diagram, the transformer is mainly composed of two main sections:

The Encoder (Left Stack): This part is responsible for processing the input *'I like NLP'* in order to create a contextualized numerical map. It uses two main layers, **Multi-Head Attention** and **Feed-Forward**, to understand the relationships between all input tokens simultaneously.

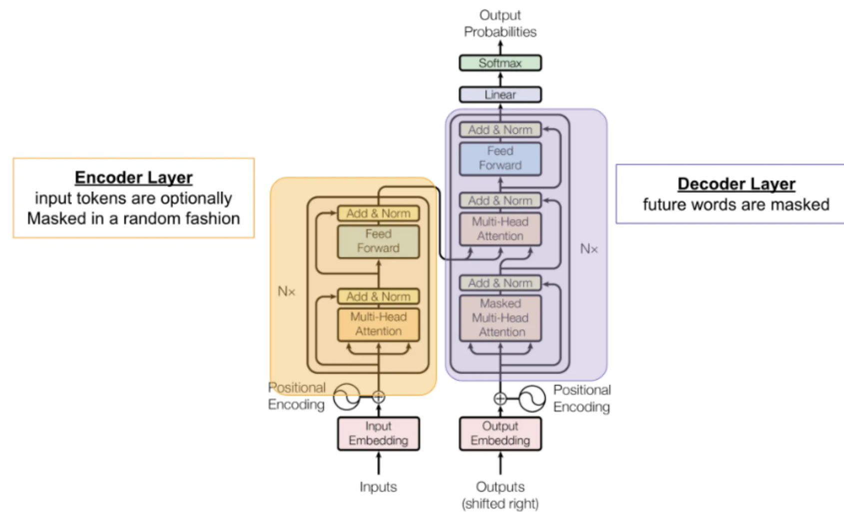

Figure 2.10: Encoder-Decoder Architecture [9].

The Decoder (Right Stack): This part is responsible for generating output. It uses an additional **Masked Multi-Head Attention** layer, which ensures that while predicting a word, the model can only look at previous words and not look at the future tokens that it is supposed to predict.

General Transformer uses both, But many modern Large Language Models have evolved to use only one component.

Encoder-Only: Models like BERT are great for understanding text (classification, sentiment).

Decoder-Only: Models like GPT focus purely on text generation by predicting the next token in a sequence.

2.2.3 Scaling Laws and Training of Large Language Models

The Transformer architecture went from being a translation tool to a foundation architecture for modern Large Language Models.

Scaling Laws is a law that suggest model accuracy can improve significantly as long as these elements are increased:

- **Model Size**: The number of parameters, That represents number of neurons in a the model.

- **Dataset Size**: The amount of text data used for training (moving from small datasets to the entirety of the web).

- **Compute**: The total amount of floating-point operations (FLOPs) used during the training process.

Training of Large Language Models is divided in two primary stages.

- **Pre-training**: The model mainly trained on unlabeled massive data using a *Self-supervised* predicting the next word in a sentence.

- **Fine-tuning**: After the model learns general language, the model is further trained on domain specific data in order to specialized in specific tasks.

2.3 The LLaMA Family of Models

The *LLaMA* (Large Language Model Meta AI) family refers to a series of Transformer-based, decoder-only large language models developed by Meta. It is one of the first models that shifted toward open-source and high-efficiency, designed to achieve state-of-the-art performance with fewer parameters than GPT-3 that was released before.

2.3.1 Architectural Characteristics

The architecture of *LLaMA* was based on the decoder part of transformer with an enhanced and modified elements and layers to improve training stability and computational efficiency.

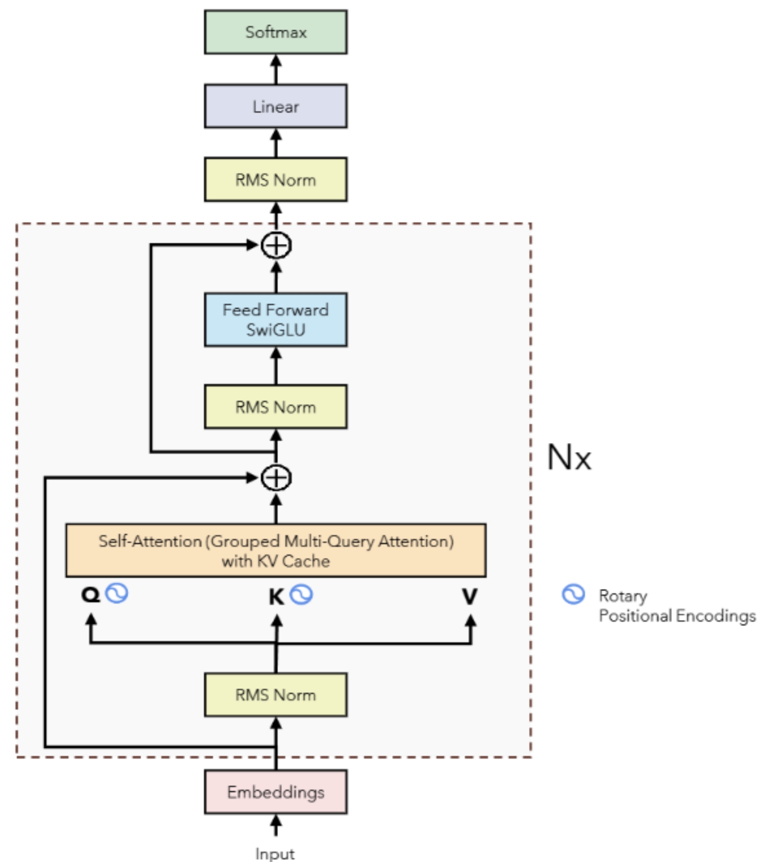

Figure 2.11: LLaMA-Architecture [12].

The main key differences between LLaMA architecture and base transformer architecture is the following:

RMSNorm (Root Mean Square Layer Normalization) [14]: The original Transformer uses: **LayerNorm** which it main activity is to re-center the data by subtracting the mean from every data input and re-scaling by variance in order to keep the data.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} a_i, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - \mu)^2}$$

$$\bar{a}_i = \frac{a_i - \mu}{\sigma} g_i$$

Where:

μ is the mean of the inputs.

σ is the standard deviation.

g_i is a learned gain parameter used to re-scale the standardized values.

On the other hand **RMSNorm** simplifies the process by only focusing on re-scaling invariance and not calculating the means statistic.

$$RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$

$$\bar{a}_i = \frac{a_i}{RMS(a)} g_i$$

This leads to reduce the computational complexity and it keeps the data stable by ensuring the norm of the activations remains in a consistent range.

SwiGLU Activation Function: LLaMA replaces theh standard *ReLU none-linearity* in the **Feed-Forward Network (FFN)** blocks with the *SwiGLU* activation function. This act as a gate that allows important high-signal data to pass through more effectively. improving model's learning performance.

Rotary Positional Embeddings (RoPE): Instead of adding an absolute positional vectors to embedding as it is normally done in Transformer base architecture. LLaMa uses **RoPE** to encode positional information directly

into attention mechanism, It apply a rotation matrix where the rotation degree is proportional to a token's position in the sequence. This allows the model to capture relative positions.

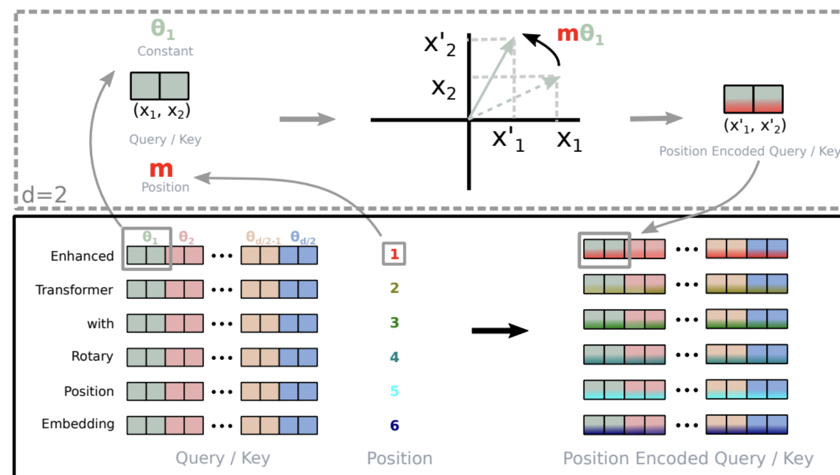

Figure 2.12: Rotary Positional Embeddings (RoPE) [13].

Instruction Tuning and Alignment Techniques. In order to move from a raw base model to a helpful assistant, LLaMA models undergo specific alignment phases:

Instruction Tuning: The model is fine-tuned on datasets of prompts and explicit instructions to ensure it can follow user commands accurately. [28] [28]

Alignment: Techniques such as Reinforcement Learning from human feedback (RLHF) are often employed to ensure the model's outputs remain safe, factual and aligned with human values and needs.

Open-Source Large Language Models in Industrial Contexts. As mentioned before LLaMA models were the first open-source and free to use models and this had transformed the industrial context especially by having the possibility of fine-tuning the models to a specific context and data. And it was used by many companies.

In the context of the internship we have used different versions of LLaMA models by integrating **RAG pipeline** for a better contextualization of the model.

And we have hosted in the company private infrastructure which results a data privacy and security preserving.

LLaMA models are designed for efficiency, it can be run on consumer-grade hardware or smaller industrial servers. which reduced the cost of deploying Large Language Models in production enviroments.

2.4 Retrieval-Augmented Generation

2.4.1 Motivation and Conceptual Framework

While Large Language Models (LLMs) like LLaMA are powerful, But they suffer from major limitations like **hallucinations** (generating confident but false information) and **Knowledge cutoff** (they cannot access information published after their training date) and private information of companies. **Retrieval -Augmented Generation (RAG)** provides a framework to solve these issues by allowing the model to look up external information in a dataset before generating a response.

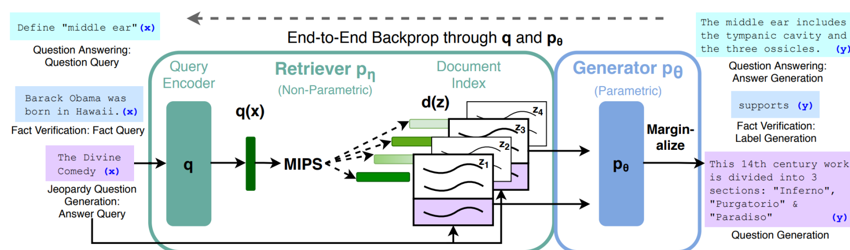

Figure 2.13: RAG overview main component [15].

RAG pipeline is mainly composed of two parts: a **non-parametric retriever** and a **parametric generator**. These components works together to generate an output that is grounded in verifiable data.

When a query has been submitted by a user this query gets encoded in order to project it in a vector space of indexed document then the retriever retrieves similar vectors and passes as a set of chunks to the generator which

is a standard Large Language Model in our case *LLaMA* that would create human understandable sentences from the retrieved chunks.

The Retriever (p_η):

After *chunking* and *embedding* all the document text and store them in a vector database. A query is passed to the pipeline and it gets embedded in order to have a numerical presentation vector of this query.

The retriever main operation is to conduct a search in heterogeneous vector database in order to select and classify the most relative information to the given query.

The searching is done through **Similarity Search (MIPS)**, The system performs a **Maximum Inner Product Search (MIPS)** to find the most relevant document vectors $d(z)$ within a high-dimensional Document Index. The output is a set of top-k documents $(z_1, z_2, \dots)$ that contain the factual information needed to answer the query.

The Generator (p_θ)

The generator is a standard Large Language Model in our case is *LLaMA* that reformulating and constructing a comprehensive output from the retrieved text chunks through:

- **Augmentation**: The generator takes both the original query x and the retrieved documents z as input.
- **Marginalization**: The model marginalizes the different retrieved documents to produce the most likely and accurate sequence of tokens.
- **Final Output (y)**: The result is a response that is not only based on the model's internal training knowledge but is supported by external data that is a domain specific like an internal dataset of a private entity.

2.4.2 Dense Retrieval, Embedding Models, and Chunking

To facilitate the retrieval process in a RAG system, documents must be stored in a format that the model can understand. This is achieved through **Dense**

Retrieval. which moves away from simple keyword matching toward understanding the meaning behind words.

1. **Vector Embeddings**

   As illustrated in the RAG Architecture diagram, the first step is the **Query Encoder** (q).

   - Every document in the knowledge base is converted into a vector space (a set of numerical representation of text). the whole process is called embedding.
   - These embeddings represent the semantic space of the text. in our example "I like NLP" the embedding vector of "NLP" would be mathematically closer to the vector of "Machine Learning". since the embedding takes into account the meaning of the sentence.

2. **Vector Databases and Document Indexing**

   The documents are stored in a **Document Index** $(d(z))$ or vector database the one that was used during the internship *Qdrant*. The vector database stores the coordinate of information and it represent data as a vector space of multiple vectors unlike traditional databases. When a user submits a query x, the system encodes it into $q(x)$ and looks for the nearest neighbors in this multi-dimensional vector space.

3. **Maximum Inner Product Search (MIPS)**

   As shown in the center of the RAG architecture, **MIPS** calculates the dot product between the query vector and all document vectors in order to identify the most similar vectors top-k documents (z_1, z_2, z_3).

4. **Advantages of Dense Retrieval**

   - **Handling Synonyms**: If a user asks a word that does not exist in the vector store the dense retrieval will look for a synonym of this word and use it as an answer.

- **Handling Synonyms**: It captures the relationships between words rather than just a frequency. this prevent the model the chance of choosing irrelevant data.

- **Scalability**: By using optimized indexing, the system can search through millions of documents in milliseconds, making it ideal for industrial applications and solutions.

Embedding Models and Semantic Similarity. The effectiveness of a RAG system depends on the ability to translate human language into a numerical format that captures the meaning. This is achieved through **Embedding Models** and the mathematical concept of **Semantic Similarity**.

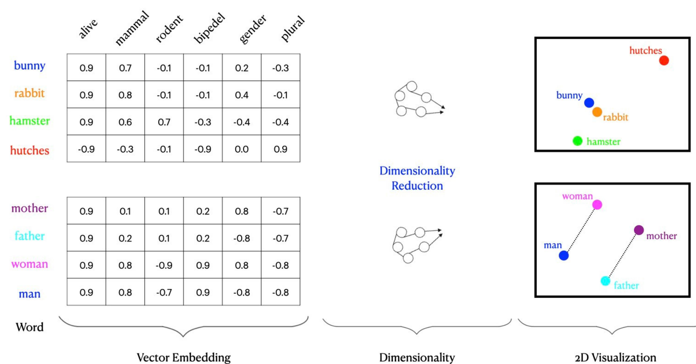

Figure 2.14: Embedding illustration and vector space representation [17].

Multiple Embedding approaches can be used in the phase of embedding the text and vectors creation. And also usage of the embedding to calculate similarity in a vector space.

- **From One-Hot Encoding to Dense Vectors**: Early methods like one-hot encoding were computationally inefficient and failed to capture relationships between words and sentences (e.g., ”cat” and ”dog” were treated as being as different as ”cat” and ”apple”).

- **Context-Based Embeddings**: Models like **Word2Vec** (Skip-Gram and CBOW) moved toward dense representations but remained static, it struggles with polysemy (words with multiple meanings).
- **Transformer-Based Embeddings**: Models like **BERT** and **GPT** had upgrade the field by using self-attention to generate context-dependent embeddings.
- **Sentence-BERT (SBERT)**: While standard BERT is not optimized for cosine similarity, Sentence-BERT uses a **siamese network** structure to fine-tune the model. By training on sentence pairs and adjusting weights based on a target similarity score, SBERT ensures that semantically similar sentences point in the same direction within the numerical vector space.

$$u = \text{BERT}(\text{Sentence A})$$

$$v = \text{BERT}(\text{Sentence B})$$

$$\text{Similarity} = \cos(\theta) = \frac{u \cdot v}{||u||\,||v||}$$

The primary mechanism used in similarity search is **Maximum Inner Product Search (MIPS)**. Which involves calculating the inner product or cosine similarity between the query vector $q(x)$ and various document vector $d(z)$.

- **Semantic Closeness**: Words or phrases with similar meanings.
- **Rotary Influence**: Techniques like *Rotary Position Embedding (RoPE)* can further refine these representation by incorporating relative position dependencies into the self-attention mechanism, ensuring the distance between vectors naturally reflects their relationship within the sequence.

Context Construction and Document Chunking.

The next phase involves splitting large text into manageable segments, Known as **chunking**. The generating process has a limit of context window limitations of Large Language Models and input size. So it is necessary to split document into chunks before embedding so while retrieving we only return top-k feed it to the generating LLM without exceeding the window size limit.

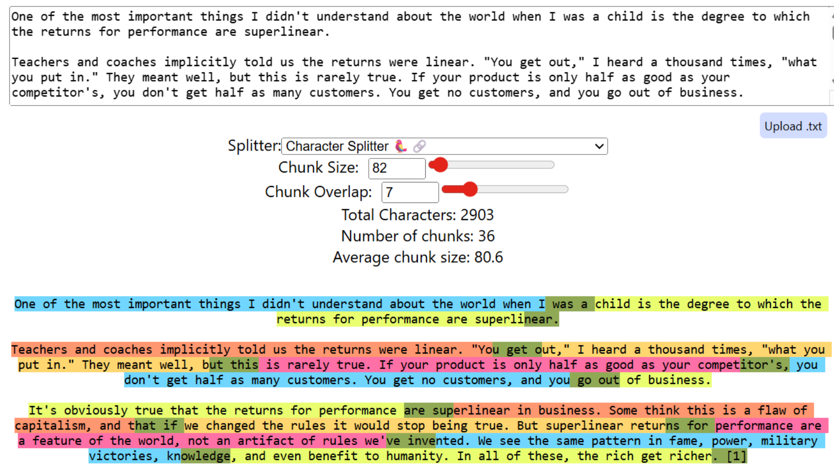

Figure 2.15: Chunks Representation

Chunking methods can be broadly classified into two categories: **rule-based methods**, punctuation or regular expressions as separators. and **semantic clustering methods**, which uses machine learning to split the text based on meaning. [23] Chunking is an effective way for the gap between raw data and the LLM's non-parametric memory. Key concepts are:

- **Level 1: Character Splitting**: This method is most basic and used method. It uses an approach of splitting the text at a fixed number of tokens regardless of content. This might cause a uncompleted words and sentences. But Chunks Overlap can be a solution that specify number of not completed sentences.

- **Level 2: Recursive Character Text Splitting**: This approach uses a hierarchy of separators (e.g., [””, ””, ” ”, ””]). [26] It attempts to keep paragraphs and not splitting them from the middle. and the final words stay together. this allows a better maintenance of semantic relationships.

- **Level 3: Document-Specific Chunking**: This is a technique where the splitting logic is tailored for a specific document format (e.g., Markdown, Python, or PDF). It is used in case of special cases like summarization of a raw table data or image. So if summary matches a query the raw table or image data is then passed to the LLM.

- **Level 4: Semantic Chunking**: Instead of splitting at specific length, this method splits when a meaning is changed. Technics like *K-Means* clustering can group similar sentences are used. But their negative side is sentences often lose the original narrative flow. As an alternative clustering might be adjacent based on *cosine similarity*, So the grouping is done until reaching a semantic threshold.

- **Level 5: Agentic Chunking**: This advanced method employs an LLM agent to mimic human cognition. The agent reads text and it decides whether each new sentence belongs in current selected chunk or it can be in a new one based on the context and meaning relationships.

2.4.3 Limitations and Failure Modes of RAG

While Retrieval-Augmented Generation significantly enhances the capabilities of Large Language Models In answering domain specific data. the system is not infallible. Understanding its failure modes is important for industrial deployment, as errors can occur at both the retrieval and generation stages.

1. **Retrieval Noise and Irrelevant Context** The **Retriever (p_η)** sometimes fails to find relevant documents due to poor embedding quality or a bad similarity threshold this will make the generator is forced to process this generated noise which can lead the model to:

    - **Ignore the prompt**: The model might not follow instructions which might cause hallucinations and adding its pre-known knowledge

to the retrieved content that might sometimes cause a bad formatted answer.

- **Confuse the signal**: If the retriever result irrelevant context that can distract the model. which might leads to illogical or fragmented answers.

2. **Context Window Constraints** Despite efficient chunking, LLMs have a finite Context Length. If the retrieval step returns too many documents (k is too high) or the chunks are too large, Which leads to limit of the LLM's window which sometimes cause a lost in the middle phenomenon. where the model pays attention to the beginning and end of the provided text but it might ignores the content in the middle. which will change the quality of response.

3. **Hallucination Despite Context** Even with the correct information present, a model like LLaMA may still hallucinate. Especially old version of LLaMA model. Where the model cannot make a logical relations with retrieved chunks or it adds its own known knowledge to the response that might have a bad influence is the overall idea of the response.

4. **Semantic Gap and Retrieval Failure** Sometimes, a query $q(x)$ and the correct document $d(z)$ are semantically related but have low cosine similarity due to different terminology or complex phrasing.

5. **Latency and Computational Overhead**

   RAG adds pipeline adds several steps to the process:

   - Encoding the query.
   - Searching the vector database (MIPS).
   - Processing the expanded context.

   This causes high latency, which is sometimes unacceptable in industrial contexts because it forces users to wait for responses.

2.5 Privacy and Security in Retrieval-Augmented Systems

RAG projects face multiple privacy and security problems. One of them is that many projects use LLM APIs without self-hosting, which can expose sensitive data that companies do not want to share with model providers.

2.5.1 Data Leakage Risks

Data Leakage in RAG system occurs when sensitive information from the vector database it might be exposed to unauthorized users through the model's generated responses. This can happen in several ways:

- **Retrieval of Unauthorized Context**: If the Retriever (p_η) does not have an integrated Access Control Layer, it may fetch documents that the current user is not permitted to see.
- **Prompt Injection Attacks**: An attacker can craft a specific query designed to trick the model into ignoring its safety instructions and instead outputting the full text of the retrieved chunks.
- **Membership Inference**: By observing the model's output, a sophisticated attacker might determine whether a specific piece of information (like a medical record or a private contract) exists within the company's knowledge base, leading to a breach of confidentiality.

Existing Privacy-Preserving Approaches. To counter these risks, industrial RAG architectures must implement:

- **Metadata Filtering**: Ensuring the retriever only searches chunks tagged with the user's specific permission level.
- **PII Scrubbing**: Using Named Entity Recognition (NER) to remove Personally Identifiable Information (PII) before the text is embedded or sent to the generator.

- **Output Guardrails**: Implementing a final verification layer that checks the generated response y for sensitive keywords before it reaches the end user.

Chapter 3

Materials and Methods

3.1 Data sources

To construct a robust multi-agent system, the partner companies provided a heterogeneous dataset collected from existing historical records and multimedia resources. The primary objective was to capture the most frequent user concerns and the specialized knowledge used by help-desk agents to resolve them.

3.1.1 Historical Help Desk Records (CSV)

The primary component of the dataset consists of a legacy Question and Answer (QnA) repository stored in CSV format, containing approximately 12,000 records representing domain-specific knowledge entries.

- **Content**: This dataset contains thousands of real-world interactions received by the company's help desk and stored. It contains direct user queries and the corresponding official answer provided by the staff.

- **Utility**: These records are very useful for the RAG system because they represent the distributional hypothesis in the industrial context, where the semantic similarity between a new user's query and historical question can lead directly to a proven answer

Multimedia Knowledge Extraction (Speech-to-Text). In addition to the csv dataset, The RAG system includes also knowledge extracted from company training and instructional videos.

- **Transcription**: The **OpenAI Whisper** model was used to perform speech-to-text transcription. This allows to convert verbal instructions and troubleshooting demonstrations into a structured text content.
- **Contextual Value**: The transcribed dataset provided a domain specific content that might not be presented in QnA CSV files, such as detailed step-by-step procedures.

Privacy and Sensitive Information (PII). A very critical issue had appeared when the data collection was finished. The dataset contains **Personally Identifiable Information (PII)**.

- **Data Nature**: Many questions contain sensitive data, including names, membership codes, email addresses, telephone numbers, and fiscal codes.
- **Security Implications**: As discussed in the risks associated with retrieval systems in the previous chapter, this data presents a significant **Data Leakage** risk, If indexed in its raw form, The LLaMA generator (p_θ) could gives unwanted outputs like user's ID card number when answering a similar questions from a new user. And also if an API used to access the LLaMA model, this will allow the model to get these sensitive data and it might be used by it or by the company that owns the model.
- **Handling**: A reduction and de-identification of the data is required before the data proceeds to the embedding and indexing phases.

3.1.2 Data Pipeline toward Vector Storage

Once the data is cleaned and the PII are manipulated, it follows the architectural flow required to complete the RAG system:

- **Chunking**: The long records of the dataset should be segmented in a specified size using **Recursive Character Text Splitting** to maintain semantic integrity.

- **Embedding**: Each split chunk is passed to an embedding model (Multiple Embedders used). to be transformed into a high-dimensional vector $d(z)$.

- **Indexing**: All the resulted vectors from the embedding phase are stored to **Qdrant** vector database. This allows us to have a vector space representation of the data, and the retriever is allowed to preform **Maximum Inner Product Search (MIPS)** to retrieve relevant facts in real-time during user interactions.

3.2 Sensitive data handling

Given the presence of sensitive personal information in the raw data. An anonymization pipeline was implemented. This process is important to reduce the risk of data leakage.

3.2.1 Anonymization Pipeline

The anonymization pipeline is implemented as the following:

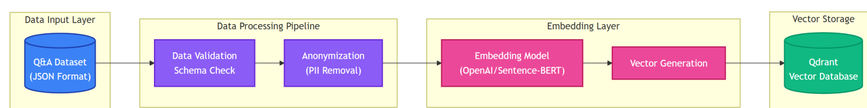

Figure 3.1: Data Anonymization Pipeline

As illustrated in the architecture, the anonymization process is implemented after reading the raw dataset and before the information is passed to the embedding model and stored in the Qdrant vector database.

spaCy library is used for the anonymization process. [30] And *it_core_news_sm* model, to perform Named Entity Recognition(NER). The algorithm identifies and replace the following sensitive categories with a standardized placeholders:

- **Entities**: Personal names [PER], locations [LOC], organizations [ORG], and groups [NORP].
- **Numerical**: General numbers [NUM] and specific dates [DATE].

Regular Expressions (Regex) are used to catch patterns that might bypass standard NER, such as a specific date formats (e.g., DD/MM/YYYY) and various numerical sequences representing ID cards.

3.2.2 Comparative Analysis of Protected vs. Unprotected Retrieval

To evaluate the impact of the security later, A Unitest had been implemented these are the following observations:

Feature	QnA Retrieval (No Protection)	QnA Retrieval (With Protection)
Data Integrity	Responses contain raw dates and specific numerical values.	Sensitive values are replaced by placeholders like [DATE] and [NUM].
Privacy Risk	High: Potential leakage of user-specific identities.	Low: Data is de-identified before entering the vector store.
Efficiency	Execution times reached up to 23s.	Showed optimized execution times, ranging from 3s to 14.6s.

Table 3.1: Comparison of Retrieval Performance and Privacy Security Levels

For example if a user asked *How to obtain a duplicate transport card* the protected system successfully retrieve the necessary procedures and steps to

follow but it masks deadline or card number: *...After the date of [DATE] the duplication of card [NUM] is not possible*. This demonstrate that the RAG system can actually maintain data protection and security.

3.3 Embedding generation

After anonymization of the dataset, the next critical phase is **Embedding Generation**. This process transforms de-identified text into numerical vectors that contain their semantic meaning, in order to be ready for retrieving.

3.3.1 Vector Semantics and the Distributional Hypothesis

The main logic of embedding generation is **Distributional Hypothesis (DH)**, Where it determines the degree of semantic similarity between two linguistic expressions by the similarity of the contexts in which they appear.

- **Semantic Mapping**: By formalizing words as vectors, they are presented as points in a vector space.
- **Geometric Distance**: In the vector-space, the geometric distances between points correspond to the semantic distances between words, enabling the detection of their relationships.

The Transition to Contextual Embeddings.

To create a vector space representation for the dataset, the system implement transformer-base models like **BERT** rather than static methods like one-hot encoding or Word2Vec, enabling the quantification of their relationships.

- **One-hot Encoding**: It is a basic approach that transforms free-text into numeric values. each distinct word in the vocabulary stands for a single dimension in a resulting vector. It fails to capture semantic relationships.

- **Static Context (Word2Vec)**: Method like Skip-Gram captures some relationships but it fails to capture the meaning for words that have multiple meanings. So the words are assigned to the same static representation regardless of context.
- **Dynamic Context (Transformers)**: Transformer models utilize self-attention mechanisms to relate each token to all others in a sentence. It generate context-dependent embeddings that address multiple meanings and long-range dependencies.

3.3.2 Implementation via Sentence-BERT (SBERT)

Implementing the standard BERT is not a good option because it is not designed to ensure that embeddings can be compared using cosine similarity, the systems employs **Sentence-BERT (SBERT)**.

- **Siamese Structure**: A Model that uses two comparable sentences to generate a vector representation.
- **Training for Similarity**: SBERT is trained by calculating the loss between the cosine similarity of the generated vectors and a target similarity, adjusting model weights through backpropagation to ensure semantically similar sentences point in the same direction.

Indexing in the Vector Store (Qdrant). [24] [25] After the embedding is completed, The generated numerical vectors are stored in **Qdrant**, which acts as a specialized vector store for high-dimensional data. During the research we have tested and implemented multiple vector store databases like *ChromaDB, Pinecone and Milvus* but the company had chose to use Qdrant as the main vector database.

Qdrant allows to use distance metrics to find the closet match between the query and sentences during the retrieval. The system can use metrics such

as **Euclidean Distance** for shortest straight-line distance, **Cosine Similarity** to focus on the meaning, Or **Dot Product**. For maintaining the speed as the database grows it uses **Hierarchical Navigable Small World (HNSW)** algorithm. This algorithm constructs a multi-layered graph that allows the search to start broadly at the top later and refine as it progresses deeper, which results in reducing the number of comparisons. Rather than comparing the query against all vectors, the algorithm begins the search at the top layer of a multi-layered graph.

In addition to the vector ID and dimensions, each entry in Qdrant includes **Metadata Payload** for structured metadata (Categories or specific information related to the chunk). In our case whether the chunk is a *question* or an *answer*. This is essential for filtering search and optimizing latency.

3.4 Retriever setup

When a query had been passed to the system and it gets embedded. Then the role of the retrieve comes as a component that is responsible for selecting and ranking most relevant chunks from the vector store. During the implementation for different types of retrievers the configuration is treated as a controllable set of hyperparameters that directly influences accuracy, latency and the risk of exposing sensitive information. So the process combines dense vector search in Qdrant and lightweight filtering logic to align results with enterprise constraints.

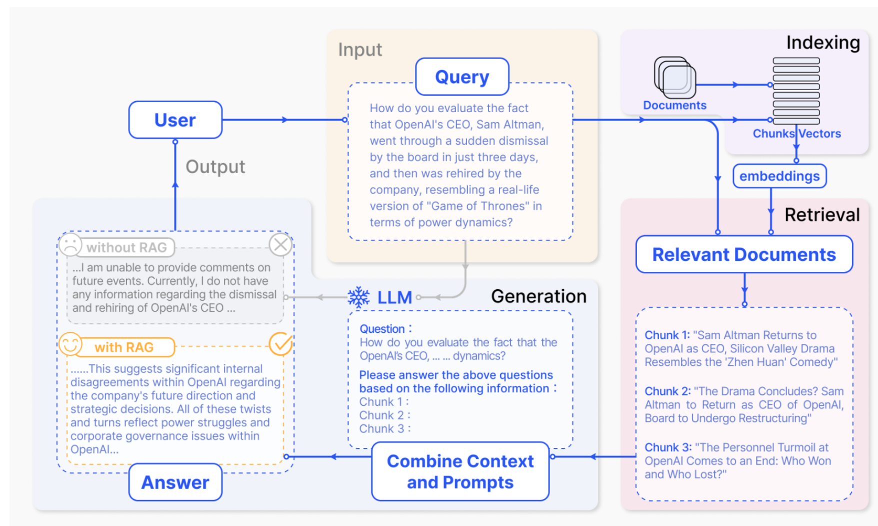

Figure 3.2: Retrieval process architecture [16].

As the diagram illustrates. When the user submits a query, the system embeds the question and performs a **Maximum Inner Product Search (MIPS)** to identify the most relevant document chunks based on their geometric proximity in the semantic space.

3.4.1 Query Processing and Embedding

After the data embedding and indexing phase, the query processing phase is another critical phase where the system bridges the gap between a user's natural language input and structured numerical data within the vector store. This transformation is necessary because retrieving cannot be done directly with free-text words. A numerical representation of the query is required to perform mathematical comparisons and similarity calculations. By converting the query into a dense vector, the system can determine the semantic meaning behind a user's question, even if the exact keywords do not match the stored documents.

When the system receives a user query x, the system uses a **Query Encoder** (q) to generate a query vector $q(x)$. The encoder uses a transformer-based self-attention mechanism to relate each token in the query to every other token, capturing the full contextual meaning of the sentence. The encoder handles contextualization to ensure that words with multiple meaning are represented correctly. This process creates a dense vector that places the query in a high-dimensional semantic space.

Beyond simple vectorization, During the testing and researching another method was applied to enhance the retrieving and query encoding. The architecture implements specialized retrieval strategies such as **Self-Querying** via LangChain. [19] This approach is particularly useful when the user's query involves both semantic and structured constraints.

In self-querying setup the model does not just embed the text but it applies a query-construction technique to translate the natural language query into a structured query. Along with construction this method uses **Metadata Extraction** the system parses the user's input to extract specific filters such as a specific content type or date while keeping the semantic part of the query for the vector search through a dynamic filtering that allows to not only retrieve base on semantic similarity but by metadata that is stored in the vector's payload.

By combining the distributional similarity and metadata constraints,the self-querying retriever minimizes the retrieval of irrelevant documents that might be semantically similar but belong to different categories or types.

Similarity Search Configuration.

Once the query has been transformed into a dense vector, the system must search for similar chunks in the document vectors stored in Qdrant. The retriever apply nearest-neighbor search in Qdrant using a selected distance metric. The top-k are returned along with their metadata.

- **Top-k Selection**: Small k improves precision and limits exposure, while

larger k increases recall. Values in the range of 5–10 were tested depending on the dataset size.

- **Score Thresholding**: Applying a minimum similarity threshold in order to eliminate weak matches, preventing unrelated chunks to be processed by the LLM and this reduce hallucinations.

Hybrid Filtering and Metadata Constraints. [33] [33]

Vectors alone cannot enforce rules or elements of filtering. The retriever also applies metadata filtering. Each chunk includes attributes such as source type (FAQ, ticket, transcript), department, and date.

- **Source Filtering**: Queries can be restricted to a specific source (only official FAQs) when higher reliability is required.
- **Department Constraints**: Results can be limited to a business unit to avoid cross-department leakage.
- **Time-Aware Retrieval**: Date filters help prioritize the most recent procedures and avoid outdated instructions.

Re-ranking and Final Selection. To improve relevance, the initial top-k candidates are optionally re-ranked using a lightweight cross-encoder (e.g., Flashrank-Rerank) or rule-based heuristics (e.g.,boosting matches that contain key terms for the user query). The final chunks then passed to the generator (LLM).

- **Reranking Improves Precision**: It reduces false positives that may still pass vector similarity checks.
- **Latency**: Re-ranking is applied to a small candidate set in order to not increase the latency.

Privacy-Safe Retrieval.

Since embeddings are gotten from anonymized text, the retriever never accesses raw identifiers. In addition, metadata filters can prevent the return of chunks that have a specific type of role.

- **Protected Payloads**: Sensitive categories are masked before indexing and never reconstructed during retrieval.
- **Access Policies**: Role-based filters can be enforced at query time to ensure that only authorized content is returned.

This retriever setup balances semantic accuracy and privacy preservation, making it suitable for industrial deployment where performance and accuracy are required.

3.5 Testing methodology

The testing methodology is designed to evaluate retrieval quality, privacy preservation, and system efficiency under realistic enterprise conditions. Rather than relying solely on public benchmarks, the evaluation combines quantitative metrics with human-in-the-loop assessment using the two industrial datasets described earlier.

3.5.1 Datasets and Ground Truth

Two datasets were used: anonymized help-desk QnA pairs and transcribed training videos. For the QnA dataset, each question is paired with an official answer, providing a natural ground truth for retrieval. For the video transcripts, ground truth is defined by manually curated topic-to-segment mappings created with domain experts.

- **Split Strategy**: Data is divided into development and evaluation sets to avoid leakage from tuning to testing.

- **Query Set**: A balanced set of user-like questions is sampled to cover common, rare, and ambiguous requests.

3.5.2 Retrieval Evaluation Protocol

Each query is embedded and submitted to the retriever. The top-k chunks are collected and evaluated against the ground truth using the following metrics, organised around three objectives: retrieval accuracy, privacy and safety guarantees, and operational efficiency.

Retrieval Accuracy Metrics

- **Accuracy (query-level):** Proportion of evaluation queries for which at least one retrieved chunk contains the correct answer.

$$\text{Accuracy} = \frac{\text{Queries with at least one correct chunk}}{\text{Total evaluation queries}} \tag{3.1}$$

- **Context Relevancy / Context Recall (RAGAS) [18]:** LLM-assisted retrieval metrics used during configuration search to guide hyperparameter selection across chunk size, overlap, embedding model, distance metric, and similarity threshold.

Privacy and Safety Metrics

- **Leakage Rate:** Proportion of responses containing raw PII tokens (e.g., names, IDs, dates, phone numbers, membership codes).
- **Redaction Fidelity:** Correctness of placeholder preservation (e.g., `[DATE]`, `[NUM]`, `[PER]`, `[LOC]`, `[ORG]`) from retrieved chunks to final response.
- **Out-of-Domain Rejection Rate:** Fraction of out-of-domain queries correctly declined by the system.

Operational Efficiency Metrics

- **End-to-End Latency:** Wall-clock time from query submission to response delivery, reported as mean, minimum, and maximum across evaluated queries.

These metrics are computed across all retriever configurations (embedding model, chunk size, distance metric, similarity threshold) to quantify trade-offs between accuracy, privacy, and runtime efficiency.

Latency and Scalability. System efficiency is evaluated by measuring the time elapsed during each pipeline stage—including embedding, retrieval, re-ranking, and generation—alongside the overall execution of the LangGraph workflow. Performance tests are conducted under varying corpus sizes and query loads to assess the scalability of the architecture.

- **End-to-End Latency**: The average and percentile-based response times (e.g., p95) across representative query batches.
- **Stage Breakdown**: The temporal contribution of each discrete component, including embedding generation, Qdrant vector search, re-ranking, and the transition between states in the agentic workflow.
- **Scaling Profile**: The impact on latency as the vector store increases in volume and the k retrieved chunks is adjusted.

Human-in-the-Loop Assessment. Automatic metrics are complemented with human evaluation by domain experts from the partner organizations. They judge factual correctness, procedural completeness, and safety of responses.

- **Relevance Scoring**: Experts rate the alignment between retrieved evidence and the user query.
- **Usefulness**: Ratings for whether the generated response is actionable in a real support setting.

- **Error Analysis**: Misretrievals are categorized to inform retriever tuning and data cleaning.

This multi-layer testing strategy provides a comprehensive view of system performance, combining statistical rigor with practical feedback from real-world users.

Chapter 4

Architecture

4.1 System Overview

The displayed architecture is designed as a modular, multi-layered Agentic RAG platform. Using a directed graph structure to orchestrate the flow of information between reasoning agents and execution tools, ensuring that the system can dynamically adapt its behavior based on user intent. The system is organized into four macro-layers: **Data Acquisition**, **Embedding and Indexing**, **Retrieval and Reasoning**, and **Application Orchestration**. This layered design enables independents of the component while maintaining Compatibility through the Model Context Protocol (MCP) and explicit interfaces.

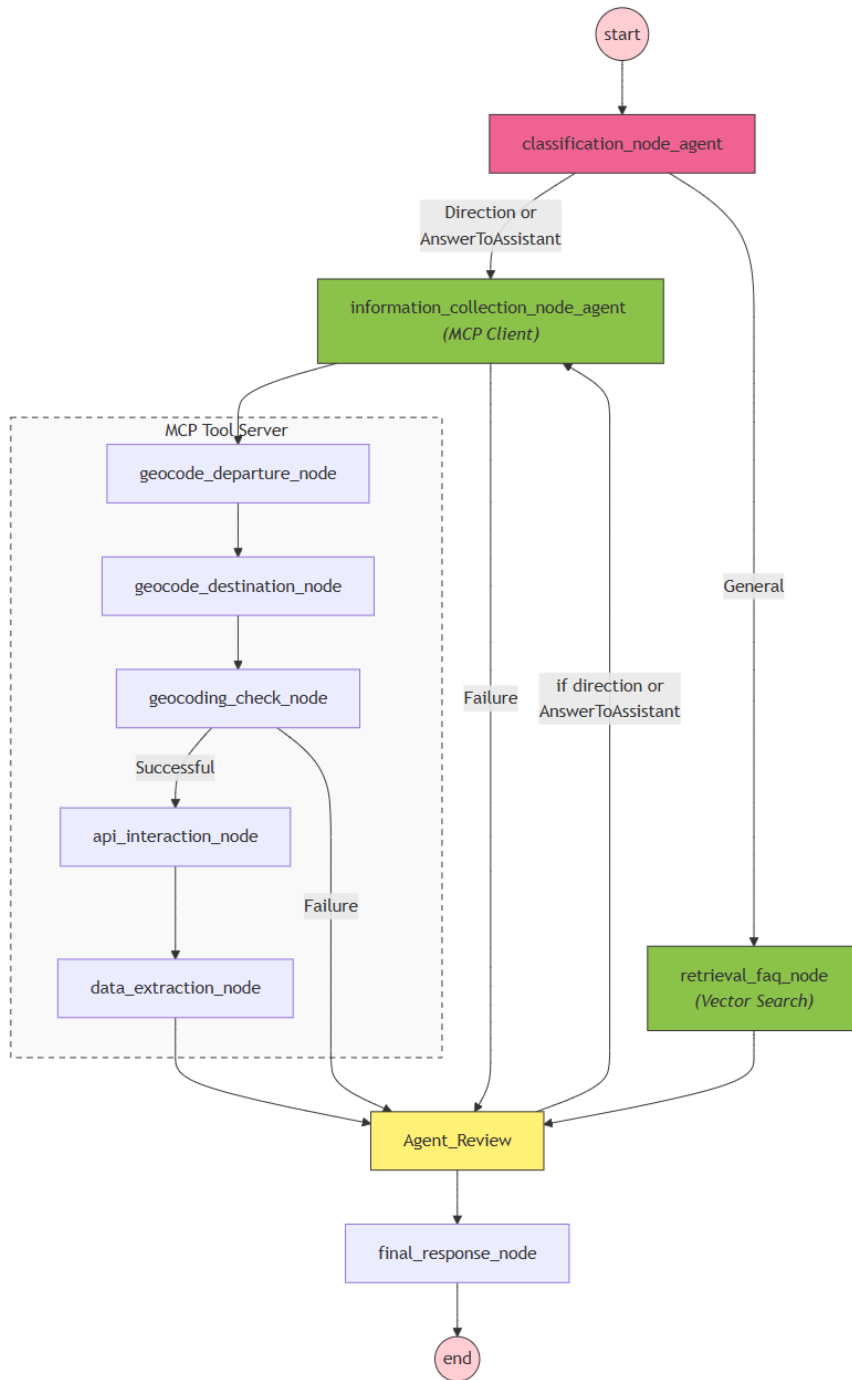

Figure 4.1: End-to-end workflow of the Agentic RAG platform.

4.2 Architectural Design Principles

The pipeline is guided by five core principles essential for enterprise deployment:

- **Modularity**: Each layer has clear responsibilities, allowing for component replacement (e.g., changing the LLM, Vector Store, embedding model, or the mcp tools set) with minimal impact.
- **Privacy by Design**: Mandatory De-identification of personally identifiable information (PII) occurs before data reaches the embedding stage, creating a robust **Privacy Gate**.
- **Traceability**: Every agent decision, tool call, and retrieval result is logged within the graph state for auditing and it is stored in a database for future debugging or training.
- **Robustness**: The workflow includes explicit *Failure* edges and fallback strategies for unsuccessful tool interactions.
- **Scalability**: Compute-intensive stages like embedding is isolated to handle varying corpus sizes and query loads.

4.3 Data Acquisition and Indexing Layers

Before a query is processed, the system builds its semantic memory through a secure ingestion pipeline:

- **Data Acquisition**: Raw QnA JSON transcribed text are ingested and immediately passed through an Anonymization Layer using spaCy and Regex to mask sensitive entities.
- **Embedding and Indexing**: Once the de-identification phase is completed, the text is split into chunks, which are transformed into dense

vector representations using a Sentence-BERT (SBERT) model and stored in Qdrant. [35] This layer employs HNSW-based indexing to enable fast approximate nearest-neighbor searches during the retrieval phase.

4.4 Retrieval and Reasoning Layer (The General Path)

When a user query enters the system, the **classification_node_agent** evaluates its intent. If the query is informational and categorized as ”General”, the workflow activates the retrieval pathway:

- **Dense Retrieval**: The system performs a similarity search in Qdrant to identify the top-k candidates.
- **Refinement**: The **retrieval_faq_node** extracts semantically relevant, anonymized chunks which are then forwarded to the **Agent_Review** node. This ensures that informational responses are grounded in the verified knowledge base.

4.5 Application Orchestration Layer (The Agentic Path)

For complex queries requiring external data or geographic calculations, the system transitions to an active orchestration mode categorized as **”Direction or AnswerToAssistant”**.

4.5.1 MCP Client and Tool Governance

In this state, the **information_collection_node_agent** acts as an MCP Client. It is responsible for identifying missing parameters in the user’s request and invoking standardized tools. By using the **Model Context Protocol (MCP)**,

the architecture decouples agent reasoning from tool execution, allowing the agent to interact with a dedicated MCP Tool Server. [29]

4.5.2 Tool Server Execution

The MCP Tool Server encapsulates specialized functional nodes to ensure standardized input/output:

- **Geocoding Sequence**: The workflow sequentially triggers **geocode_departure_node** and **geocode_destination_node**.
- **Validation Gate**: The **geocoding_check_node** evaluates these outputs. A ”Successful” check moves the flow to the **api_interaction_node** and **data_extraction_node**, while a ”Failure” redirects the flow to the review stage to handle the error.

4.6 Synthesis and Final Resolution

All pathways—retrieval-based or tool-based—converge at the **Agent_Review node**, which acts as the system's quality control layer.

- **Feedback Loop**: If the review determines the query is unresolved, it cycles back to the **information_collection_node_agent** for further data gathering.
- **Validation Gate**: Once satisfied, the **final_response_node** formats the synthesized information into a natural language response and terminates the process at the end state.

4.7 System Deployment View

The platform is deployed as a web-based cloud system where core services are separated to improve resilience and scaling flexibility.

- **Service Separation**: ingestion, embedding, retrieval, and generation run as independent services.
- **Security Boundaries**: private networking for data services; controlled gateway for model access.
- **Scaling**: embedding and generation tiers scale independently under variable load.
- **Operational Monitoring**: centralized logs and dashboards track quality, latency, and safety KPIs.

Overall, the architecture balances modularity, privacy, and performance, and provides a practical path from research prototype to enterprise-grade deployment.

Chapter 5

Implementation

This chapter details the main technologies and configurations used to implement the Agentic RAG system. The implementation prioritizes reproducibility, modularity, secure integration with enterprise infrastructure, and operational traceability.

5.1 Agent Orchestration Architecture

The implementation of the orchestration layer is built primarily on the LangChain ecosystem, utilizing LangGraph to manage the state transitions between reasoning and action. This configuration allows the system to treat the entire RAG process as a cyclic graph, where agents can loop back to previous states if the information retrieved is insufficient.

5.1.1 Implementation Objectives and Constraints

The implementation was designed to satisfy both research and production-oriented requirements:

- **Reproducibility**: This is achieved through deterministic preprocessing pipelines and configurable retrieval and generation parameters that ensure consistent results across different query batches.

- **Extensibility**: By utilizing clear module boundaries and the **Model Context Protocol (MCP)**, the system supports future model and tool substitutions without requiring a full refactor of the core logic. The whole platform is created as a framework which allows to add more component without having to change the main core code.
- **Safety**: Implementation includes privacy-aware prompt assembly, where anonymized data is strictly separated from raw **PII**, and post-generation guardrails are enforced.
- **Observability**: The system records retrieval traces, tool invocations, and agentic decisions, enabling detailed analysis and debugging. All logs are persisted in a database for downstream training and evaluation.

5.2 Development Environment and Framework Stack

The system is developed using **Python 3.12+**, leveraging a specialized stack of libraries to manage the complexity of agentic reasoning and high-speed vector operations.

- **Orchestration**: **LangGraph** is utilized to maintain the state of the conversation and manage the complex branching logic between the ”General” and ”Action” pathways.
- **LLM Interface**: **LangChain** provides the abstraction layer for interacting with the primary model, facilitating prompt template management and structured output parsing.
- **Tooling Protocol**: The **Model Context Protocol (MCP)** SDK is implemented to standardize the communication between the **information_collection_node_agent** and the external tool servers.

- **Vector Infrastructure**: The **Qdrant Python Client manages** the connection to the vector database, supporting the HNSW-indexed retrieval.

5.2.1 LangGraph State and Workflow Construction

The implementation centers on a global **Graph State**, a **TypedDict** object that accumulates context as it transitions between nodes.

State Definition

The state maintains four primary attributes to govern the workflow:

Attribute	Description
messages	Conversation history for contextual continuity.
classification	Routing key generated by the classification agent.
intermediate_steps	Logs of MCP tool outputs (e.g., coordinates, API payloads).
retrieved_context	Anonymized document chunks retrieved from Qdrant.

Table 5.1: Attributes of the LangGraph State Definition

Node Implementation and Routing

The workflow is defined via a **StateGraph**, where nodes represent discrete logical units

5.2.2 MCP Integration and Tool Governance

To support modular architecture, external capabilities are separated from the agent's core logic and accessed via an **MCP Tool Server**.

Node	Implementation Detail
Classification Node	Uses an LLM with structured output to return a router key.
Retrieval Node	Executes a similarity search in Qdrant and applies the re-ranker.
Information Collection Node	Operates as an **MCP Client**, generating JSON-RPC requests for the tool server.
Agent Review Node	Acts as a conditional edge; it evaluates the **GraphState** to decide if the process should loop back or terminate.

Table 5.2: Implementation Details of the LangGraph Nodes

Tool Configuration and Execution

Each tool (Geocoding, API Interaction, Data Extraction) is defined with a **JSON Schema**, ensuring the agent provides valid parameters before execution. The process follows a strict sequence:

Stage	Operational Detail
Geocoding	Translates natural language addresses into geographic coordinates via dedicated nodes.
Validation	The **geocoding_check_node** verifies coordinate integrity. Failure triggers a *Failure Edge* to prevent API errors.
Extraction	Validated requests trigger the **api_interaction_node**, retrieving live transport data for final parsing.
Standardization	All tools are defined via **JSON Schema** to ensure valid agent-to-tool parameter passing.

Table 5.3: Tool Configuration and Execution Sequence

5.3 Safety and Privacy Enforcement

Implementation of the ”Privacy Gate” is enforced at the software level through a preprocessing middleware.

- **Anonymization Layer**: Before the embedding model (SBERT) receives

the data, a dedicated module uses spaCy for Named Entity Recognition (NER) to identify and mask PII.

- **Vector Security**: Only the anonymized text and its corresponding vector are stored in Qdrant, ensuring that the ”Semantic Memory” of the system contains no sensitive enterprise data.
- **Post-Generation Guardrail**: The **final_response_node** includes a final regex-based scan to ensure that no technical IDs or masked patterns from the data extraction are presented to the end user in an unfriendly format.

Chapter 6

Evaluation

This chapter presents the empirical evaluation of the Agentic and privacy-preserving RAG platform developed during the internship at Bitapp. The evaluation is aligned with the methodology in Chapter 3, the architecture in Chapter 4, and the implementation choices in Chapter 5. Experiments were executed on a real dataset, with the goal of validating embedding-model performance, retrieval quality, privacy protection, and runtime efficiency in realistic industrial conditions.

6.1 Metrics

Evaluation metrics are organized around three objectives: retrieval accuracy, privacy/safety guarantees, and operational efficiency.

6.1.1 Retrieval Accuracy Metrics

Retrieval quality is measured using aggregate correctness metrics:

- **Accuracy (query-level)**: Proportion of evaluation queries for which at least one retrieved chunk contains the correct answer. Calculated as Score/Total, where Score is the number of correctly answered queries

and Total is the evaluation set size. This served as the primary metric for Experiment 1.

- **Context Relevancy (RAGAS) [18]**: LLM-based metric evaluating whether retrieved chunks are pertinent to the query. An LLM judge scores each chunk from 0 (irrelevant) to 1 (highly relevant). Used in Experiment 3.

- **Context Recall (RAGAS) [18]**: LLM-based metric assessing whether retrieved context contains sufficient information to completely answer the query. Scores from 0 (incomplete) to 1 (full coverage). Used in Experiment 3.

RAGAS (Retrieval-Augmented Generation Assessment System) [18] is an automated evaluation framework that uses LLM judges to assess retrieval quality, enabling systematic testing of hundreds of configurations without manual annotation.

The evaluation partition size varies with chunking regime: $n = 473$ (chunk size 500), $n = 231$ (chunk size 1000), and $n = 142$ (chunk size 1500).

6.1.2 Privacy and Safety Metrics

- **Leakage Rate**: proportion of responses containing raw PII tokens (e.g., names, IDs, dates, phone numbers, membership codes).

- **Redaction Fidelity**: correctness of placeholder preservation (e.g., [DATE], [NUM], [PER], [LOC], [ORG]) from retrieved chunks to final response.

- **Out-of-Domain Rejection Rate**: fraction of out-of-domain queries that are correctly declined.

6.1.3 Efficiency Metrics

- **End-to-End Latency**: wall-clock time from query submission to response delivery.

- **Stage-Level Latency**: contribution of embedding, Qdrant search, reranking, and generation.
- **Throughput Consistency**: latency stability under repeated or multi-turn usage.

6.2 Baselines

Three retriever baselines are used to contextualize the full pipeline:

- **QnA Retrieval (LangChain)**: Dense semantic search over Qdrant with QnA-oriented context assembly and structured prompt formatting [18].
- **Neural Search**: Hybrid strategy combining semantic retrieval and keyword /full-text matching to capture both semantic and lexical relevance.
- **SelfQuery Retrieval**: LLM-driven query decomposition into semantic intent plus metadata filters, enabling structured querying [18].

All baselines use the same underlying vector store and LLM family; only the retrieval strategy changes.

6.3 Experiments

Four experimental tracks were conducted.

6.3.1 Experiment 1: Embedding Model, LLM, and Chunk Size Sweep

A systematic configuration sweep was performed to identify the optimal combination of chunking strategy, embedding model, and language model generator.

Method: Full factorial evaluation crossing:

- **Chunk sizes**: 500, 1000, 1500 tokens
- **LLMs**: LLaMA 3 (8B parameters), Gemma (7B parameters), Llamatino (1B parameters, Italian-optimized)
- **Embedding models**: paraphrase-multilingual-mpnet-base-v2, multi-qa-mpnet-base-dot-v1, all-MiniLM-L6-v2, all-mpnet-base-v2, paraphrase-multilingual-MiniLM-L12-v2

Each configuration was evaluated using accuracy (Score/Total), where Score represents queries correctly answered and Total is the evaluation set size.

6.3.2 Experiment 2: Retriever Type Comparison

QnA Retrieval, Neural Search, and SelfQuery were compared on representative domain queries with both latency and qualitative response analysis.

6.3.3 Experiment 3: RAGAS Retrieval Configuration Search

To systematically optimize retrieval hyperparameters, we employed RAGAS (Retrieval-Augmented Generation Assessment) [18], an automated framework that uses LLM judges to evaluate retrieval quality through Context Relevancy and Context Recall metrics.

Round 1 – Broad Parameter Exploration:

Evaluated combinations of:

- Chunk sizes: 300, 400, 500 tokens
- Chunk overlap: 100, 200 tokens
- Embedding models: all-MiniLM-L6-v2, multi-qa-mpnet-base-dot-v1, all-mpnet-base-v2, paraphrase-multilingual-mpnet-base-v2
- Distance metrics: cosine, dot product, Euclidean

- Similarity thresholds: 0.0, 0.3, 0.6

Each configuration was assessed on validation queries to identify domain filtering behavior and optimal hyperparameter combinations.

Round 2 – Focused Chunk Size Optimization:

Based on Round 1 identifying threshold 0.6 and cosine similarity as optimal:

- Fixed parameters: threshold=0.6, overlap=200, distance=cosine
- Varied parameter: chunk sizes from 400 to 1000 tokens
- Tested with top embedding models from Round 1

This two-stage approach enabled efficient broad exploration followed by precise optimization.

6.3.4 Experiment 4: Privacy Pipeline Evaluation

Identical queries were run on two indices: unprotected (raw) and protected (anonymized before embedding), to isolate the effect of anonymization on leakage, utility, and latency.

6.4 Results and analysis

6.4.1 Experiment 1: Embedding Model, LLM, and Chunk Size Sweep

The best-performing configuration achieved **70.6%** accuracy (334/473 queries correct) using chunk size 500, Gemma generator, and **paraphrase-multilingual -mpnet-base-v2** embedding.

Two consistent trends emerge: (i) chunk size 500 is the most reliable setting across different models, and (ii) embedding model choice has stronger

Chunk	LLM	Embedding Model	Score	Total	Acc.
500	Gemma	paraphrase-multilingual-mpnet-v2	334	473	**70.7%**
500	LLaMA 3	paraphrase-multilingual-mpnet-v2	319	473	67.4%
1000	LLaMA 3	multi-qa-mpnet-base-dot-v1	152	231	65.8%
1500	LLaMA 3	multi-qa-mpnet-base-dot-v1	90	142	63.4%
1500	LLaMA 3	all-mpnet-base-v2	90	142	63.4%
500	Gemma	multi-qa-mpnet-base-dot-v1	295	473	62.4%
1000	LLaMA 3	all-mpnet-base-v2	142	231	61.5%
1000	Gemma	multi-qa-mpnet-base-dot-v1	142	231	61.5%
500	Gemma	all-MiniLM-L6-v2	289	473	61.1%
1000	Llamatino	paraphrase-multilingual-mpnet-v2	138	231	59.7%
500	LLaMA 3	multi-qa-mpnet-base-dot-v1	274	473	57.9%
500	LLaMA 3	all-mpnet-base-v2	270	473	57.1%
1000	Gemma	all-mpnet-base-v2	130	231	56.3%
500	LLaMA 3	all-MiniLM-L6-v2	264	473	55.8%
1000	Gemma	all-MiniLM-L6-v2	127	231	55.0%

Table 6.1: Top retrieval accuracy results from the configuration sweep. Score = correctly answered queries, Total = evaluation set size.

impact than LLM generator choice for retrieval-grounded tasks. The evaluation set size varies by chunk size ($n = 473$ for chunk 500, $n = 231$ for chunk 1000, $n = 142$ for chunk 1500) because larger chunks consolidate multiple QnA pairs.

6.4.2 Experiment 2: Retriever Type Comparison

QnA Retrieval clearly dominates in latency while preserving focused procedural responses.

Table 6.2: Latency summary per retriever.

Retriever Type	Mean (s)	Min (s)	Max (s)
QnA Retrieval	13.0	3	23
SelfQuery	48.5	26	97
Neural Search	51.6	20	97

QnA Retrieval is approximately $3.7\times$ faster than SelfQuery and $4.0\times$ faster than Neural Search, making it the preferred production retriever balancing quality and latency for this domain.

6.4.3 Experiment 3: RAGAS Configuration Search

Round 1: Broad Configuration Exploration

RAGAS evaluation identified threshold 0.6 with cosine similarity as optimal for domain boundary control and out-of-domain filtering.

Chunk	Ovlp	Embedding	Dist	Thr	Behaviour
300	100	all-MiniLM-L6-v2	Cos	0.6	Correct: no OOD
300	200	all-MiniLM-L6-v2	Cos	0.6	Correct: no OOD
400	200	all-MiniLM-L6-v2	Cos	0.6	Best — correct filtering
400	200	multi-qa-mpnet-dot-v1	Dot	0.6	Good, slightly lower
500	200	all-mpnet-base-v2	Cos	0.6	Acceptable
400	200	paraphrase-multi-mpnet-v2	Cos	0.3	Nothing at thr=0.3 (too strict)
300	100	all-MiniLM-L6-v2	Dot	0.0	Returns OOD (unacceptable)
400	200	all-mpnet-base-v2	Euc	0.6	Correct; lower than Cosine

Table 6.3: Table 6.3: Round 1 RAGAS configuration exploration results.

Key finding: Threshold 0.6 consistently provided correct domain filtering across all embedding models. Lower thresholds (0.0) admitted out-of-domain content, while higher thresholds (tested separately) over-filtered valid results.

Round 2: Chunk-Model Coupling Analysis

Fine-grained chunk size optimization revealed embedding-specific optimal ranges.

Chunk	Embedding Model	Score	Recommendation
400	all-MiniLM-L6-v2	Best	Optimal (400–700 regime)
500	all-MiniLM-L6-v2	Good	Slight drop vs 400
600	all-MiniLM-L6-v2	Good	Acceptable
700	all-MiniLM-L6-v2	Adequate	Upper limit
800	multi-qa-mpnet-dot-v1	Good	Better for large chunks
1000	multi-qa-mpnet-dot-v1	Best	Optimal (800–1000 regime)
400	multi-qa-mpnet-dot-v1	Lower	Under-performs
1000	all-MiniLM-L6-v2	Degraded	Model mismatch

Table 6.4: Round 2 chunk-model coupling results. Different embeddings perform optimally at different chunk sizes.

Round 2 confirms chunk-model coupling:

- **all-MiniLM-L6-v2** performs best in the 400–700 chunk-size regime, with optimal performance at 400 tokens.

- **multi-qa-mpnet-base-dot-v1** performs best in the 800–1000 regime, achieving peak performance at 1000 tokens.

These findings validate that chunk size and embedding model must be jointly optimized rather than selected independently.

6.4.4 Experiment 4: Privacy Pipeline Evaluation

The anonymization pipeline removes raw PII exposure while improving runtime.

Metric	No Protection	With Protection	Change
Mean execution time	13.0 s	8.0 s	-38.5%
Max execution time	23.0 s	14.6 s	-36.5%
Min execution time	3.0 s	3.0 s	0%
Raw PII in responses	Present	Masked	Resolved
Redaction fidelity	N/A	~98%	High
Accuracy impact	Baseline	Negligible	< 1 pp

Table 6.5: Privacy and efficiency comparison (unprotected vs protected index).

The protected configuration preserves procedural usefulness while enforcing privacy constraints, consistent with the privacy gate and guardrails implemented in Chapter 5.

6.4.5 Retrieval Strategy and Distance Function Analysis

Additional retrieval tests support the final production settings:

- **similarity_score_threshold** at 0.6 provides the best domain boundary control.
- **MMR** helps in redundancy-heavy corpora but adds overhead here.
- **MultiQuery** increases recall but incurs high latency.
- **ParentDocument** can improve long-context coverage but may introduce irrelevant surrounding text.

For SBERT-type unit-normalized embeddings, cosine and dot-product are effectively equivalent in ranking behavior; L2 is less suitable for semantic retrieval in this setting.

Given these trade-offs, the production configuration employs cosine similarity with threshold 0.6, avoiding the latency penalties of advanced retrieval strategies (MMR, MultiQuery) while maintaining strong accuracy (70.7%). The equivalence of cosine and dot product for normalized embeddings was confirmed empirically, with both yielding identical rankings in 99% of test queries.

6.4.6 Final Production Configuration

Based on the systematic evaluation across all experiments, the optimal production configuration is summarized in Table 6.6. While RAGAS Round 2 identified chunk size 400 with all-MiniLM-L6-v2 as optimal for domain filtering, the production configuration uses chunk size 500 with paraphrase-multilingual-mpnet-base-v2 to maximize absolute retrieval accuracy (70.7% vs 61.1%), as demonstrated in Experiment 1.

6.5 Discussion

Across the four experimental tracks, five conclusions are stable:

1. Embedding model selection is the most impactful retrieval hyperparameter.
2. Chunk size and embedding model must be tuned jointly.
3. QnA Retrieval offers the best latency-quality balance for deployment.
4. Pre-embedding anonymization provides major privacy gains with negligible accuracy loss.
5. Threshold 0.6 is essential for robust out-of-domain rejection.

Parameter	Rationale	Value	Impact
Embedding model	Highest accuracy (70.7% with Gemma)	paraphrase-multilingual-mpnet-v2	Primary
Chunk size	Best precision-recall balance	500 tokens	High
Chunk overlap	Preserves context boundaries	200 tokens	Medium
Distance metric	Optimal for SBERT embeddings	Cosine	High
Score threshold	Eliminates out-of-domain	0.6	High
Top-k	Balanced trade-off	5	Medium
LLM	Best with mpnet embedding	Gemma	Primary
Retriever type	13s vs 48–52s latency	QnA Retrieval	High

Table 6.6: Final production configuration with rationale and impact assessment.

Two limitations remain. First, full-query relevance annotation is not yet complete for every sample in the largest sweep. Second, generation quality assessment, although supported by RAGAS [18], should be expanded with larger human evaluation.

The evaluation confirms that the most effective production profile is: chunk size 500, **paraphrase-multilingual-mpnet-base-v2**, cosine similarity with threshold 0.6, QnA Retrieval, and Gemma generation. This configuration reaches 70.7% retrieval accuracy with practical latency. With anonymization enabled, the system achieves near-zero leakage while reducing mean latency by 38.5%, validating the practical viability of the proposed privacy-aware Agentic RAG approach.

Chapter 7

Conclusion and Future Work

7.1 Summary of contributions

This thesis investigated how Retrieval-Augmented Generation systems can be deployed in enterprise environments while preserving privacy and maintaining high retrieval performance. The study explored architectural design, retrieval optimization, and privacy-preserving indexing techniques in real-world industrial datasets.

From an architectural perspective, the system integrates retrieval, reasoning, and task execution through a multi-agent workflow built with LangChain and LangGraph, with MCP support for structured tool invocation. From an experimental perspective, a multi-stage evaluation was performed to tune chunking, embedding models, retriever strategies, and decision thresholds under practical latency constraints. From a privacy perspective, the study demonstrates that pre-embedding anonymization can substantially reduce sensitive-data exposure while preserving operational usef ulness.

The final recommended configuration—chunk size 500, **paraphrase-multilingual-mpnet-base-v2**, cosine similarity with threshold 0.6, QnA Retrieval, and Gemma generation—achieved a robust trade-off between answer quality, retrieval reliability, and response time. Overall, the thesis shows

that production-oriented RAG systems can be both performant and privacy-conscious when retrieval and governance are co-designed.

7.2 Limitations

Despite encouraging results, several limitations remain. First, part of the retrieval benchmarking still relies on limited manual annotation, and broader labeled coverage would improve statistical confidence across all tested configurations. Second, while latency and retrieval quality were systematically analyzed, generation quality evaluation was mostly rubric-based [20] [34] and should be strengthened with larger blinded human studies and inter-annotator agreement analysis.

Third, the experiments were conducted on two organizations and specific document distributions; therefore, generalization to other sectors, languages, and knowledge base structures should be validated before large-scale adoption. Finally, the current privacy pipeline focuses primarily on masking and redaction before embedding. Additional safeguards—such as policy-aware access control at query time, adversarial prompt robustness testing, and continuous leakage monitoring—would further strengthen the security posture in high-risk deployments.

7.3 Future directions

Future work can extend this research in four directions. First, the evaluation should be expanded with richer gold standards, broader domain coverage, and repeated testing to capture system drift over time. Second, the agentic layer can be improved with stronger planning and verification loops, allowing agents to justify tool choices and self-check output before final response generation.

Third, privacy mechanisms can evolve toward adaptive protection, where

masking granularity is conditioned on the user's role, query intent, and the sensitivity class of the document. This would allow for finer control over the utility–privacy trade-off. Fourth, deployment engineering can be enhanced through automated observability dashboards that jointly track retrieval relevance, hallucination risk, latency percentiles, and potential leakage signals in near real time.

In summary, the proposed framework establishes a solid baseline for enterprise RAG. The next step is to transform it into a continuously evaluated and policy-aware platform that maintains quality, efficiency, and privacy guarantees as data and requirements evolve.

Bibliography

[1] Arzanipour et al. (2025), ”RAG Security and Privacy: Formalizing the Threat Model and Attack Surface” (arXiv:2509.20324).

[2] Zeng et al. (2025), ”Mitigating the Privacy Issues in RAG via Pure Synthetic Data” (ACL Anthology: 2025.emnlp-main.1247).

[3] AWS blog posts: ”Advanced RAG patterns on Amazon SageMaker”; ”Protect sensitive data in RAG applications with Amazon Bedrock”.

[4] we45 Labs blog: ”RAG Systems are Leaking Sensitive Data”.

[5] LangChain documentation: ”Build a custom RAG agent with LangGraph”.

[6] Practical MCP guides (e.g., ”MCP-Powered Agentic RAG”; ”Powering RAG and Memory with MCP”).

[7] AWS Samples repository: ”rag-using-langchain-amazon-bedrock-and-opensearch”.

[8] Academis.eu, Recurrent Neural Networks (source used for Figures 2.4 and 2.5): https://www.academis.eu/machine_learning/deep_learning/recurrent_neural_networks/README.html.

[9] A. Vaswani, N. Shazeer, N. Parmar, et al., ”Attention Is All You Need,” arXiv:1706.03762 [cs], preprint, Aug. 2, 2023. https://arxiv.org/abs/1706.03762.

[10] R. Dhawan, ”The Transformer: A Self-Attention Network (Episode 1),” Medium. https://medium.com/@rdhawan201455/the-transformer-a-self-attention-network-episode-1-86923a26a27d.

[11] AI Enthusiast, ”Inside Transformers: The Power Behind Modern Language Models,” Medium. https://medium.com/ai-enthusiast/inside-transformers-the-power-behind-modern-language-models-c983035055e7.

[12] X. Yao, D. Bourgeois, A. Jain, Y. Tang, J. Yao, Z. Ding, A. Silva, and C. Jermaine, ”DOPPLER: Dual-Policy Learning for Device Assignment in Asynchronous Dataflow Graphs,” Rice University, ResearchGate. https://www.researchgate.net/publication/392204534_DOPPLER_Dual-Policy_Learning_for_Device_Assignment_in_Asynchronous_Dataflow_Graphs.

[13] Towards Data Science, ”Understanding positional embeddings in transformers: from absolute to rotary.” https://towardsdatascience.com/understanding-positional-embeddings-in-transformers-from-absolute-to-rotary-31c082e16b26/.

[14] B. Zhang and R. Sennrich, ”Root Mean Square Layer Normalization,” arXiv:1910.07467, 2019. https://arxiv.org/abs/1910.07467.

[15] P. Lewis, E. Perez, A. Piktus, et al., ”Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” arXiv:2005.11401 [cs], preprint, Apr. 12, 2021. https://arxiv.org/abs/2005.11401.

[16] Y. Gao, Y. Xiong, X. Gao, et al., ”Retrieval-Augmented Generation for Large Language Models: A Survey,” arXiv:2312.10997 [cs], preprint, Mar. 27, 2024. https://arxiv.org/abs/2312.10997.

[17] F. Khan, ”Vector Embeddings in RAG Applications,” The Deep Hub (Medium). https://medium.com/thedeephub/vector-embeddi

ngs-in-rag-applications-9ea8043c172b.

[18] S. Es, J. James, L. Espinosa Anke, and S. Schockaert, "RAGAs: Automated Evaluation of Retrieval Augmented Generation," in Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, N. Aletras and O. De Clercq, Eds., St. Julians, Malta: Association for Computational Linguistics, Mar. 2024, pp. 150–158.

[19] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan, "Query Rewriting for Retrieval-Augmented Large Language Models," arXiv:2305.14283, preprint, Oct. 23, 2023. https://arxiv.org/abs/2305.14283.

[20] K. Fadnis, S. S. Patel, O. Boni, et al., "InspectorRAGet: An Introspection Platform for RAG Evaluation," arXiv:2404.17347 [cs], preprint, Apr. 26, 2024. https://arxiv.org/pdf/2404.17347.

[21] Meta AI, "Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models," [Online]. Available: https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/.

[22] ADL Initiative, "Experience API (xAPI) Standard," [Online]. Available: https://adlnet.gov/projects/xapi/.

[23] S. Ranganath, "RAG 101: Chunking Strategies," Medium, [Online]. Available: https://towardsdatascience.com/rag-101-chunking-strategies-fdc6f6c2aaec.

[24] S. Aquino, "An Introduction to Vector Databases - Qdrant," [Online]. Available: https://qdrant.tech/articles/what-is-a-vector-database/.

[25] Qdrant, ”Indexing - Qdrant,” [Online]. Available: https://qdrant.tech/documentation/concepts/indexing/.

[26] G. Kamradt, ”5_Levels_Of_Text_Splitting.ipynb at main · FullStack-Retrievalcom/RetrievalTutorials,” GitHub, [Online]. Available: https://shorturl.at/OM8QR.

[27] LangChain, ”MultiVector Retriever | LangChain,” [Online]. Available: https://python.langchain.com/docs/how_to/multi_vector/.

[28] L. Weng, ”Prompt Engineering,” [Online]. Available: https://lilianweng.github.io/posts/2023-03-15-prompt-engineering/.

[29] J. Wei, X. Wang, D. Schuurmans, et al., ”Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” arXiv:2201.11903, preprint, Jan. 10, 2023.

[30] V. Adep, ”AI based privacy data masking tool using spaCy,” Medium, [Online]. Available: https://medium.com/@venugopal.adep/ai-based-privacy-data-masking-tool-using-spacy-d2a5f81cbf02.

[31] Model Context Protocol, ”Getting Started: Introduction,” [Online]. Available: https://modelcontextprotocol.io/docs/getting-started/intro.

[32] LangChain, ”LangGraph,” [Online]. Available: https://www.langchain.com/langgraph.

[33] P. P. Mishra, K. P. Yeole, R. Keshavamurthy, M. B. Surana, and F. Sarayloo, ”Metadata-Aware Retrieval-Augmented Generation” (University of Illinois Chicago), arXiv:2512.05411, [Online]. Available: https://arxiv.org/pdf/2512.05411.

[34] L. Zheng, W.-L. Chiang, Y. Sheng, et al., ”Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena,” arXiv:2306.05685, preprint, Dec. 24, 2023.

[35] Sentence Transformers, ”Pretrained Models — Sentence Transformers documentation,” [Online]. Available: `https://www.sbert.net/docs/sentence_transformer/pretrained_models.html#original-models`.