Politecnico di Torino

Master's Degree in Mathematical Engineering
Academic Year 2024/2025

AI-Powered Document Intelligence with Retrieval-Augmented Generation

Designing a Large Language Model Agent for the Automated Analysis of Official Resolutions

Supervisor:
Riccardo COPPOLA
Luigi PREZIOSI

Candidate:
Valentina PAOLETTI

Abstract

This thesis presents the design and development of an intelligent agent based on the Retrieval-Augmented Generation (RAG) architecture, integrating a Large Language Model (LLM) to support automated analysis and generation of official resolutions with a fixed structural format. The proposed system addresses the limitations of conventional language models in handling domain-specific, high-precision information by combining neural text generation with targeted knowledge retrieval from structured and unstructured sources.

The work includes the implementation of a data ingestion pipeline for indexing resolutions, leveraging semantic embeddings to enable similarity-based retrieval. A carefully engineered prompting strategy, enriched with few-shot examples, guides the LLM in generating contextually accurate responses while preserving non-generative factual elements such as legal references, personal names, and financial data. The evaluation process covers both retrieval performance, measured through vector similarity metrics, and generation quality, assessed via domain-specific accuracy and completeness criteria.

Results demonstrate that the RAG-enhanced LLM approach offers factual reliability and contextual relevance. The proposed architecture gives a scalable and adaptable framework for intelligent document processing, with potential applications in public administration, legal tech, and regulatory compliance automation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Problem Statement

This section introduces the context in which the study was carried out, outlines the problem being addressed, and explains the practical motivations behind the proposed solution. This thesis presents the design and development of an intelligent AI agent aimed at automating the analysis and generation of official administrative documents, specifically in the medical field. The project was conducted during an internship at Reply, a company specializing in innovative digital solutions and consulting services that support organizations in embracing emerging technologies such as artificial intelligence and cloud computing.

In many public administration bodies, including healthcare organizations, the production and dissemination of official documents are still largely manual and time-consuming processes. This traditional approach not only demands significant human effort but also introduces a high risk of errors, which can compromise both efficiency and compliance. Furthermore, the growing volume and complexity of administrative documentation exacerbate these challenges, creating bottlenecks in information processing and decision-making.

The problem addressed in this thesis is how to leverage cutting-edge AI technologies to streamline and automate the creation, management, and querying of complex legal and administrative documents. Traditional language models, while powerful, often struggle to handle domain-specific, precise information such as legal references, medical terminology, proper names, and financial data. Moreover, they can produce outputs that are plausible in language but incorrect in factual content, known as hallucinations, which poses a significant risk in sensitive domains where accuracy is critical.

Automating these processes is of great practical importance because it allows organizations to reduce the workload on staff, minimize human errors, and accelerate access to relevant information. This, in turn, can lead to faster and more informed decision-making, improved regulatory compliance, and optimized resource allocation. In healthcare and public administration, such improvements not only enhance operational efficiency but also directly impact service quality, patient safety, and legal accountability. Additionally, the adoption of intelligent AI agents can foster a more consistent and auditable documentation process, ensuring that institutional knowledge is preserved and easily accessible across organizational units.

Overall, this study is motivated by the intersection of technical feasibility and practical necessity: advanced AI models capable of understanding and generating complex text meet the pressing need for accurate and efficient administrative systems. By integrating Retrieval-Augmented Generation (RAG) with agentic architectures, the proposed approach seeks to address both the technical challenges and operational demands of automating document workflows. Beyond immediate efficiency gains, AI-driven document processing offers a strategic opportunity for organizations to modernize operations, embrace innovation, and maintain competitiveness in a digital landscape.

1.2 Main Contribution

The main contribution of this thesis is the design and implementation of a Retrieval-Augmented Generation (RAG) based intelligent agent that integrates a Large Language Model (LLM) with semantic search techniques. The system combines neural text generation with targeted retrieval of relevant information from both structured and unstructured sources. This hybrid approach improves factual accuracy and contextual relevance in document analysis and generation tasks.

The development followed a comprehensive methodology including:

1. Theoretical analysis of recent AI architectures, with a particular focus on RAG and large language models;
2. Implementation of a data ingestion pipeline for indexing and embedding documents;
3. Design of prompting strategies enriched with few-shot examples to guide the generation;
4. Experimental evaluation through retrieval performance metrics and domain-specific quality assessments;
5. Integration of the system in a cloud-native architecture to ensure scalability and modularity.

1.3 Thesis Structure

This thesis aims to demonstrate how combining advanced AI models with effective retrieval mechanisms can transform document-intensive processes in public administration, offering scalable and accurate automation solutions. The work is organized as follows:

- **Chapter 1: Introduction**. It introduces the research context, the motivation for leveraging AI, and the main contribution of the thesis in designing a RAG-based intelligent system.
- **Chapter 2: Theoretical Foundations**. It provides the background on AI technologies relevant to this work, including language models, word embeddings, retrieval systems, and agentic architectures.
- **Chapter 3: System Architecture**. It defines the functional and non-functional requirements of the intelligent agent and the overall system, and describes the system architecture, covering backend components, frontend interfaces, and cloud deployment.
- **Chapter 4: Methodology**. It details the methodology and the workflow adopted for data processing, model integration, and generation.
- **Chapter 5: Results**. It presents the results, including tests and validation of retrieval and generation quality.
- **Chapter 6: User Interface and Interaction Modalities**. It showcases the user interface and explains how the system can be used in practice.
- **Chapter 7: Conclusions**. It concludes with a summary of contributions, limitations, and suggestions for future work.

Chapter 2

Theoretical Foundations

This chapter provides the theoretical and technical foundations necessary to contextualize the work presented in this thesis.

2.1 Language Modeling

Language Models (LMs) are foundational components in the field of Natural Language Processing (NLP), with applications ranging from text generation and machine translation to question answering and dialogue systems. Their primary objective is to estimate the probability of a sequence of words, thereby capturing the statistical properties and structure of human language [27, 33].

2.1.1 Definition and Objective

Formally, a language model defines a probability distribution over sequences of words, given by

$$P(w_1, w_2, \ldots, w_n).$$

By applying the chain rule of probability, this joint distribution can be factorized into a product of conditional probabilities; it holds the following relation:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, \ldots, w_{i-1}).$$

This decomposition enables the model to predict each word based on its preceding context, which is essential for both generative and discriminative language tasks [33, 76].

2.1.2 From Rule-Based to Statistical Models

Early language models were *rule-based* systems, relying on manually crafted grammars, lexical databases, and sets of linguistic rules defined by experts. In these systems, language understanding and generation were driven by explicit instructions such as syntactic patterns, morphological rules, or semantic frames. For instance, a rule might specify that a sentence in English follows the pattern `<Subject> <Verb> <Object>`, or that plural nouns are formed by adding the suffix `-s` unless specific exceptions apply.

While effective in highly constrained or domain-specific settings, such as early machine translation prototypes or text parsers, rule-based models exhibited two major limitations:

1. **Lack of scalability:** Creating and maintaining an extensive set of linguistic rules for all the complexities and exceptions of a natural language required immense effort from human experts.

2. **Limited generalization:** These systems struggled when faced with sentences or linguistic phenomena not explicitly covered by their rules, making them brittle and prone to failure in open-domain contexts.

The introduction of statistical approaches, particularly *n-gram* models, mitigated these limitations by learning patterns directly from real-world text, enabling better scalability, robustness to unexpected inputs, and probabilistic handling of linguistic ambiguity. In such models, the probability of a word is estimated given its preceding $n-1$ words:

$$P(w_i \mid w_{i-n+1}, \ldots, w_{i-1}).$$

Despite their simplicity and efficiency, *n-gram* models are limited by their fixed context size and susceptibility to data sparsity [27].

2.1.3 Limitations of Traditional Models

Traditional approaches to language modeling, both rule-based and statistical, struggle to capture long-range dependencies and nuanced semantic relationships. Additionally, they often rely on manually engineered features and are constrained by vocabulary size and memory requirements [27].

These limitations led to the development of *neural language models*, which use dense vector representations and deep learning architectures to learn language patterns from large corpora in a more expressive and scalable manner. This paradigm shift set the stage for modern architectures such as Recurrent Neural Networks (RNNs), Transformers, and Large Language Models (LLMs) [67, 76], which will be explored in the subsequent sections.

2.2 Transformers

The field of Language Modeling has progressed from using Recurrent Neural Networks to adopting Transformer architectures. These innovative architectures have not only redefined benchmarks in Natural Language Processing but have also extended their influence across various domains of artificial intelligence. Distinguished by their attention mechanisms and parallel processing capabilities, Transformer models represent a quantum leap in understanding and generating human language with unprecedented accuracy.

First introduced in 2017 through Google's seminal paper *"Attention is All You Need"* [67], the Transformer architecture underpins revolutionary models like ChatGPT and has become fundamental to OpenAI's language models. For data scientists and NLP practitioners, mastering Transformer models has become essential in this new era of AI advancement.

2.2.1 From RNNs to Transformers

Recurrent Neural Networks and Transformers are foundational architectures in Natural Language Processing. While RNNs process sequences sequentially, Transformers utilize self-attention mechanisms to process entire sequences simultaneously. We will now examine the motivations behind the evolution from RNNs to Transformers and why the latter have become the dominant architecture in NLP [4, 67].

Recurrent Neural Networks. RNNs are a class of neural architectures specifically designed to model sequential data by maintaining a hidden state that evolves over time, thereby capturing temporal dependencies in the input. At each time step t, given the input vector $\mathbf{x}_t \in \mathbb{R}^d$ and the previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^m$, the hidden representation is updated as:

$$\mathbf{h}_t = \sigma\left(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t + \mathbf{b}\right),$$

where $W_h \in \mathbb{R}^{m \times m}$ and $W_x \in \mathbb{R}^{m \times d}$ are learnable weight matrices, $\mathbf{b} \in \mathbb{R}^m$ is a bias vector, and $\sigma(\cdot)$ is a non-linear activation function, typically tanh or ReLU [17]. The output $\mathbf{y}_t$ can then be computed, for example, via a softmax layer in classification tasks:

$$\mathbf{y}_t = \text{softmax}(W_y \mathbf{h}_t + \mathbf{c}),$$

with $W_y \in \mathbb{R}^{k \times m}$ and $\mathbf{c} \in \mathbb{R}^k$ as output parameters.

The recurrent formulation allows RNNs to model temporal dynamics in tasks such as language modeling, speech recognition, and time series forecasting [42]. However, despite their theoretical ability to capture long-term dependencies, Recurrent Neural Networks face several practical limitations [3]:

- **Vanishing/exploding gradients**: During backpropagation through time (BPTT), the gradient of the loss function with respect to earlier time steps involves repeated multiplication of the recurrent weight matrix W_h. Formally, for a sequence of length T, the gradient at time step t is proportional to:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \propto \prod_{k=t}^{T} W_h^\top \, \text{diag}(\sigma'(\mathbf{a}_k)),$$

  where $\sigma'(\cdot)$ denotes the derivative of the activation function and $\mathbf{a}_k$ is the pre-activation input. If the largest singular value of W_h is greater than 1, gradients tend to explode; if it is smaller than 1, they vanish exponentially [5, 47].

- **Sequential processing**: The recursive dependency $\mathbf{h}_t \to \mathbf{h}_{t+1}$ prevents full parallelization across time steps, making RNNs computationally less efficient compared to architectures such as Transformers [67].

To overcome these limitations, more advanced variants were proposed. Long Short-Term Memory (LSTM) networks [28] and Gated Recurrent Units (GRU) [10] introduce gating mechanisms that regulate information flow, allowing the network to preserve long-range dependencies more effectively. In particular, LSTMs use input, output, and forget gates to control memory updates, while GRUs combine update and reset gates to provide a more compact alternative [12]. These architectures have demonstrated superior performance across a wide range of sequential modeling tasks, including natural language processing and speech recognition [25, 64].

Transformers. Transformer models represent a breakthrough in the design of neural architectures for sequential data processing, introducing a paradigm shift from recurrence- or convolution-based methods to attention-based mechanisms. Instead of relying on sequential computation as in Recurrent Neural Networks, Transformers employ a *self-attention* mechanism that computes pairwise interactions between elements of the input sequence, assigning context-dependent weights to each token [67]. This design enables the model to capture both short- and long-range dependencies more effectively.

Originally proposed for sequence transduction tasks such as neural machine translation, Transformers adopt an encoder-decoder architecture in which the encoder maps an input sequence to a continuous representation and the decoder generates an output sequence autoregressively [67]. Their name reflects their ability to transform an input sequence into an output sequence, making them particularly suited for tasks such as machine translation, summarization, and dialogue systems. At their core, Transformers can be described as neural networks that learn contextual representations by analyzing vast corpora of sequential data, which has led them to become the backbone of modern Natural Language Processing systems [8, 15, 66].

The elimination of recurrence is a defining characteristic: unlike traditional encoder-decoder models based on RNNs, Transformers achieve parallelization by processing all sequence elements

simultaneously, which significantly accelerates training. Furthermore, self-attention enables a more efficient handling of long-range dependencies, a limitation of RNNs due to vanishing gradients [5, 47]. These advantages have allowed Transformers to scale effectively with both data and model size, a property that has fueled the rise of Large Language Models with billions of parameters [8, 55]. Key features of the Transformer architecture include:

- **Parallelization:** The absence of sequential recurrence allows all tokens to be processed simultaneously, greatly reducing training time compared to RNN-based architectures.
- **Long-range dependency modeling:** Self-attention enables the model to directly connect any two tokens in a sequence, regardless of their distance, overcoming the limitations of fixed-size contexts in RNNs.
- **Scalability:** Transformers scale effectively with both data and computational resources, enabling the training of extremely large models that achieve state-of-the-art performance across diverse NLP tasks.

2.2.2 The Transformer Architecture

The original Transformer is built upon an encoder-decoder architecture, which is particularly effective for sequence-to-sequence tasks such as machine translation [67]. This architecture is composed of two primary components:

- **Encoder:** Transforms an input sequence of tokens into a sequence of embedding vectors, often referred to as hidden states or contextual representations.
- **Decoder:** Generates an output sequence of tokens one step at a time, conditioning on both the encoder's output and the previously generated tokens.

The Transformer processes input text by first tokenizing it and mapping the tokens to dense embeddings. Since the attention mechanism does not encode positional information, positional embeddings are added to the token embeddings to retain the order of the sequence [67].

Each encoder layer in the stack processes the token embeddings and passes the result to the next layer. The decoder stack similarly processes embeddings, attending to both previous outputs and the encoder's final hidden states. At each decoding step, the decoder predicts the next most probable token, feeding it back into the model until an end-of-sequence (EOS) token is produced or a predefined maximum length is reached [1].

Although originally proposed for translation [67], the encoder and decoder components of the Transformer have been adapted for a broad range of tasks. Transformer-based models typically fall into three main categories [30]:

- **Encoder-only models:** These models, such as BERT [15], RoBERTa [41], and DistilBERT [59], are optimized for tasks requiring deep contextual understanding of input text, such as classification and named entity recognition. They leverage bidirectional self-attention to capture context from both sides of a token.
- **Decoder-only models:** Examples include the GPT family [8, 52, 53]. These models generate text autoregressively, predicting one token at a time based solely on leftward context (causal attention).
- **Encoder-decoder models:** Designed for tasks involving input-to-output mappings, such as translation and summarization. Aside from the original Transformer [67], models like BART [37] and T5 [55] also fall into this category.

In practice, the distinctions between these categories are not always strict. For example, GPT-style models can perform translation if prompted appropriately, and BERT-like models have been adapted for summarization.

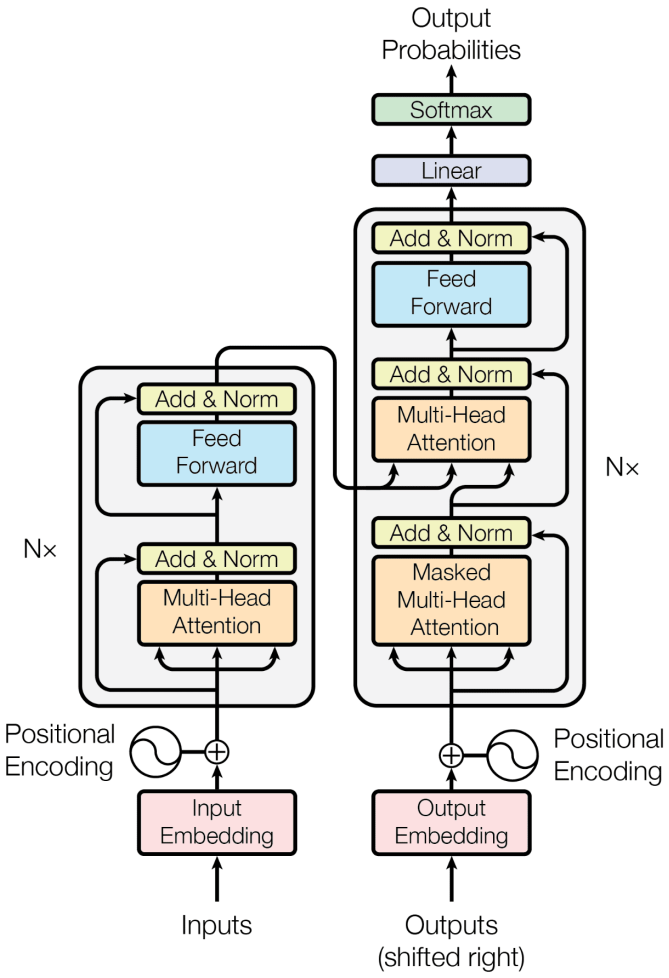


Figure 2.1: Architecture of the Transformer model, from [67].

The Encoder

The encoder is composed of a stack of identical layers [67]. Each encoder layer applies two main subcomponents to its input embeddings:

- **Multi-head self-attention:** Allows each token to attend to every other token in the sequence, enabling the model to capture dependencies regardless of distance.
- **Feed-forward network (FFN):** A fully connected network applied independently to each token embedding.

Each sublayer is followed by a residual connection and layer normalization, facilitating stable training of deep networks [67]. The goal of the encoder is to produce contextualized representations of tokens: for example, disambiguating the word "apple" depending on whether nearby words like "keynote" or "fruit" appear.

In particular, self-attention is the core mechanism of the Transformer, allowing the model to weigh the relevance of different tokens when computing the representation of each one. Unlike attention in RNN-based architectures, which links encoder and decoder states, self-attention operates within a single sequence [67].

Formally, given a sequence of input embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_n$, self-attention computes new embeddings $\mathbf{x}'_1, \ldots, \mathbf{x}'_n$ such that each output $\mathbf{x}'_i$ is a weighted sum of all input embeddings:

$$\mathbf{x}_i' = \sum_{j=1}^{n} w_{ji}\mathbf{x}_j,$$

where the weights w_{ji}, known as *attention weights*, reflect the contextual relevance between tokens, and satisfy the following constraint:

$$\sum_{j=1}^{n} w_{ji} = 1.$$

This mechanism enables the model to dynamically reweight token representations based on their surrounding context. For instance, the meaning of the word "flies" changes depending on whether the sentence is "fruit flies are common" or "time flies like an arrow" [1].

The most commonly used variant of attention in Transformers is scaled dot-product attention [67]. It proceeds in four steps:

1. **Projection:** Each input embedding is linearly projected to three vectors: a query $\mathbf{q}$, a key $\mathbf{k}$, and a value $\mathbf{v}$.
2. **Score computation:** Attention scores are calculated using the dot product between queries and keys. These scores measure similarity between tokens.
3. **Normalization:** The scores are scaled by the square root of the key dimension and passed through a softmax to produce attention weights.
4. **Weighted sum:** The output embedding is computed as the weighted sum of the value vectors using the attention weights.

Mathematically, given query, key, and value matrices Q, K, V, the output of scaled dot-product attention is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

where d_k is the dimension of the key vectors. The softmax is applied row-wise, ensuring that the attention weights form a probability distribution over the input tokens. This mechanism enables each token representation to be updated as a weighted combination of all others, with weights reflecting learned contextual relevance.

A fundamental extension is multi-head attention. Instead of computing attention once, the model projects the input embeddings into multiple subspaces. For each head $h \in \{1, \ldots, H\}$, three learned projection matrices are applied:

$$Q^{(h)} = XW_Q^{(h)}, \quad K^{(h)} = XW_K^{(h)}, \quad V^{(h)} = XW_V^{(h)},$$

with $W_Q^{(h)}, W_K^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W_V^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_v}$. Here, d_{model} denotes the dimensionality of the token embeddings and the internal representations throughout the Transformer, defining the size of the vectors processed by all layers. Each head then computes its own attention output:

$$\text{head}^{(h)} = \text{Attention}(Q^{(h)}, K^{(h)}, V^{(h)}).$$

The outputs of all H heads are concatenated and linearly transformed:

$$\text{MultiHead}(X) = \text{Concat}\left(\text{head}^{(1)}, \ldots, \text{head}^{(H)}\right)W_O,$$

where $W_O \in \mathbb{R}^{Hd_v \times d_{\text{model}}}$. A common setting is $d_k = d_v = d_{\text{model}}/H$, ensuring that the concatenated result has the same dimensionality d_{model} as the input.

Each head performs attention computations independently, allowing the model to focus on various linguistic or semantic relations. For instance, one head might capture subject-verb dependencies, while another identifies adjective-noun associations. These relationships are not predefined but are learned from data during training.

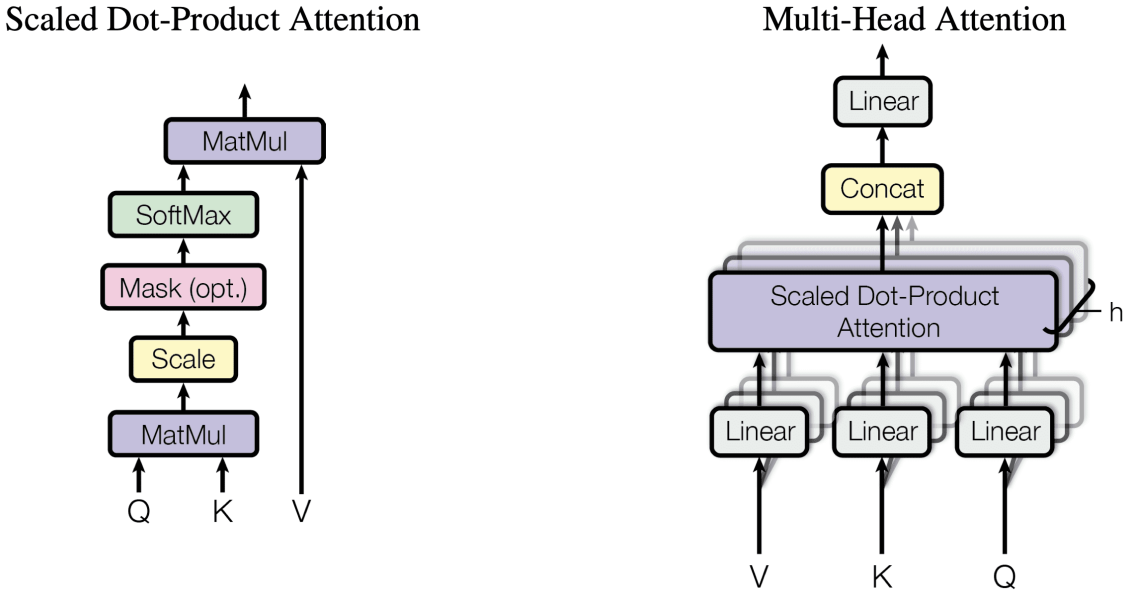


Figure 2.2: Scaled dot-product attention and multi-head attention, from [67].

The feed-forward layer in the encoder and decoder consists of a two-layer fully connected neural network. However, unlike traditional architectures that operate on the entire sequence collectively, this layer processes each token embedding independently. For this reason, it is referred to as a *position-wise feed-forward layer*.

The Transformer architecture further incorporates residual connections and layer normalization to stabilize and accelerate training. Formally, denoting the output of a sublayer as $\text{Sublayer}(\mathbf{x})$, there are two main schemes for applying layer normalization:

- **Post-norm:** Layer normalization is applied after the residual connection:

$$\mathbf{y} = \text{LayerNorm}(\mathbf{x} + \text{Sublayer}(\mathbf{x})).$$

  This was the setup used in the original Transformer paper [67]. However, training from scratch with this setup can be unstable due to gradient issues, often necessitating a learning rate warm-up.

- **Pre-norm:** Layer normalization is applied before the residual addition:

$$\mathbf{y} = \mathbf{x} + \text{Sublayer}(\text{LayerNorm}(\mathbf{x})).$$

  This configuration is more stable in practice and usually does not require warm-up schedules.

Token embeddings alone lack position information. To incorporate positional context, positional embeddings are added to the input embeddings. These vectors encode position information in a way that enables the model to distinguish token order. Let $\mathbf{p}_i \in \mathbb{R}^{d_{\text{model}}}$ denote the positional embedding for token i. The input to the first encoder layer becomes:

$$\mathbf{z}_i = \mathbf{x}_i + \mathbf{p}_i.$$

Several strategies exist to define positional embeddings:

- **Learnable positional embeddings:** Position-specific vectors are learned during training. These are effective when large amounts of training data are available.
- **Absolute positional representations:** Use fixed patterns, such as sine and cosine waves, to encode positions, e.g.,

$$\mathbf{p}_i[2k] = \sin\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right), \quad \mathbf{p}_i[2k+1] = \cos\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right),$$

  for $k = 0, \ldots, d_{\text{model}}/2 - 1$ [67]. This approach is beneficial in low-data regimes.
- **Relative positional representations:** Encode the relative distances between tokens instead of absolute positions. This requires modifying the attention mechanism itself. For example, DeBERTa incorporates relative position information directly into the attention computation.

Transformer models are generally organized into a *task-agnostic body*, which encodes the input sequence into contextualized representations, and a *task-specific head* that performs the final prediction. For classification tasks, a classification head is added on top of the encoder. Since the encoder produces a hidden state for each token, it is common practice to use the hidden state corresponding to a special token as a summary representation of the entire sequence. This representation is then passed through a dropout layer to improve generalization and finally through a linear layer to produce the class scores for the prediction.

The Decoder

The Transformer decoder differs from the encoder in that it includes two types of attention sublayers [1, 66, 67, 72]:

- **Masked multi-head self-attention:** Prevents attending to future tokens during training by masking subsequent positions. This enforces autoregressive behavior, ensuring each output token is generated based only on preceding tokens.
- **Encoder-decoder attention:** Allows the decoder to attend to the encoder outputs. Queries come from the decoder's intermediate representations, while keys and values are derived from the encoder. This mechanism facilitates cross-sequence alignment, such as translating between languages.

2.2.3 Applications in Natural Language Processing

Transformers have revolutionized NLP by achieving state-of-the-art results in various tasks [72], including:

- **Machine translation:** Converting text from one language to another with high accuracy.
- **Text summarization:** Generating concise summaries of long documents.
- **Question answering:** Providing precise answers to questions based on context.
- **Sentiment analysis:** Determining the sentiment expressed in a piece of text.

2.3 Large Language Models

The success of the Transformer architecture has paved the way for the development of Large Language Models. While Recurrent Neural Networks and their variants marked an important step in modeling sequential data, the paradigm shift introduced by Transformers enabled unprecedented scalability, parallelization, and contextual understanding. These properties made it possible to train models with billions of parameters on massive corpora, giving rise to LLMs as the natural evolution of Neural Language Modeling. The transition from traditional models to LLMs has been driven not only by architectural innovations, but also by the increase in computational power, the advancement of training techniques, and the availability of large-scale datasets.

2.3.1 Characteristics and Training

Large Language Models represent the culmination of advancements in Neural Language Modeling, scaling both the architecture and the training data to unprecedented levels and leading to many changes across multiple work domains. These models are typically based on the Transformer architecture, and are trained on massive and heterogeneous corpora [44].

LLMs, such as GPT (Generative Pre-trained Transformer) [8, 45], BERT (Bidirectional Encoder Representations from Transformers) [16], and their successors, are characterized by:

- **Scale**: They often contain billions (or even trillions) of parameters, enabling them to model complex linguistic structures and world knowledge.
- **Pretraining and finetuning**: LLMs are usually pretrained on large-scale unsupervised tasks, such as language modeling or masked token prediction, and then finetuned on specific downstream tasks.
- **Generalization**: Due to their size and the diversity of training data, LLMs exhibit strong generalization capabilities across a wide range of NLP applications, often requiring minimal task-specific adaptation.

At their core, Large Language Models learn to model the conditional probability of the next token given a context:

$$P(w_n \mid w_1, w_2, \ldots, w_{n-1}).$$

This allows them to generate coherent, contextually relevant text, making them suitable for tasks such as summarization, translation, dialogue, and code generation [76].

Importantly, the power of LLMs lies not only in their predictive capacity, but also in the rich, high-dimensional representations they produce for tokens and sequences. These representations, known as embeddings, will be the focus of one of the next sections.

2.3.2 Notable Examples and Evolution

The evolution of Large Language Models has been marked by several landmark models that introduced key innovations and demonstrated the potential of scaling neural architectures. Some of the most notable examples include:

- **ELMo (Embeddings from Language Models)** [50]: Among the first contextual word embedding models, ELMo demonstrated that deep, bidirectional language models could capture nuanced semantic and syntactic information, laying the foundation for subsequent pretrained models.
- **GPT (Generative Pre-trained Transformer)** [52]: Introduced the paradigm of pretraining a Transformer decoder on large-scale text corpora followed by finetuning for downstream tasks. This approach significantly improved transfer learning in NLP.

- **BERT (Bidirectional Encoder Representations from Transformers)** [16]: BERT pioneered the use of bidirectional Transformers and masked language modeling, achieving state-of-the-art results on multiple NLP benchmarks and popularizing the pretraining–finetuning paradigm.
- **GPT-2** [53]: With 1.5 billion parameters, GPT-2 showcased the emergent ability of large-scale models to generate coherent, long-form text without task-specific training, sparking significant public and academic interest.
- **GPT-3** [8]: Scaling up to 175 billion parameters, GPT-3 introduced the concept of in-context learning, demonstrating that sufficiently large models could perform tasks from examples provided directly in the prompt, without additional finetuning.
- **PaLM (Pathways Language Model)** [11]: With 540 billion parameters, PaLM demonstrated state-of-the-art few-shot and reasoning capabilities, highlighting the role of scaling laws and advanced training strategies in pushing the limits of LLMs.
- **GPT-4** [45]: The most widely known model in the GPT series, GPT-4 further advanced the robustness, factual accuracy, and multimodal capabilities of LLMs, setting a new standard for general-purpose language understanding and generation.
- **GPT-5** [9]: Released in August 2025, GPT-5 represents a unified, multimodal model architecture that dynamically routes between fast and deep "thinking" modes via a real-time router, optimizing performance based on task complexity.

These milestones illustrate the rapid progress in model architecture, scale, and training methodologies that have defined the trajectory of LLM development, ultimately enabling their application across diverse domains, including those examined in this thesis.

2.3.3 Applications

Large Language Models have found widespread applications across various natural language processing tasks, demonstrating remarkable versatility and effectiveness [27]. Some of the most relevant applications include:

- **Text generation:** LLMs can generate coherent and contextually relevant text, enabling automated report writing, content creation, and drafting of official documents.
- **Summarization:** By condensing long documents into concise summaries, LLMs facilitate rapid understanding of complex texts, which is particularly valuable in administrative and legal contexts.
- **Translation:** LLMs support multilingual applications by translating text between languages with high accuracy, improving accessibility and communication.
- **Question answering and dialogue systems:** LLMs can interpret natural language queries and provide informative responses, forming the basis of conversational agents and AI assistants.
- **Code generation and analysis:** Certain LLMs can assist in generating, understanding, and debugging code, supporting software development and automation tasks.

In the context of this thesis, LLMs are employed primarily for document analysis, information retrieval, question answering, and automated generation of administrative texts, demonstrating their practical relevance in public administration and healthcare organizations.

2.3.4 Challenges

Despite their remarkable capabilities, Large Language Models face several challenges that must be considered, especially in critical domains such as administrative document processing [27,30].

1. **Generability:** While LLMs are highly flexible, their ability to generalize to unseen tasks or domain-specific content is not perfect. Models pretrained on broad corpora may struggle with specialized terminology, precise legal or financial references, and nuanced domain-specific rules, which can lead to inaccurate or misleading outputs.

2. **Evaluation:** Assessing the performance of LLMs is inherently difficult. Standard metrics, such as perplexity or BLEU scores [46], often fail to capture the quality, factual correctness, or usefulness of generated text. Human evaluation is frequently required, but it is costly and time-consuming, making automated, reliable evaluation a persistent challenge.

3. **Interpretability:** LLMs operate as highly complex, high-dimensional neural networks, which makes their internal decision-making opaque. Understanding why a model produced a certain output, identifying potential biases, or explaining errors remains an active area of research. This lack of transparency can hinder trust, particularly in regulated sectors such as healthcare and public administration.

4. **Hallucinations:** LLMs may produce information that is plausible in form but incorrect or fabricated in content, a phenomenon known as hallucination. This is particularly problematic in high-stakes domains, where inaccurate outputs can lead to misunderstandings, legal issues, or faulty decision-making. Mitigating hallucinations often requires integrating external verification mechanisms, domain-specific knowledge bases, or retrieval-augmented generation techniques to improve factual reliability.

5. **Ethical considerations:** The deployment of LLMs raises significant ethical challenges, including ensuring fairness, preventing biased or discriminatory outputs, and protecting sensitive personal or organizational data. Failing to address these concerns can result in harm, legal repercussions, and loss of public trust, making ethical oversight a critical component of any LLM-based system.

Addressing these challenges is crucial for safely and effectively deploying LLM-based systems in practical applications, guiding the design of tools like the AI agent developed in this thesis.

2.4 Word Embeddings

As discussed in the previous section, the strength of Large Language Models lies not only in their ability to predict the next word, but also in the rich, high-dimensional representations, known as *embeddings*, they generate for words and sequences.

2.4.1 Definition and General Characteristics

Word embeddings are dense vector representations of words, phrases, or entire texts. These vectors capture the semantic meaning of the input by positioning similar concepts closer together in a high-dimensional space. In modern NLP, Large Language Models generate contextual embeddings, which extend this idea by making the representation dependent on context.

Unlike one-hot encoding, which represents words as sparse vectors with no meaningful structure, LLM embeddings encode words into lower-dimensional, dense vectors. This encoding preserves semantic similarity: words with similar meanings or usage contexts are mapped to nearby points in the embedding space. They enable context-aware representations, allowing for disambiguation of polysemous words based on surrounding text. This makes them particularly

useful for a variety of Natural Language Processing tasks such as sentiment and trend analysis, text classification, semantic search and information retrieval, question answering, conversational agents and chatbots, machine translation, and Retrieval-Augmented Generation pipelines [2,65].

2.4.2 Embedding Strategies

There are different types of embeddings and the following paragraphs provide an overview of the most commonly used approaches [65].

Word-Level Embeddings. These vectors are static embeddings, obtained from dedicated models, which do not rely on Large Language Models. They assign a fixed vector to each word, independent of context. Popular examples include:

- Word2Vec: Uses either Continuous Bag of Words (CBOW) or Skip-gram architectures to predict a word from its surrounding context or vice versa. It is particularly effective at capturing local word relationships [43].
- GloVe (Global Vectors): Builds embeddings by factorizing a word co-occurrence matrix derived from a corpus, capturing broader, global statistical relationships [49].
- FastText: Extends Word2Vec by incorporating subword information, improving representations for rare or morphologically complex words [7].

Contextualized Word Embeddings. Unlike word-level embeddings, these vectors are contextual embeddings, produced by modern Large Language Models, where representations vary depending on the surrounding text, allowing for a richer semantic representation:

- ELMo: Generates context-sensitive embeddings by analyzing the entire sentence, capturing how a word's meaning changes with context [51].
- BERT: Produces embeddings conditioned on both left and right context, making it suitable for tasks requiring deep semantic understanding [16].
- GPT: Uses left-to-right context to generate embeddings, particularly effective for generative tasks such as text completion [54].

2.4.3 Key Properties

LLM embeddings exhibit several important properties:

- **Dimensionality**: Fixed-length vectors (typically hundreds to thousands of dimensions) encode rich information.
- **Contextuality**: Word meaning is dynamically adjusted based on textual context.
- **Semantic similarity**: Similar concepts are mapped to nearby vectors in the embedding space.
- **Transferability**: Pre-trained embeddings can be reused across tasks with minimal adaptation.
- **Robustness and adaptability**: Capable of handling ambiguity, synonyms, and cross-lingual contexts.
- **Multi-modality**: Some LLMs support embeddings across text, images, and other data types.

Many of these properties, such as fixed dimensionality, semantic similarity, and transferability, are also shared by traditional word embeddings. However, LLM-based embeddings extend these capabilities by introducing contextuality and, in some cases, multimodality, making them more robust and adaptable to complex NLP tasks [2].

2.5 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) represents a powerful architectural paradigm that extends the capabilities of Large Language Models by incorporating external, non-parametric knowledge through retrieval mechanisms. The central idea is to augment the generative process with factual information dynamically retrieved from a corpus of documents, thus mitigating issues such as hallucinations and knowledge cutoffs typical of standalone LLMs [38].

2.5.1 Core Components of a RAG Pipeline

A Retrieval-Augmented Generation (RAG) pipeline integrates information retrieval with generative modeling to enhance the quality and factuality of generated content [39]. The core components of a RAG pipeline can be broadly categorized into retrieval, augmentation, and generation.

1. **Retrieval** is the first stage in the pipeline and involves identifying relevant documents or data fragments from a large corpus in response to a query. This step typically employs dense or sparse vector representations, such as embeddings from pre-trained transformers, to compute semantic similarity between the query and potential documents [34]. Effective retrieval is crucial because it determines the quality and relevance of the information available for the subsequent generation step.

2. **Augmentation** refers to the process of integrating the retrieved information with the input query or context to create an enriched prompt for the generative model. This step often involves concatenating, summarizing, or re-ranking retrieved documents to ensure that the most pertinent information is emphasized while reducing redundancy or noise [31]. Proper augmentation allows the generative model to leverage external knowledge effectively without hallucinating or producing incorrect outputs.

3. **Generation** is the final component, where a language model produces a response conditioned on the augmented input. Generative models in RAG pipelines are typically large-scale transformer architectures capable of producing coherent and contextually appropriate text [53]. By grounding generation in retrieved evidence, RAG systems improve factual consistency and enable more accurate and reliable outputs compared to standalone generative models.

Overall, the RAG architecture embodies a synergy between retrieval-based grounding and generation-based flexibility, allowing systems to answer queries with higher precision and contextual awareness.

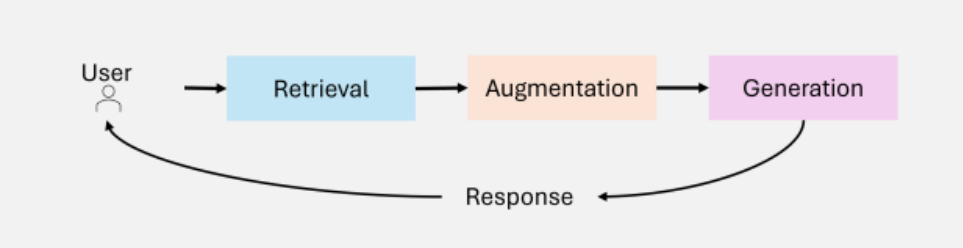


Figure 2.3: Core components of a RAG pipeline, from [61].

2.5.2 Advantages of the RAG Approach

The Retrieval-Augmented Generation architecture provides several notable advantages over traditional language models [38]:

- **Updatability**: The knowledge base can be updated dynamically by modifying the contents of the external vector database, without requiring retraining of the underlying language model. This allows the system to incorporate new information rapidly and maintain relevance over time.
- **Explainability**: Each generated response can be directly linked to the supporting documents retrieved from the database, providing transparency and enabling users to trace the source of the information.
- **Scalability**: By leveraging an external memory, RAG models can access and reason over vast knowledge bases that extend well beyond the model's pretraining data, supporting applications that require extensive and up-to-date information.
- **Reduced hallucinations**: By grounding responses in retrieved documents, RAG models are less likely to produce incorrect or fabricated information, improving factual reliability in critical applications.
- **Cross-domain flexibility**: RAG can integrate information from diverse domains or data types, enabling the system to handle heterogeneous knowledge sources without retraining the language model.

2.5.3 Pipeline Optimization: Stages and Techniques

While the core RAG architecture is conceptually straightforward, its effectiveness can be substantially enhanced through specialized optimization techniques applied across different stages of the pipeline.

In the context of advanced RAG [61], these stages include indexing, pre-retrieval filtering, retrieval, and post-retrieval refinement [70]. Such optimizations enable more precise selection and integration of relevant knowledge, improving both the factual accuracy and the efficiency of the generative output.

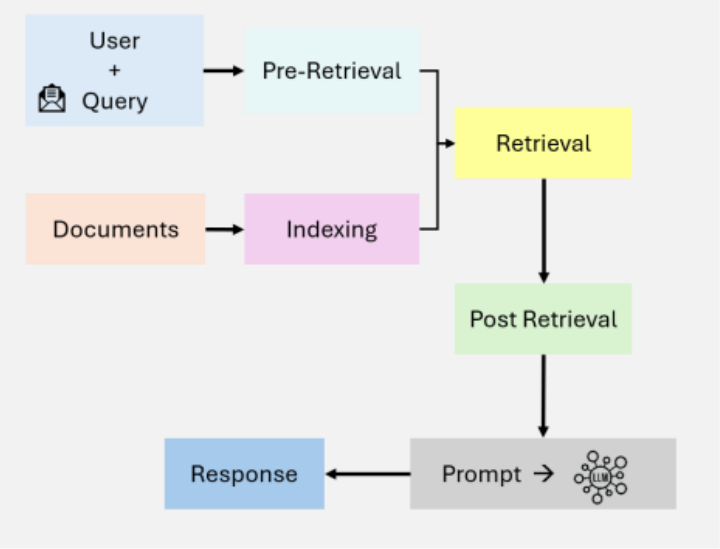


Figure 2.4: Overview of advanced RAG, from [61].

Indexing Optimization. The indexing stage concerns the preparation of data before it becomes searchable. This begins with *data pre-processing*, where documents are collected, cleaned, and transformed. Textual content is extracted from various formats, such as PDFs, Word documents, or HTML, and, where necessary, Optical Character Recognition (OCR) is employed for scanned images or other non-machine-readable formats. During cleaning, elements like headers, footers, and boilerplate content are removed, and text is standardized into a consistent format.

Once cleaned, documents must be segmented into chunks. This process, known as *chunking*, is non-trivial and has a direct impact on retrieval quality. Several chunking strategies exist, for example:

- Fixed-size chunking: it's simple but risks breaking semantic coherence.
- Recursive chunking: it respects structural units such as paragraphs and sentences.
- Semantic chunking: it uses vector similarity to preserve contextual integrity.
- LLM-based chunking: it leverages language models to generate high-quality, context-aware divisions, though at higher computational cost.

Pre-Retrieval Optimization. Before retrieval takes place, one can improve performance through *query transformation*. This includes techniques like *query rewriting*, where the original user input is reformulated, often via an LLM, to better align with the style and vocabulary of the indexed content.

Another approach is *query expansion*, where multiple semantically similar queries are generated from the original, increasing recall at the potential cost of lower precision. In more complex pipelines, *query routing* can be employed to direct different types of questions to different specialized retrievers or indexes, optimizing relevance based on query intent.

Retrieval Optimization. During the retrieval phase, several enhancements can be introduced to improve the relevance and accuracy of retrieved chunks. One such technique is *metadata filtering*, where additional document attributes, such as date, source, or category, are used to constrain the search space. This is particularly effective in domains where temporal or categorical relevance is critical.

A powerful extension of standard vector retrieval is *hybrid search*, which combines dense (embedding-based) search with sparse (keyword-based) retrieval techniques, such as BM25 or TF-IDF. Hybrid search often yields better results in scenarios where embeddings alone fail to capture rare or domain-specific terms. Furthermore, *embedding fine-tuning* allows the adaptation of the embedding model to domain-specific corpora, improving semantic matching for specialized vocabularies or contexts.

Post-Retrieval Optimization. Once documents have been retrieved, further processing can enhance the quality of the final response. One common method is *re-ranking*, where a secondary model evaluates the relevance of each retrieved chunk in context, enabling better prioritization. This can be based on cross-encoders or LLM scoring functions.

In addition, *context processing* techniques can be applied to modify the retrieved content before it is fed into the prompt. These include *context enhancement*, which appends metadata or expands the window of surrounding text, and *context compression*, which reduces token length by eliminating redundant or irrelevant parts. Finally, if the target domain requires highly specialized knowledge, *LLM fine-tuning* may be considered. This involves training the generative model on domain-specific datasets to align its outputs with expected terminology and reasoning patterns. However, this step is resource-intensive and requires high-quality annotated data.

2.5.4 Holistic System Considerations

When designing a RAG pipeline, it is essential to evaluate the trade-offs at each stage [38, 62]. Indexing optimizations often offer the best return on investment, especially in early development [62]. Pre-retrieval and retrieval enhancements can be selectively applied based on use-case complexity, while post-retrieval modifications and model fine-tuning are reserved for high-precision applications [31].

Moreover, considerations such as latency, token usage, and computational cost must inform system design, particularly when deploying at scale [38]. Establishing rigorous validation procedures, including offline metrics like retrieval precision and end-to-end user satisfaction, is key to ensuring that pipeline changes result in measurable improvements [31, 62].

In conclusion, the Retrieval-Augmented Generation framework provides a modular and extensible architecture for grounding LLM outputs in external knowledge [38]. By strategically applying optimization techniques across the indexing, retrieval, and generation pipeline, one can build robust systems capable of delivering factual, relevant, and context-aware responses [31, 62]. These capabilities make RAG particularly suited for enterprise applications, legal and scientific domains, and any scenario where accuracy and transparency are paramount.

2.6 Prompt Engineering in RAG Systems

Prompt engineering refers to the process of designing and structuring inputs, typically consisting of user instructions, examples, and retrieved content, in a way that elicits the desired output from a language model. It plays a pivotal role in Retrieval-Augmented Generation, as it serves as the final interface between retrieved knowledge and the generative model. Even when relevant documents are correctly retrieved, the effectiveness of the final output largely depends on how this information is presented to the Large Language Model through the prompt.

A well-engineered prompt can guide the model toward accurate, structured, and contextually appropriate responses, while a poorly designed one can lead to irrelevant, verbose, or even incorrect generations. This involves not only formatting the prompt correctly but also choosing the right strategy depending on the complexity of the task, the capabilities of the model, and the use case requirements [58, 76].

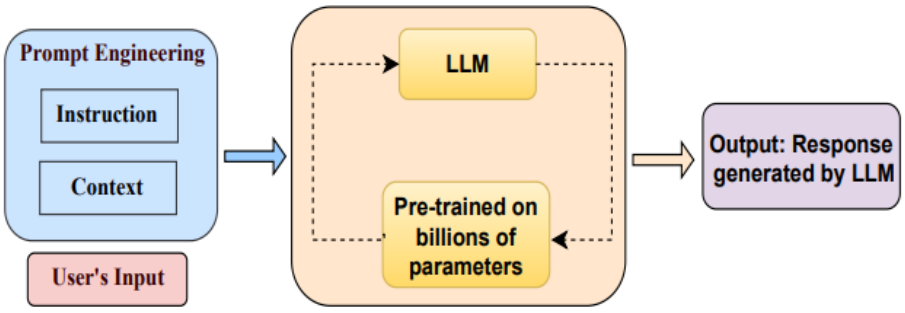


Figure 2.5: Visual breakdown of prompt engineering components, highlighting Large Language Models trained on extensive datasets, the pivotal role of instructions and context in shaping prompts, and the user input interface enabling interaction, from [58].

2.6.1 Prompting Strategies

There exist multiple prompting strategies, ranging from simple to highly sophisticated, that are designed to address tasks across different domains. Figure 2.6 illustrates a taxonomy of these techniques, providing a structured overview of how prompts can be customized depending on the application context.

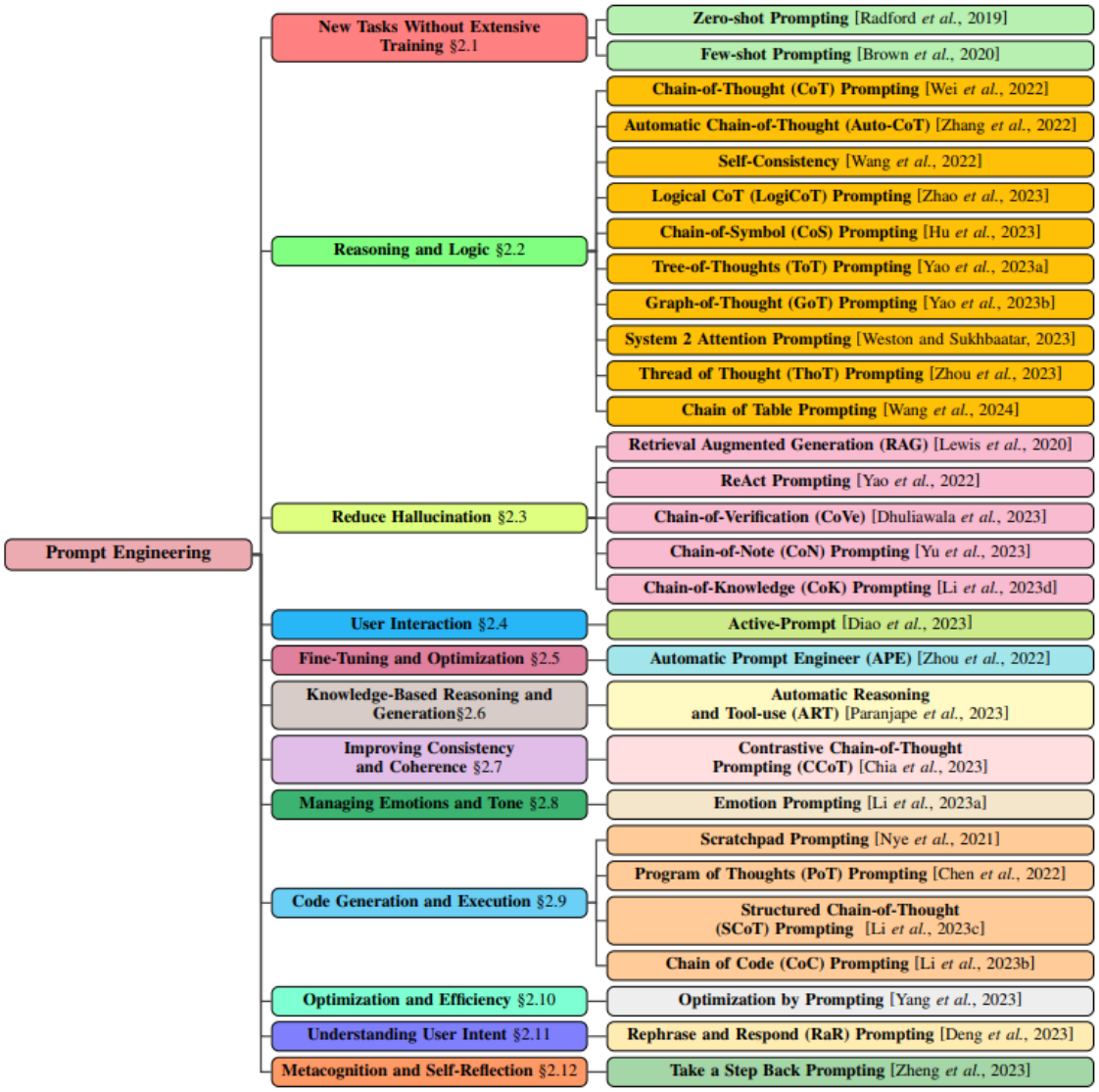


Figure 2.6: Taxonomy of prompt engineering techniques in Large Language Models, organized across application domains, from [58].

Several prompting strategies have emerged as particularly effective within the RAG framework [58]. In the following, the most relevant approaches are examined.

Zero-Shot Prompting. It involves providing the model with a task instruction without any accompanying examples. The model is expected to infer the desired behavior from the instruction alone. This approach is simple and cost-efficient, as it requires no additional tokens for demonstrations. However, its effectiveness is highly dependent on the clarity and specificity of the instruction. In RAG, zero-shot prompts are often used for straightforward tasks or when the user query itself can serve as a clear directive.

Few-Shot Prompting. It enhances the model's performance by including a small number of task-specific examples within the prompt. These examples demonstrate the format, style, and logic that the model should follow. In a RAG context, few-shot prompts can be combined with retrieved documents to show the model how to synthesize and reference external information when answering a question. This technique is particularly useful when the task involves complex reasoning or domain-specific language.

Chain-of-Thought Prompting. It encourages the model to generate intermediate reasoning steps before arriving at a final answer. By prompting the model to "think aloud," Chain-of-Thought Prompting (CoT) enables more transparent and accurate reasoning, especially for tasks involving arithmetic, logic, or multi-step deduction. Within RAG systems, CoT can be integrated with retrieved evidence to ensure that the model not only references external documents but also articulates a rational pathway to the answer.

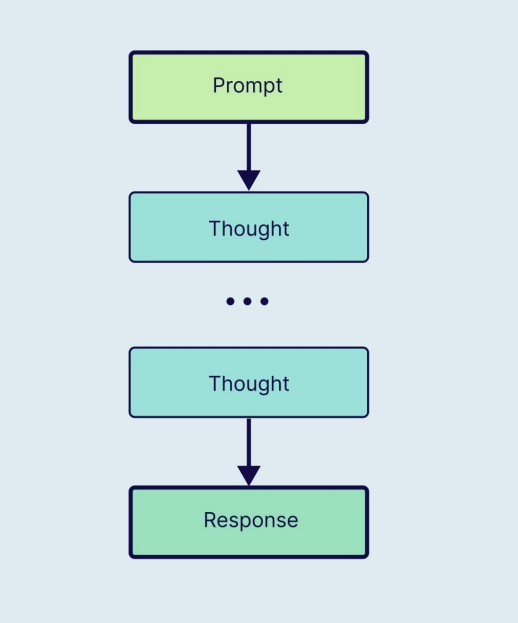


Figure 2.7: Chain-of-Thought prompting, from [70].

Tree-of-Thought Prompting. It extends CoT by generating multiple reasoning paths in parallel and evaluating their outcomes before choosing the best one. This mimics a tree search, where the model explores various branches of thought. While computationally more expensive, Tree-of-Thought Prompting (ToT) can improve robustness in decision-making tasks. In a RAG system, this can be especially effective when multiple retrieved documents suggest competing answers, allowing the model to weigh different perspectives before concluding.

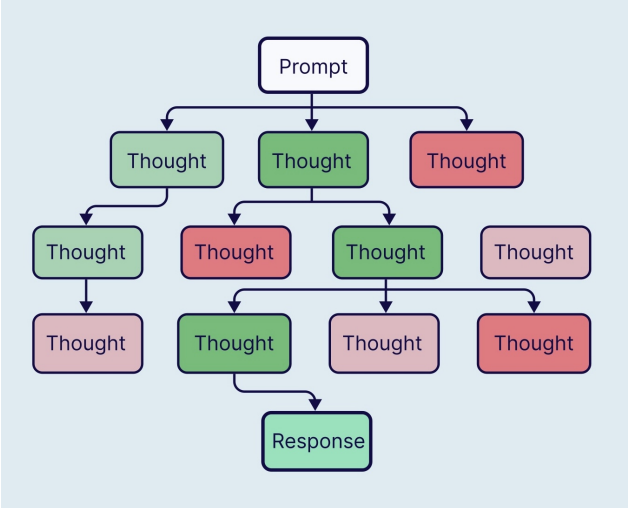


Figure 2.8: Tree-of-Thought prompting, from [70].

ReAct (Reasoning + Acting). The ReAct framework integrates reasoning and decision-making with interaction, allowing the model to reason step-by-step while also performing actions such as retrieving new information or asking clarification questions. In a RAG setting, ReAct can be used to simulate dynamic interactions with the retrieval system, where the model not only consumes the retrieved content but actively decides when and how to retrieve additional documents. This turns the generation process into an iterative dialogue between reasoning and information gathering.

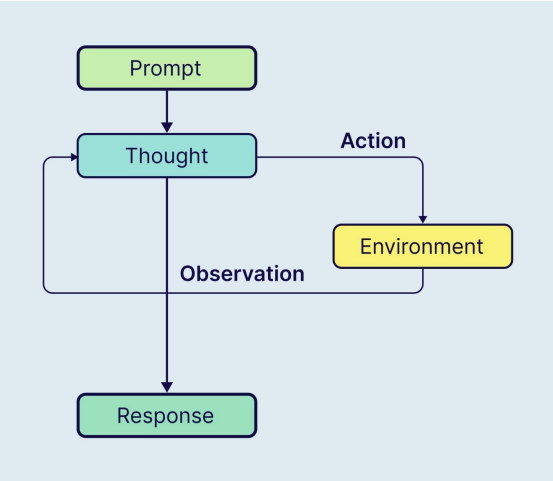


Figure 2.9: ReAct prompting, from [70]).

2.6.2 Design Considerations

Prompt engineering is not merely a cosmetic aspect of RAG, but a core enabler of reliable and explainable generation. It determines how retrieved evidence is interpreted and how the final response is framed. By selecting appropriate prompting strategies, carefully designing prompt templates, and employing iterative refinement, developers can significantly enhance the factuality, coherence, and usability of RAG-based outputs [31, 38].

Designing effective prompts in a RAG architecture involves several considerations [31,38,62]:

- **Context formatting:** Retrieved documents must be clearly separated from instructions and examples, often using delimiters or tags to avoid confusing the model [31].
- **Token management:** Since input length is constrained, prompts must balance retrieved content, instructions, and examples to remain within model limits [38].
- **Domain adaptation:** Prompts should reflect the terminology and structure expected in the target domain, especially when dealing with legal, medical, or technical documents [62].
- **Dynamic prompting:** Advanced systems may use prompt templates that dynamically adapt to the user query or retrieval outcome, generating context-aware instructions on the fly [62].
- **Instruction clarity:** Explicit and unambiguous instructions help guide the model toward the intended behavior, reducing the risk of misinterpretation or irrelevant outputs [31].
- **Prompt modularity:** Separating instructions, context, and examples into reusable prompt modules can simplify maintenance, enable consistent performance across tasks, and facilitate rapid adaptation to new queries.

- **Bias mitigation:** Prompt design can influence model outputs; careful wording and inclusion of counterexamples may reduce the risk of biased or inappropriate responses [62].
- **Iterative refinement:** Prompts should be tested and refined iteratively based on model outputs and evaluation metrics, ensuring that the RAG system meets accuracy, coherence, and usability requirements over time.

As RAG systems evolve toward more interactive, adaptive, and multimodal architectures, the role of prompt engineering will continue to grow in importance. Future developments may involve automatic prompt optimization, personalized prompts for user-specific contexts, and integration with external verification mechanisms to further improve reliability and transparency.

2.7 Agentic Architectures

The evolution of Large Language Models from static, reactive systems to dynamic agentic architectures represents a paradigm shift in artificial intelligence. Traditional LLM applications follow predetermined control flows, such as fixed sequences of operations where the model simply reacts to inputs. For instance, in basic Retrieval-Augmented Generation systems, the workflow is linear: retrieve documents based on a query, then generate a response. While effective for simple tasks, this static approach severely limits the model's problem-solving capabilities, particularly when faced with complex, multi-faceted challenges that require dynamic decision-making. Agentic behavior fundamentally transforms this paradigm by empowering LLMs to determine their own control flows. An agentic LLM becomes an autonomous entity capable of making contextual decisions about which steps to execute, which tools to employ, and when to terminate a process. This shift from reactive to proactive systems enables LLMs to tackle sophisticated tasks requiring adaptability, strategic planning, and iterative refinement, that represent capabilities that mirror human problem-solving processes [40, 57].

2.7.1 From Chains to Autonomous Agents

The progression toward agentic behavior in LLM systems can be understood as an evolution from predefined process flows, commonly referred to as *chains*, to fully autonomous decision-making frameworks [38, 62, 68]. In their simplest form, chains are linear sequences of operations where each step's output becomes the next step's input. For example, in a basic question-answering chain, the system might first retrieve relevant documents, then summarize them, and finally generate a final answer [31]. While effective for well-structured tasks, such linear flows are inherently rigid and fail to adapt when the task deviates from the expected pattern.

More advanced configurations, such as tree chains and router chains, introduce conditional branching. Tree chains can follow multiple paths depending on intermediate results, while router chains incorporate a decision-making component that selects the most appropriate subchain based on the input's characteristics [62]. This represents a step toward flexibility, as the system begins to make simple choices rather than strictly following a fixed route.

The transition to *true* agentic behavior occurs when an LLM is no longer limited to selecting among predefined paths but can instead design its own course of action. An autonomous agent can dynamically plan multi-step strategies, modify its approach in response to unexpected outcomes, and recursively refine its reasoning until the goal is achieved [62, 68]. This leap transforms the system from a deterministic executor into a proactive problem-solver capable of operating in open-ended, unpredictable environments.

2.7.2 Core Components of Agentic Architectures

The architecture of an AI agent comprises three fundamental subsystems that work in concert to enable autonomous operation [61, 71].

1. **Large Language Model:** It serves as the central reasoning engine, combining the model's parametric knowledge with dynamic decision-making capabilities. Unlike traditional LLMs that simply process inputs, the agentic core maintains a persistent sense of task and role, allowing it to approach problems with intentionality. This core engages in both planning, breaking down complex objectives into actionable subgoals, and reflection, critically evaluating past actions to improve future performance.
2. **Memory:** Memory systems provide the agent with temporal context, divided into short-term and long-term memory components. Short-term memory functions as a working buffer, retaining immediate context such as conversation history or intermediate computation results. Long-term memory operates as a persistent knowledge store, enabling the agent to learn from accumulated experience and maintain personalized user profiles across interactions. This dual-memory architecture allows agents to balance immediate task requirements with longitudinal learning.
3. **Toolkit:** It represents the agent's interface with external systems and specialized capabilities. Common tools include vector search engines for information retrieval, web search APIs for real-time data access, calculators for precise computations, and code interpreters for executing programming tasks. These tools dramatically expand the agent's operational scope beyond its parametric knowledge, enabling it to perform tasks that would otherwise require human intervention or specialized software.

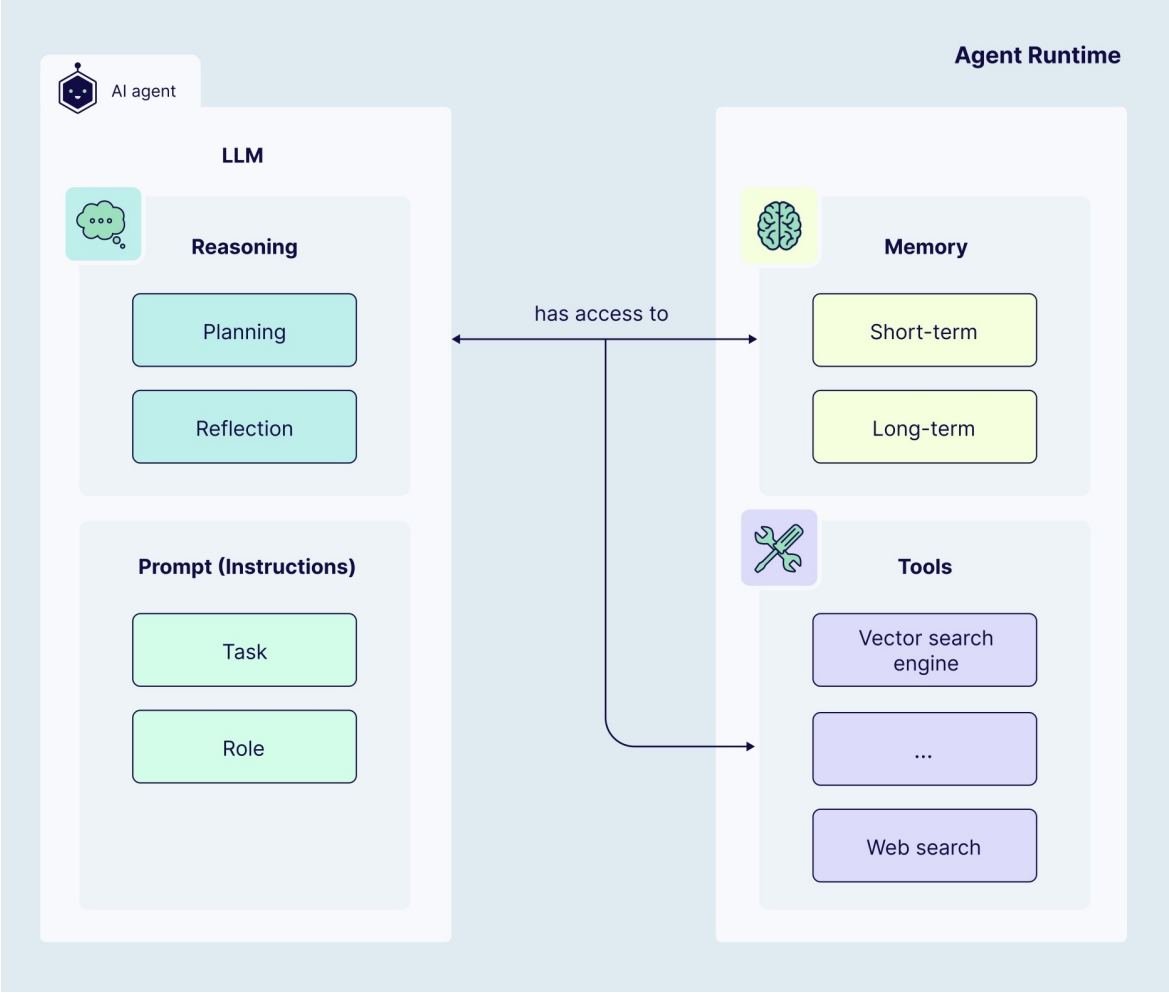


Figure 2.10: Core components of AI agents, from [71].

2.7.3 Reasoning Mechanisms and Knowledge Utilization

Agentic systems employ sophisticated planning strategies to decompose complex objectives into executable subgoals. The one-step planning method breaks tasks into sequential subtasks through a single comprehensive reasoning process. While effective for well-structured problems, this approach lacks adaptability when faced with unexpected challenges or ambiguous requirements. Multi-step reasoning addresses this limitation through iterative planning cycles. The agent generates partial plans, executes initial steps, evaluates outcomes, and then refines subsequent actions based on accumulated context. This recursive process continues until the objective is achieved or termination conditions are met [75]. The system maintains coherence through mechanisms like chain-of-thought reasoning, where each decision is explicitly justified, and subgoal decomposition, which ensures modular, manageable task components [13]. Reflection and self-critique mechanisms further enhance planning quality.

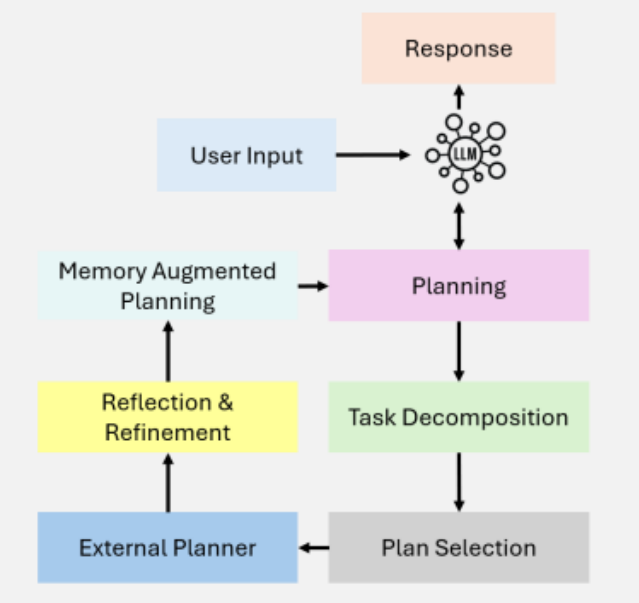


Figure 2.11: Overview of agentic planning, from [61].

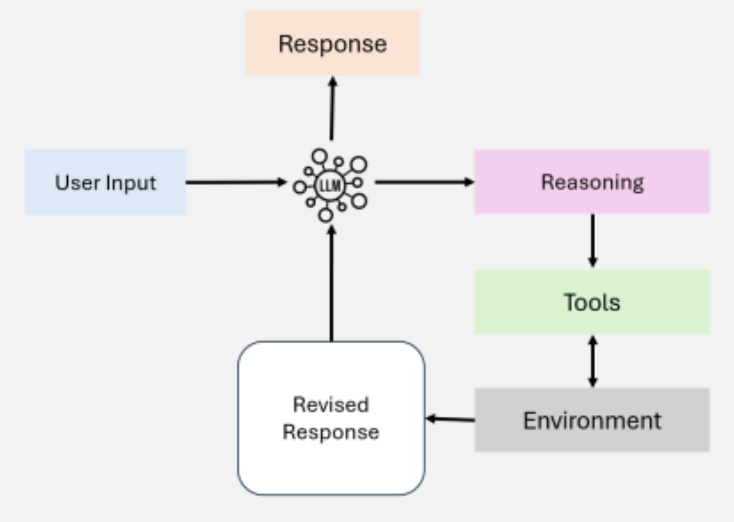


Figure 2.12: Overview of tool use, from [61].

After action execution, agents analyze performance outcomes, identifying successful strategies and areas for improvement. This meta-cognitive capability allows for continuous adaptation, with insights being codified into both short-term context for immediate task adjustment and long-term memory for future performance enhancement [19].

The memory architecture of agentic systems operates through three interconnected processes: storage, retrieval, and reflection. Storage mechanisms handle information persistence, employing strategies ranging from simple addition of new facts to sophisticated merging of related concepts and correction of outdated knowledge. The system dynamically selects storage methods based on semantic similarity assessments and task requirements [74]. Retrieval processes are equally nuanced, varying significantly between short-term and long-term memory access. Short-term retrieval typically recalls entire context windows, while long-term retrieval employs semantic filtering to identify the most relevant historical information. Knowledge utilization extends beyond internal memory to incorporate external information sources. Agentic systems integrate database queries, knowledge graph traversals, and real-time web scraping to maintain information freshness. Retrieval-Augmented Generation techniques are particularly valuable, combining the breadth of external knowledge with the generative capabilities of LLMs. The system actively manages knowledge quality through validation mechanisms that identify and correct hallucinations, biases, and outdated information [71].

2.7.4 Multi-Agent Architectures

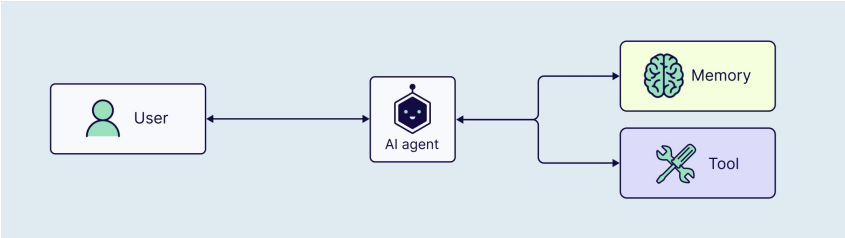


Figure 2.13: Overview of a single-agent architecture, from [71].

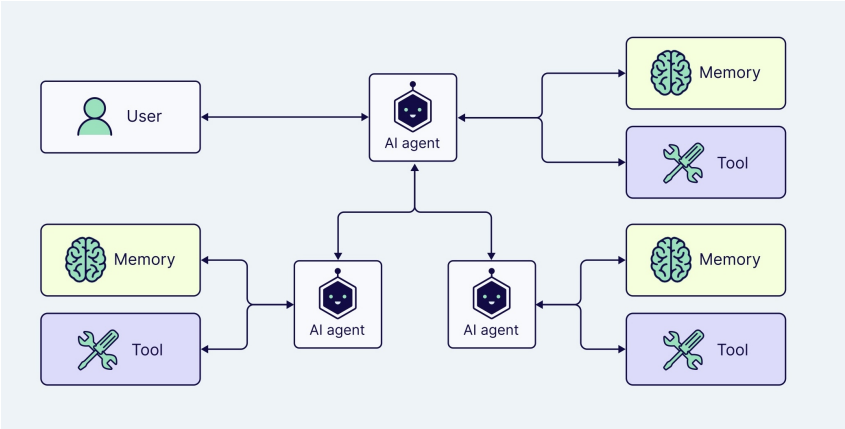


Figure 2.14: Overview of a multi-agent architecture, from [71].

While single-agent architectures are capable of managing a broad range of tasks, they rely on a single reasoning engine to oversee the entire problem-solving process. This centralization offers advantages in terms of simplicity, predictable control flow, and reduced coordination overhead. A single agent can operate with minimal latency, as there is no need for inter-agent communication, and resource usage is generally lower. However, this approach inherently limits

scalability and specialization. When tasks require expertise in multiple domains, simultaneous processing of subtasks, or the adaptation of different reasoning strategies, single-agent systems can become a bottleneck. Furthermore, they represent a single point of failure: if the agent encounters limitations in reasoning or tool use, the entire process is hindered.

Multi-agent architectures [73] overcome many of these constraints by distributing responsibilities among several autonomous agents, each potentially specialized in a particular skill set, reasoning style, or access to tools. This division of labor allows for greater scalability and adaptability, enabling tasks to be handled in parallel and solutions to emerge from cooperative or competitive interactions between agents. The modular nature of multi-agent systems facilitates the integration of new capabilities without redesigning the entire architecture. Despite these strengths, multi-agent systems introduce additional challenges [61]. Coordination between agents requires robust communication protocols, which can increase latency and computational overhead. Decision-making may be complicated by conflicts between agents or redundant efforts, and system orchestration becomes inherently more complex. As a result, the benefits of specialization and parallelism must be weighed against the costs of synchronization and management. Common multi-agent design patterns include:

- **Sequential**: Agents execute in a fixed order, passing intermediate outputs along the chain.

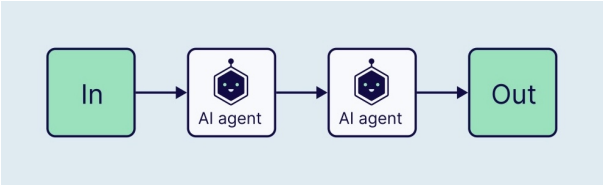


Figure 2.15: Sequential pattern of multi-agent architectures, from [71].

- **Parallel**: Multiple agents work independently on different subtasks, with results combined at the end.

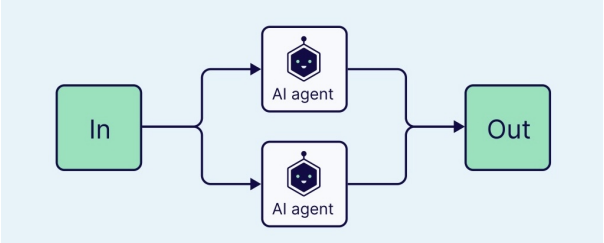


Figure 2.16: Parallel pattern of multi-agent architectures, from [71].

- **Loop**: Agents iteratively refine each other's outputs until a stopping condition is met.

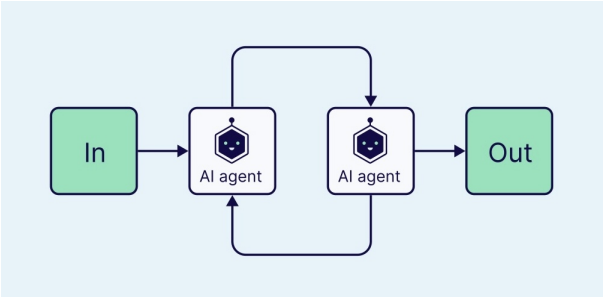


Figure 2.17: Loop pattern of multi-agent architectures, from [71].

- **Router**: A dispatcher agent routes tasks to specialized agents based on task characteristics.

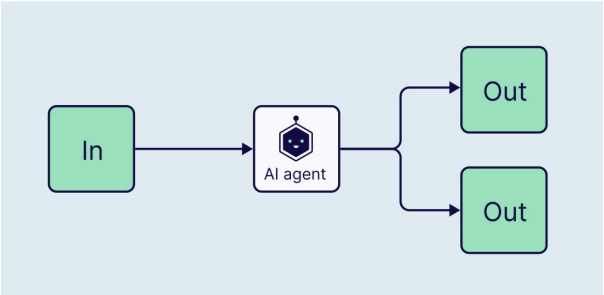


Figure 2.18: Router pattern of multi-agent architectures, from [71].

- **Aggregator**: Multiple agents produce outputs that are combined or ranked by a central agent.

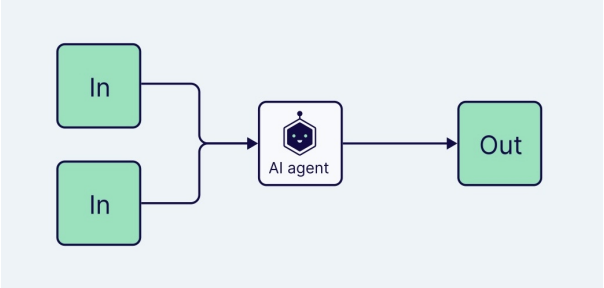


Figure 2.19: Aggregator pattern of multi-agent architectures, from [71].

- **Network**: Agents form a mesh topology, exchanging information dynamically without a central controller.

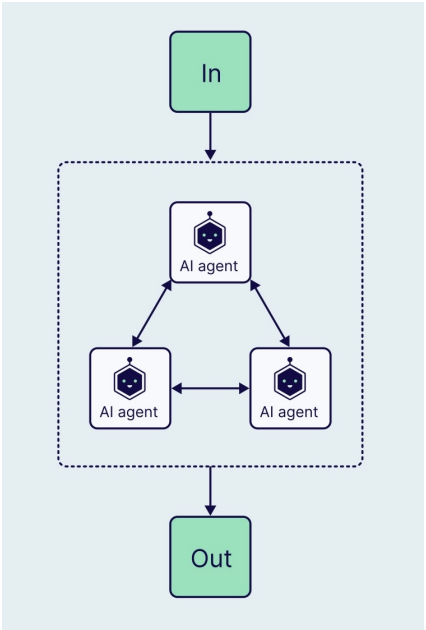


Figure 2.20: Network pattern of multi-agent architectures, from [71].

- **Hierarchical**: Higher-level agents delegate tasks to lower-level ones and consolidate their results.

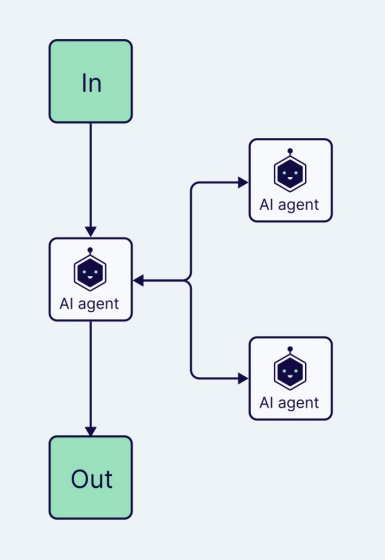


Figure 2.21: Hierarchical pattern of multi-agent architectures, from [71].

2.7.5 Single-Agent RAG Architecture

In a single-agent Retrieval-Augmented Generation architecture [69, 71], a single multipurpose agent is responsible for both retrieving the information required to answer a user query and generating the final response. This design centralizes the reasoning and tool-use capabilities within one agent, which manages the entire process from query analysis to response generation.

The workflow begins when the agent receives a user query. As a first step, the agent checks its memory to determine whether the same or a similar question has already been answered. If a suitable answer exists, the agent can directly reuse it, ensuring efficiency and consistency. If no prior answer is available, the agent evaluates whether additional information is required to address the query. When more information is needed, the agent can analyze the query and, if it is complex, decompose it into simpler sub-queries. This *query decomposition* process is particularly useful for multifaceted questions that may require consulting different knowledge sources. By breaking down the query, the agent can obtain more accurate and relevant retrieval results.

Each (sub-)query is then routed to the most appropriate knowledge source through a *query routing* step. The agent decides which external source to use (e.g., a vector database or a web search tool), and may also apply additional parameters such as metadata filters. In order to interact effectively with the chosen source, the query may undergo a *query transformation*, which reformulates it into the correct format for the retrieval system (for example, converting it into an embedding for a vector search, or into keywords for a traditional search engine).

Once information has been retrieved, the agent performs an *evaluation* step. Here, it assesses whether the evidence is relevant to the original query and whether it is sufficient to provide a reliable answer. If the retrieved content is inadequate or incomplete, the agent may revise its decomposition or retrieval strategy and attempt the process again. Finally, once enough relevant information has been collected, the agent synthesizes an answer. This response is generated by combining the original query with the retrieved evidence. The resulting output is then delivered to the user, and optionally stored in memory for future reuse.

Overall, the single-agent RAG architecture [61] emphasizes an iterative process of reasoning, retrieval, evaluation, and generation, all coordinated by a single agent. This approach reduces

system complexity compared to multi-agent solutions, while still enabling accurate and grounded responses.

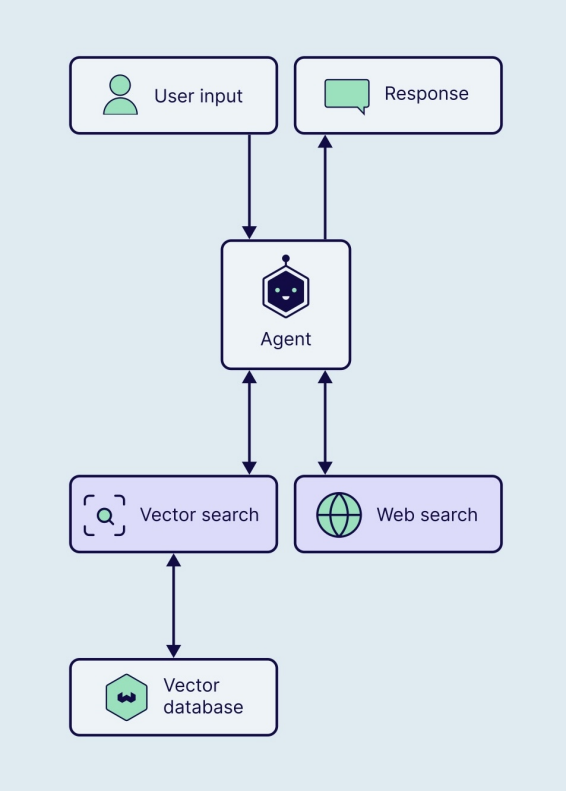


Figure 2.22: Single-agent RAG architecture, from [71].

2.8 Conclusion

This chapter has outlined the fundamental concepts and technical foundations required to understand the subsequent work presented in this thesis. We began by introducing the principles of Large Language Models, their representational power through embeddings, and the architectural underpinnings that enable their advanced reasoning capabilities. We then examined the evolution from traditional sequence models, such as Recurrent Neural Networks, to the Transformer architecture, highlighting how attention mechanisms and parallel processing have reshaped the field of Natural Language Processing. Finally, we explored the emerging paradigm of agentic LLMs, detailing the transition from static pipelines to autonomous decision-making systems, and the design considerations underlying both single-agent and multi-agent architectures. In addition to mapping the theoretical landscape, this chapter has emphasized the practical challenges that accompany these advances, including issues of evaluation, interpretability, ethical considerations, and the persistent problem of hallucinations. These limitations are not merely academic concerns but critical obstacles that must be addressed when deploying LLM-based systems in sensitive contexts, such as the medical and administrative domains targeted in this work.

Together, these elements establish a comprehensive theoretical background for the case study and methodology described in the following chapters. The concepts presented here will serve as the basis for understanding how modern LLM-based agents can be designed, orchestrated, and applied to address complex, real-world challenges. In particular, the next chapters will illustrate how the Retrieval-Augmented Generation (RAG) approach and agentic design principles can be combined to build intelligent systems that are both adaptable and trustworthy. By bridging

theoretical insights with applied design, the thesis aims to demonstrate not only the technical feasibility but also the broader implications of deploying autonomous AI agents in critical, knowledge-intensive domains.

Chapter 3

System Architecture

This chapter presents the overall architecture of the developed system, describing its requirements, both functional and non-functional, the logical structure, the main components, and the interactions among the modules that constitute its operation. The system was designed following a modular and scalable architecture, which logically separates responsibilities among the various components. This separation enables high code maintainability, greater flexibility during development and testing, and facilitates future system extensions.

In addition, the chapter also illustrates the specific tools and technologies that were employed to implement the system, providing a practical perspective on how the architecture was realized in concrete terms.

3.1 Requirements

This section outlines the main functional and non-functional requirements that the system must satisfy to ensure behavior consistent with the desired objectives and user expectations. The designed and implemented system aims primarily to assist in the consultation and generation of administrative resolutions through the use of a Retrieval-Augmented Generation architecture enhanced by Large Language Models. Consequently, the definition of requirements has been guided by the need to guarantee response accuracy, compliance with the formal structure of documents, and the secure handling of sensitive content.

Moreover, each functionality has been designed to operate within clearly defined rules and constraints, ensuring that actions such as document generation, modification, and scheduling adhere to established workflows, maintain consistency across different modules, and prevent the introduction of unauthorized or unverifiable information. This approach ensures that the system behaves in a predictable and reliable manner, supporting both efficiency and trustworthiness in administrative operations.

3.1.1 Functional Requirements

From a functional perspective, the system must effectively manage user interaction by performing semantic retrieval operations and automatic text generation based on the retrieved documents. Specifically, the system is required to:

- Accept as input a textual request from the user, which may consist of a question, a clarification request, or a prompt for generating a resolution;
- Perform efficient semantic search within a pre-loaded corpus of resolutions, identifying the most relevant textual segments based on vector similarity;
- Use the retrieved content as context to feed an LLM and generate a coherent, relevant, and stylistically appropriate response;

- Automatically censor sensitive information not present in the documents, such as names of persons or organizations, by using dedicated placeholders;
- Distinguish between informational requests and structured generation requests, selecting the most appropriate behavior accordingly;
- Enable automatic generation of a complete resolution starting from a structured input, faithfully reproducing the form, tone, and organization of official documents;
- Provide a scheduling functionality that allows users to plan the generation of one or more resolutions in advance by specifying a completion date and one or more requests;
- Produce, following text generation, a `.docx` document with pagination and formatting compliant with the standards of real resolutions, ready for review by a human operator.

3.1.2 Non-Functional Requirements

In addition to the main functionalities, the system must satisfy a set of non-functional requirements essential to ensure quality, reliability, and usability. In particular, the system is required to:

- Guarantee the accuracy of the generated responses, avoiding hallucinations and relying exclusively on the context provided by the RAG system;
- Maintain a formal and technical tone appropriate for the administrative-legal domain, both in textual responses and generated documents;
- Be modular and easily extensible, allowing integration of new document sources, updating the language model, or modifying prompts;
- Operate in a deterministic mode, with a low generation temperature to reduce output variability;
- Comply with privacy and data protection principles, avoiding generation of personal or unverifiable data and implementing explicit anonymization techniques;
- Provide an acceptable response time that is sufficiently fast to ensure smooth and uninterrupted interaction, maintaining a level of performance compatible with real-time use by an end user;
- Ensure reliability and robustness, so that scheduled batches are executed at the correct time without loss or duplication;
- Produces readable documents that fully comply with established typographic standards, ensuring consistent formatting, and immediate readiness for practical use.

Overall, these requirements define the technical and operational constraints within which the system was developed, guiding both the initial design phase and subsequent optimization and validation iterations.

3.2 Main Components

The system architecture is composed of four primary macro-components that cooperate to process user inputs, retrieve relevant data, and generate formal administrative documents. These components are the *Frontend*, the *Backend*, the *AI Engine*, and the *Document Storage*.

Figure 3.1 illustrates the high-level architecture of the system.

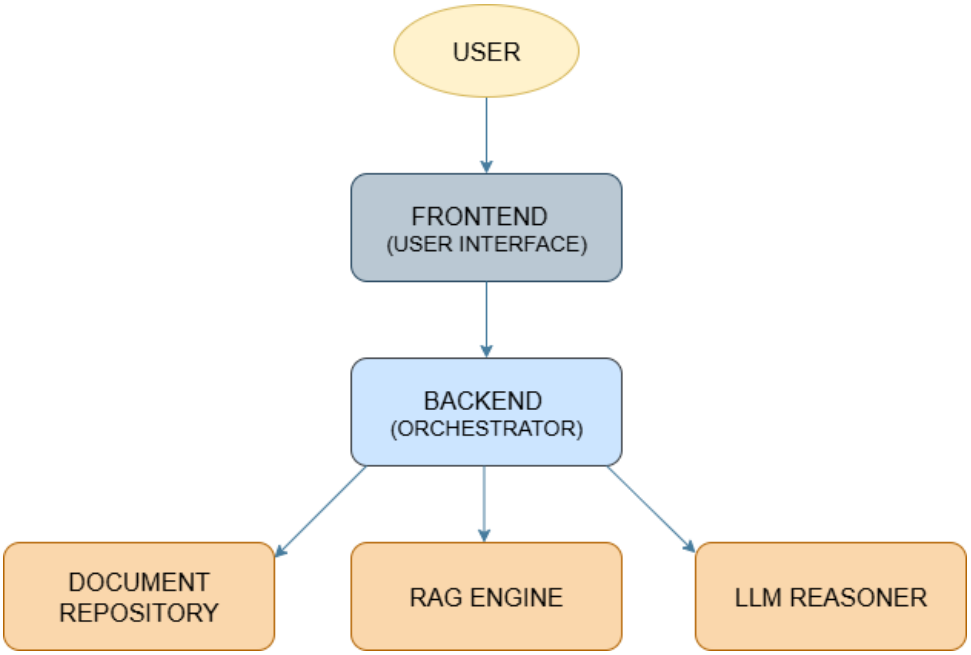


Figure 3.1: Architectural diagram of the application.

3.2.1 Frontend

The frontend constitutes the user interface layer and serves as the primary point of interaction between the end user and the system. It is implemented as a responsive web application using standard technologies such as `HTML5`, `CSS3`, and `JavaScript` (with frameworks like `Vue.js` or `React` where applicable). The frontend handles both the collection of user inputs and the rendering of outputs produced by the backend and AI engine.

The key functionalities offered to the user include:

- Input of free-text prompts, questions, or requests for document generation;
- Visualization of the system's responses in real time, including rich formatting;
- Access to the history of previous conversations stored per user session;
- Scheduling of documents by specifying a target completion date;
- Export and download of generated documents in `.docx` format;
- Search, filtering, and management of previously created administrative resolutions.

The frontend is stateless and communicates with the backend via asynchronous HTTP requests (REST API), enabling modularity and scalability.

3.2.2 Backend

The backend is the central orchestrator of the system's logic and it's implemented in `Python`, leveraging frameworks such as `FastAPI` or `Flask` for the exposure of RESTful services. It acts as a middleware layer, interfacing with the frontend, AI services, storage systems, and logging infrastructure.

Its core responsibilities include:

- Validation and normalization of input prompts;
- Invocation of the semantic search engine via embedding-based queries;

- Construction of structured prompts for the LLM, following specific templates and adhering to design patterns;
- Integration of LLM outputs with domain-specific rules (e.g., placeholders, formatting constraints);
- Management of scheduled documents, including assignment of time slots, persistence of requests in the database, and asynchronous processing of batches;
- Generation of `.docx` documents, applying predefined templates compliant with legal-administrative standards;
- Real-time monitoring and logging of all requests and responses for auditability and performance analysis.

The backend is designed following a microservices-oriented approach, enabling future extension of capabilities (e.g., validation services).

3.2.3 AI Engine

The AI engine constitutes the semantic and generative core of the platform and it's composed of two tightly integrated submodules: the Semantic Retriever and the Large Language Model.

Semantic Retriever. This module is built upon Vertex AI Vector Search, which enables efficient vector similarity search at scale. Each document in the corpus is pre-processed into coherent text chunks and transformed into dense vector representations using the embedding model `text-embedding-005`. At runtime, the user query is embedded and matched against this index using cosine similarity or dot product, retrieving the *top-k* most relevant contexts.

Generative Module (LLM). Once the retrieval phase concludes, the retrieved contexts are integrated with the original user prompt to form a structured input for the LLM. The language model generates a coherent, well-structured, and legally compliant response. The output is post-processed to incorporate placeholders, ensure formatting consistency, and detect potential ambiguities or missing information. This architecture enables the implementation of a full Retrieval-Augmented Generation pipeline, enhancing both factual accuracy and domain specificity.

3.2.4 Document Storage

The storage layer contains both the static knowledge base (i.e., the corpus of administrative resolutions) and the dynamically generated content (e.g., user sessions and finalized documents). The corpus is stored in chunked format, with each chunk linked to metadata such as document ID, title, creation date, and topic. The vector representations are indexed separately in the semantic engine, while the textual data is stored on `Google Cloud Storage` (GCS), offering scalable, low-latency access. Dynamically generated documents, such as finalized resolutions or interaction logs, are stored in dedicated buckets with versioning and access control mechanisms. This ensures that all documents are properly tracked over time, with previous versions preserved for auditing and rollback purposes, while access is restricted according to user roles and security policies. The storage layer is designed to provide not only reliable persistence but also a set of enhanced functionalities, including:

- Real-time updates of the vector index in response to document modifications;
- Secure download and export in standardized formats;
- Data analytics for usage monitoring and system improvement.

3.3 Technological Infrastructure

The reference architecture adopted for the system implementation is designed to support a generative AI application based on the Retrieval-Augmented Generation approach, leveraging the scalable and fully managed infrastructure of Google Cloud [23]. Specifically, the solution exploits services provided by Vertex AI and Vector Search to guarantee advanced semantic search capabilities and controlled content generation.

The infrastructure is organized as a modular architecture composed of two main subsystems: the data ingestion subsystem and the publishing subsystem, each responsible for a distinct phase of the RAG pipeline. The data ingestion subsystem manages the entire process of acquisition, preprocessing, embedding, and indexing of data, thus preparing the information for efficient querying via vector search. The publishing subsystem, instead, handles the interaction flow with the end user, orchestrating natural language generation requests through retrieval of relevant data and querying of the language model.

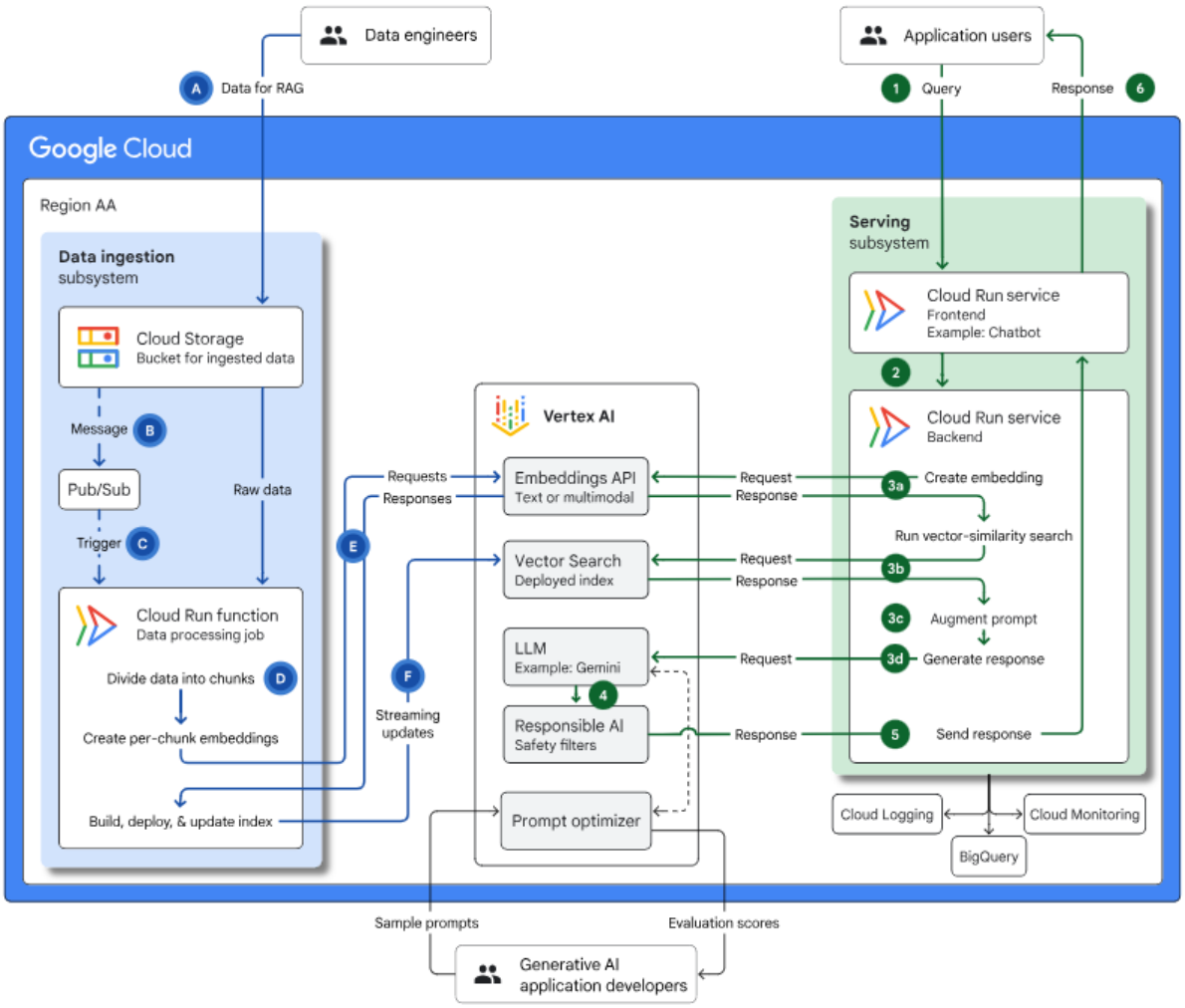


Figure 3.2: Detailed visualization of the system architecture, from [24].

3.3.1 Google Cloud Services Utilized

The infrastructure relies on the following main services from the Google Cloud platform:

- **Vertex AI**: a comprehensive ML platform for training, deploying, and managing AI models, including customized LLMs.

- **Vector Search**: a service for large-scale vector similarity indexing and search, designed to support semantic search scenarios and efficient retrieval over large embedding collections.
- **Cloud Run**: a serverless service for running scalable containers, enabling secure web service exposure with automatic resource management.
- **Cloud Storage**: a high-availability and durable object storage system, used for storing static files, documents, and models.
- **Cloud Scheduler**: a fully managed service for scheduling cron jobs, used to execute recurring operations (e.g., periodic vector index updates or synchronizations).
- **Cloud Logging**: a system for real-time log collection, aggregation, and analysis from various system components, useful for debugging and auditing.
- **Cloud Monitoring**: a platform for monitoring metrics, setting alerts, and analyzing the health status of applications and infrastructure.
- **Firestore**: a serverless, highly scalable NoSQL document database used for structured storage of application data, conversation histories, and system configurations.

This architecture represents a scalable, modular, and highly integrable solution for implementing intelligent agents based on advanced language models, capable of providing contextualized and reliable responses from dynamic and heterogeneous document bases.

3.3.2 Data Ingestion Subsystem

The ingestion and preparation of data is performed through an automated pipeline that transforms unstructured data from external sources into semantically meaningful vector representations. The operational sequence is as follows:

1. Data are initially uploaded to a Cloud Storage bucket, serving as a centralized collection point for files from applications, databases, or data streams.
2. Cloud Scheduler periodically triggers a monitoring function that checks for new files in the bucket.
3. Upon message receipt, a Cloud Run job is activated that parses and formats the content, dividing it into coherent informational blocks.
4. Each block is transformed into a vector embedding via the Vertex AI Embeddings API, using a configurable embedding model (textual or multimodal).
5. The embeddings are finally used to build a semantic index through Vertex AI Vector Search, which is then deployed for efficient search. When new data arrive, the index is dynamically updated via incremental update mechanisms.

3.3.3 Publishing Subsystem

The publishing subsystem is responsible for handling user queries and generating responses via a large language model. The operational flow includes the following steps:

1. The user interacts with the application through a frontend interface exposed via a Cloud Run service.
2. The user query is forwarded to a backend service, also implemented as a Cloud Run component.

3. The backend converts the query into an embedding using the same model employed during data ingestion, ensuring semantic consistency between queries and indexed documents.
4. The system performs a vector similarity search on the index managed by Vertex AI Vector Search, identifying the most relevant informational blocks for the query.
5. The original query is augmented with the retrieved grounding documents and transformed into an enriched prompt, which is then sent to an LLM deployed on Vertex AI.
6. The model generates a response, which is filtered by responsible AI mechanisms applying security and moderation policies configured at the system level.
7. The response is finally returned to the user via the frontend and, if necessary, archived in Cloud Logging for monitoring and interaction analysis.
8. Conversation metadata, including timestamps, user IDs, session IDs, and structured logs, are saved in Firestore to guarantee traceability, auditability, and support for advanced features such as history and aggregated analysis.

3.4 Application Design and Implementation

This section describes the design choices and implementation strategies adopted to translate the proposed architecture into a functional system.

3.4.1 Application Workflow Orchestration

To orchestrate the overall application workflow, Google Cloud offers several services, each with a generous free tier:

- **Cloud Composer**: a managed service for orchestrating workflows based on Apache Airflow.
- **Cloud Workflows**: a serverless service for defining sequential pipelines.
- **Cloud Scheduler**: schedules jobs at regular intervals (free for up to three jobs per month per account).
- **Cloud Pub/Sub**: an asynchronous messaging system suitable for event-driven workflows, such as reacting to new files in `Cloud Storage`.

Among these options, we selected Cloud Scheduler due to its ease of implementation, fast activation, and low cost, while maintaining a good balance between scalability and control.

3.4.2 Development of the Application Interface

For the implementation of the frontend interface, several JavaScript frameworks were considered:

- **React**: a versatile solution suitable for small to medium-sized projects.
- **Angular**: more appropriate for complex, large-scale applications.
- **Svelte 5**: a high-performance framework recommended when speed and responsiveness are critical, although its community is currently more limited.

The main difference between React and Svelte 5 lies in their respective reactivity and state management models. Svelte 5 introduces a signals-based system that allows for more efficient UI updates compared to React's traditional virtual DOM approach.

Among the evaluated alternatives, React was selected as it provides a good compromise between simplicity, modularity, and scalability.

3.4.3 Backend Integration and Authentication

The backend of the application can be developed using different frameworks depending on the complexity of the project:

- **Flask**: suitable for simple or prototypical projects.
- **FastAPI**: an excellent compromise between performance and ease of use.
- **Django Rest Framework** and **Spring Boot**: ideal for large-scale applications requiring advanced features and high scalability.

For user management, credentials and user data can be stored in `Cloud Firestore`, a highly scalable, document-based NoSQL database with low operational costs.

Among these alternatives, we chose to adopt FastAPI, as it represents an excellent balance of high performance, ease of development, and maintainability. FastAPI is designed to provide a modern development experience, featuring native support for automatic data validation, OpenAPI documentation generation, and asynchronous integration, all of which are particularly advantageous in a cloud-native and serverless context such as that enabled by Cloud Run.

Chapter 4

Methodology

This chapter provides a detailed description of the technical workflow underlying the system, illustrating each stage of the processing pipeline. Beginning with the pre-processing of input documents, the discussion covers text normalization, segmentation into chunks, and generation of semantic embeddings, that represent essential components for the subsequent vector-based retrieval phase.

The functioning of the semantic retrieval engine is then examined in depth, with particular attention given to the indexing mechanisms and the configuration of the parameters governing similarity search within the embedding space. Finally, the integration of the Large Language Model within the system is analyzed, both in the context of responding to informational queries and in the automatic generation of structured documents. The strategies adopted to guide and constrain the model's output in formal and domain-specific contexts are also discussed.

4.1 Preliminary Document Analysis

The institution made available the administrative resolutions produced from the year 2021 onward. However, for the purposes of this project, it was decided to focus exclusively on documents issued in 2023 and 2024, based on discussions with current personnel at the institution. This choice was motivated by the observation that the relevant regulatory frameworks are subject to frequent updates, significantly affecting the structure and content of the resolutions. As a result, the most recent documents are more representative of current practices, having been drafted in accordance with the latest stylistic and regulatory conventions, and exhibiting greater formal consistency and editorial quality.

In order to gain a detailed understanding of the structure and content of these documents, an exploratory analysis was conducted with the aim of identifying recurring patterns and structurally significant sections. The analysis was focused on resolutions issued in 2024, excluding amendments and documents classified as rectified, resulting in a total of 395 resolutions. This filtering ensured a homogeneous and representative dataset reflective of standard production.

The document review revealed that, despite a degree of variability in structure and content, primarily attributable to individual authorship, the resolutions consistently share a set of common elements that characterize their internal organization.

- **First page:** This is consistent across all analyzed resolutions and contains information pertaining to the proposal, responsible parties, and the object of the resolution. The object typically follows the structure `<topic>: <action undertaken by the resolution>`.
- **Central section:** This comprises a sequence of paragraphs outlining the legal and administrative context of the resolution, along with the rationale and intended actions. Although this section does not adhere to a fixed template and varies significantly across documents, it frequently exhibits a recurring structure in the form of introductory keywords

at the beginning of paragraphs, such as "*Richiamati*", "*Visti*", "*Preso atto*", "*Ricordato*", "*Ritenuto*", "*Dato atto*", among others.

- **Final section:** This consists of a numbered list that formally specifies the approved decisions and actions, along with their associated costs. This section serves as a consolidation and summary of the elements previously discussed in the narrative portion of the document.

To streamline the subsequent phases of analysis and modeling, the resolutions were manually classified into 21 thematic categories according to their content similarity. This grouping facilitated more efficient data handling and enabled a more focused assessment of the distinctive features associated with each document cluster.

4.2 Semantic Processing of Documents

Following the preliminary analysis phase, a transformation process was carried out to convert textual documents into numerical representations, which are essential for the subsequent stages of semantic retrieval and generation. This process involved a series of operations including data cleaning, text pre-processing, chunking, and finally, the generation of embeddings using dedicated models.

4.2.1 Text Preprocessing

The documents were subjected to a text preprocessing pipeline to ensure adequate data quality for input into embedding models. The following steps were performed:

- **Optical Character Recognition (OCR)**: The open-source engine Tesseract [63] was used to extract text from image-based documents or non-textual PDFs.
- **Document cleaning**: Irrelevant or redundant elements were removed, including the logo of the institution, digital signatures, page numbering, and attachments.
- **OCR error correction**: Systematic errors produced by Tesseract (e.g., the word "*acquisiti*" being incorrectly recognized as "*aecquisiti*") were identified and corrected using manual inspection and regular expressions.
- **Text normalization**: This included converting text to lowercase, removing non-ASCII and non-alphanumeric characters, and eliminating redundant empty lines.
- **NLP preprocessing**: Additional processing was applied using SpaCy [29] and NLTK [6] pipelines to remove Italian stopwords and lemmatize tokens.

4.2.2 Chunking Strategies

To make the text suitable for embedding, it was divided into segments, called chunks, of sizes compatible with the tokenization limits of the models, while preserving semantic coherence as much as possible.

Several text segmentation strategies were explored, evaluating their effectiveness in terms of semantic granularity and informativeness:

- **Fixed-size sliding window**: Splitting text into fixed-size windows of characters or tokens with partial overlap. This method was simple to implement but ineffective, as it ignored both structure and meaning.

- **Paragraph/sentence splitting**: Chunking based on punctuation to extract full sentences or paragraphs. However, this often resulted in fragments that were too short or semantically sparse.
- **Recursive structure-aware splitting**: A hierarchical segmentation based on grammatical and logical structures. However, this too suffered from limited contextual coherence and excessive fragmentation.
- **Content-aware splitting**: A custom implementation designed specifically for the document domain, based on the following rules:
  - The entire first page is treated as a single chunk;
  - Paragraphs in the central section are individually segmented using recurring keywords such as "*Richiamati*", "*Visti*", "*Preso atto*", etc.;
  - The final section is treated as a single chunk. If it exceeds 15,000 characters, it is further split into its constituent paragraphs.

This last strategy proved most effective at preserving both semantic coherence and the logical structure of the documents, overcoming limitations related to the lack of visual separators such as vertical spacing or HTML layout.

Using this custom content-aware strategy, each document yielded an average of 19 chunks, from which embeddings were generated.

4.2.3 Embedding Generation

For each chunk extracted from the documents, a vector embedding was generated using Google Cloud Vertex AI's `text-embedding-005` model [22]. This model produces a dense numerical representation of 768 dimensions for each input text, optimized for semantic similarity tasks.

Each embedding was enriched with metadata about the originating chunk, including:

- The associated document ID;
- Chunk length (in characters and tokens);
- Relative position within the document.

These metadata facilitated subsequent stages of semantic retrieval, traceability, and qualitative evaluation.

4.2.4 Embedding Distribution with Principal Component Analysis

After generating embeddings for each chunk of all documents, an exploratory analysis was performed to understand the semantic distribution of the resolutions in vector space. The goal was to assess the consistency of the manually defined clusters presented in Table 5.1, that we'll see later, and to identify similarities or anomalies across different groups.

The analysis focused on a sample of 29 resolutions from 2024, selected to represent the existing groups uniformly. For each document, the embeddings of its chunks were considered, each being a vector with 768 real components. These embeddings were aggregated using different strategies to obtain fixed-size document representations.

Formally, let $D = \{d_1, \ldots, d_{29}\}$ be the set of documents, and for each document d_i, with index $i \in \{1, \ldots, 29\}$, let $j \in \{1, \ldots, n_i\}$ be the index of its chunks, where n_i denotes the number of chunks into which document d_i has been divided (approximately 19 on average).

In addition, let

$$\mathbf{h}_{i,j} \in \mathbb{R}^{768}, \quad j = 1, \ldots, n_i$$

be the embedding of the j-th chunk of document d_i. The goal is to obtain a single representation

$$\mathbf{r}_i \in \mathbb{R}^{768}$$

for each document. For each index $i \in \{1, \ldots, 29\}$, the aggregation strategies are defined as follows:

- **Arithmetic mean**:
$$\mathbf{r}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{h}_{i,j}$$
- **Weighted mean** (with weights $w_{i,j} \geq 0$, normalized such that $\sum_{j=1}^{n_i} w_{i,j} = 1$):
$$\mathbf{r}_i = \sum_{j=1}^{n_i} w_{i,j}\, \mathbf{h}_{i,j}$$
with different weighting strategies:
  - Proportional to chunk length (in characters):
$$w_{i,j} = \frac{\ell_{i,j}}{\sum_{k=1}^{n_i} \ell_{i,k}},$$
where $\ell_{i,j}$ is the length of the j-th chunk of document d_i
  - Uniform weights:
$$w_{i,j} = \frac{1}{n_i}$$
  - Linearly increasing (higher weights for later chunks):
$$w_{i,j} = \frac{j}{\sum_{k=1}^{n_i} k}$$
  - Linearly decreasing (higher weights for earlier chunks):
$$w_{i,j} = \frac{n_i - j + 1}{\sum_{k=1}^{n_i} (n_i - k + 1)}$$
- **Max pooling** (element-wise maximum across chunks):
$$\mathbf{r}_i^{(k)} = \max_{1 \leq j \leq n_i} \mathbf{h}_{i,j}^{(k)}, \quad k = 1, \ldots, 768$$
- **Sum pooling** (element-wise summation of embeddings):
$$\mathbf{r}_i = \sum_{j=1}^{n_i} \mathbf{h}_{i,j}$$

On the other hand, the second strategy represents each document as a matrix by concatenating the chunk embeddings and applying padding if necessary.

To evaluate the semantic relevance of different document sections, three input configurations were considered for each resolution:

1. **All chunks**: All embeddings were included in the aggregation.

2. **All except the last chunk**: The final chunk corresponding to the final section was excluded, focusing on narrative and justificatory parts like preambles and legal references.

3. **Only the last chunk**: Only the embedding for the final section was used, typically containing the decision and formal content.

To visualize the spatial distribution and potential grouping of documents, Principal Component Analysis (PCA) was employed as a dimensionality reduction technique. Although Minimum-Distortion Embedding (MDE) was also tested for this purpose, PCA proved to be more suitable for our data, providing clearer and more interpretable visualizations of the document clusters.

Principal Component Analysis [32, 48] is a widely used dimensionality reduction technique that transforms a dataset with possibly correlated features into a set of linearly uncorrelated variables called principal components. These components are ordered such that the first captures the highest possible variance in the data, the second the next highest variance under the constraint of being orthogonal to the first, and so on.

Formally, given a dataset

$$\mathbf{X} \in \mathbb{R}^{n \times d},$$

where n is the number of observations and d the number of features, PCA proceeds as follows:

1. **Standardization:** The data is mean-centered by subtracting the empirical mean

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

from each observation, obtaining the centered matrix

$$\mathbf{X}_c = \mathbf{X} - \mathbf{1}_n \boldsymbol{\mu}^\top,$$

where $\mathbf{1}_n \in \mathbb{R}^n$ is a column vector of ones.

2. **Covariance matrix computation:** The empirical covariance matrix is then computed as

$$\mathbf{C} = \frac{1}{n-1} \mathbf{X}_c^\top \mathbf{X}_c \in \mathbb{R}^{d \times d}.$$

By construction, $\mathbf{C}$ is symmetric and positive semi-definite.

3. **Eigen decomposition:** Since $\mathbf{C}$ is symmetric, it admits an eigendecomposition

$$\mathbf{C} = \mathbf{W} \boldsymbol{\Lambda} \mathbf{W}^\top,$$

where $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_d]$ is an orthonormal matrix whose columns are eigenvectors satisfying

$$\mathbf{C}\mathbf{w}_j = \lambda_j \mathbf{w}_j, \quad j = 1, \ldots, d,$$

and $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_d)$ is the diagonal matrix of eigenvalues.

The eigenvalues $\lambda_j \geq 0$ quantify the variance explained by each principal component $\mathbf{w}_j$, while the eigenvectors form an orthogonal basis of $\mathbb{R}^d$. Without loss of generality, the eigenvalues are sorted in decreasing order:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0,$$

so that the first principal component corresponds to the direction of maximum variance, the second to the next maximum variance orthogonal to the first, and so on.

4. **Projection:** To reduce the dimensionality, the data is projected onto the top-k eigenvectors associated with the largest k eigenvalues:

$$\mathbf{Z} = \mathbf{X}_c \mathbf{W}_k,$$

where $\mathbf{W}_k = [\mathbf{w}_1, \ldots, \mathbf{w}_k] \in \mathbb{R}^{d \times k}$. The resulting matrix $\mathbf{Z} \in \mathbb{R}^{n \times k}$ provides a k-dimensional representation of the dataset that preserves as much variance as possible.

Principal Component Analysis was applied to the 29-document sample to reduce the dimensionality of embeddings and facilitate visual inspection. Two-dimensional and three-dimensional scatter plots were produced by projecting onto the first two and three principal components, respectively. Each point was color-coded based on its group, as defined in Table 5.1.

To quantitatively evaluate the quality of clustering, two widely adopted metrics were computed:

- **Silhouette Score** [56]: This metric evaluates the cohesion and separation of clusters by comparing, for each data point, the average distance to points within the same cluster (intra-cluster distance) against the average distance to points in the nearest neighboring cluster (inter-cluster distance). The Silhouette coefficient for each point ranges from -1 to 1, where values close to 1 denote well-separated and cohesive clusters, values around 0 indicate overlapping clusters or ambiguous assignments, and negative values reveal potential misclassification or poor clustering structure. The overall Silhouette Score is computed as the mean of all individual coefficients, providing a global measure of clustering effectiveness.

- **Davies-Bouldin Index** (DBI) [14]: This index quantifies clustering quality by calculating the average similarity between each cluster and its most similar one, where similarity is defined as the ratio of the sum of intra-cluster dispersions to the inter-cluster centroid distance. Formally, for each cluster, the maximum value of this ratio with respect to other clusters is computed, and the DBI is the average of these maxima across all clusters. Lower DBI values correspond to clusters that are compact and well-separated, indicating better clustering performance.

The combination of PCA with cluster validity indices supports visual interpretation with statistical evidence, offering a robust understanding of the dataset's structure and the effectiveness of the observed groupings.

4.3 Semantic Retrieval in the RAG Architecture

The objective of this design phase is the development and evaluation of a semantic retrieval pipeline, aimed at returning relevant documents in response to natural language queries. This pipeline represents the core component of a system based on the Retrieval-Augmented Generation architecture. In fact, this architecture combines a semantic retrieval module with a generative language model. The query is used to retrieve relevant documents via vector search, which are subsequently used as context for generating a more accurate and informed response.

The analysis was initially conducted using an open-source vector store, that is FAISS [18], and subsequently replicated using the Vertex AI Vector Search service provided by Google Cloud Platform [23].

The experiments were carried out on the same set of 29 resolutions previously analyzed, where each document was segmented into chunks and converted into vector representations through embedding techniques.

4.3.1 Types of Queries Considered

The queries used in the analysis were divided into three categories to evaluate the robustness and generalization capabilities of the system:

1. **Queries derived from real chunk excerpts**: These consist of text segments directly extracted from the documents, used to verify the system's ability to retrieve the same or semantically similar documents.

2. **Generalized queries**: Starting from the original excerpts, paraphrased and more abstract versions were formulated in order to evaluate the system's sensitivity to broader and less textually anchored queries.

3. **User-oriented queries**: Finally, realistic queries were tested, formulated in interrogative or imperative style, simulating the behavior of a final user.

4.3.2 Evaluation Metrics

To assess the effectiveness of the retrieval system, we employed metrics based on cosine similarity between embedding vectors, as well as metrics such as Context Precision and Context Recall. We describe each in detail below.

Cosine similarity is a measure of similarity between two vectors that evaluates the cosine of the angle between them in vector space. Given two vectors $\mathbf{A}$ and $\mathbf{B}$, it is defined as:

$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

where $\mathbf{A} \cdot \mathbf{B}$ denotes the dot product and $\|\mathbf{A}\|$, $\|\mathbf{B}\|$ are the Euclidean norms of the vectors. The resulting value ranges from -1 to 1, where 1 indicates perfect alignment and thus maximum similarity, 0 corresponds to orthogonality (no correlation), and -1 denotes diametrically opposed vectors and thus maximum dissimilarity in direction.

Cosine similarity was implemented to compute the following metrics:

1. **Query-Document similarity**: This measures the semantic coherence between the query and each retrieved document by computing the cosine similarity between the query vector and each of the vectors associated with the retrieved documents. The computed statistics include:

   - The average similarity between the query and all top-k retrieved documents;
   - The maximum similarity between the query and the most relevant document;
   - The minimum similarity among the *top-k* results.

2. **Intra-Document similarity**: This evaluates the internal semantic consistency among the retrieved documents, by computing the cosine similarity between each pair of document vectors retrieved. The computed statistics include:

   - The average pairwise similarity among the retrieved documents;
   - The most semantically similar document pair;
   - The least semantically similar document pair.

On the other hand, the metrics Context Precision and Context Recall, introduced by the Retrieval-Augmented Generation Assessment (RAGAS) framework [60], allow for a quantitative evaluation of the relevance of retrieved documents to a given query in RAG-based systems.

Context Precision measures the proportion of retrieved content that is actually relevant to the query, while Context Recall assesses the extent to which the system covers all the relevant content necessary to answer the query. They are formally defined as:

$$\text{Context Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Context Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

where:

- **True positives**: Text segments in the retrieved documents that contain correct and relevant information with respect to the expected answer.
- **False positives**: Text segments in the retrieved documents that do not contribute relevant information to the answer.
- **False negatives**: Relevant content that would have been necessary for a complete answer but was not retrieved.

These metrics rely on a semantic comparison between the retrieved content and a manually defined *ground truth*, which represents the ideal information that the system should provide for each query. Initially, these metrics were employed in our system to evaluate the performance of the retrieval phase. However, the practical difficulty of constructing and validating a reliable and representative ground truth for each query rendered the systematic use of these metrics unsustainable. For this reason, their use was discontinued in later stages of the analysis in favor of unsupervised alternatives, such as cosine similarity.

4.3.3 Experimental Methodology and Comparative Analysis

We considered the sample of the 29 resolutions previously analyzed and applied two vector search frameworks: FAISS [18] and Vertex AI Vector Search [23]. The objective was to assess the semantic alignment between user-defined queries and the retrieved document chunks using similarity-based metrics.

Using FAISS in combination with LangChain [36], the system retrieved the top-4 most semantically similar document chunks for each query. The evaluation, based on cosine similarity, demonstrated strong semantic coherence, with average query-document similarities exceeding 0.85 and low variability among the top-ranked results. This suggests that FAISS is capable of effectively capturing the latent meaning of user queries, even when they are abstract or expressed in natural language. Intra-document similarity analysis further confirmed the homogeneity of the retrieved content, indicating a robust and reliable retrieval mechanism.

The same experiment was replicated using Vertex AI Vector Search and the retrieved documents remained semantically coherent and contextually appropriate. In particular, Vertex AI exhibited greater consistency in the selection of documents across different queries, reflecting its effectiveness in large-scale vector search scenarios.

Overall, both systems demonstrated the ability to retrieve semantically relevant results and validated the use of similarity metrics as reliable tools for evaluating the semantic relevance and internal coherence of retrieval-based systems.

4.4 LLM Configuration

Following the development and validation of the semantic retrieval component, a new phase of the project was initiated, focused on the integration of a Large Language Model within the

system. The primary objective of this phase was to enable the system to respond to complex user requests, with a strong emphasis on the quality, consistency, and accuracy of the provided information.

4.4.1 Prompt Engineering

The first step involved the design and iterative optimization of the prompt supplied to the model. This process required extensive experimentation, qualitative analysis, and comparison of different prompt versions, with the aim of steering the model toward formally correct, consistent, and domain-specific responses. The prompt was progressively refined to enhance its ability to guide the model in generating relevant content, while significantly reducing hallucinations or arbitrary inferences.

A crucial aspect of this stage was the explicit formulation of instructions within the prompt itself. The desired tone was defined as technical-administrative, suitable for a professional audience, and the content of the responses was constrained to rely exclusively on the combination of the following three elements:

1. **The user's query**;
2. **The context retrieved by the retrieval system**;
3. **The conversation history**.

The prompt includes an explicit prohibition on generating proper names, organizations, personal data, or any other information not present in the provided context. In the case of missing or confidential data, the model was instructed to use appropriate placeholders, avoiding autonomous interpretations or completions. Each response concludes with an interactional sentence that encourages the user to ask additional questions, thereby fostering an ongoing dialogue.

4.4.2 Operational Integration of the LLM

The concrete integration of the Large Language Model into the system followed an incremental approach, allowing for the progressive refinement of configurations and the validation of system performance across increasingly realistic scenarios.

Initially, the infrastructure was tested on a reduced subset consisting of the 29 resolutions used in previous analyses, in order to limit computational costs and facilitate manual evaluation of the generated responses. In this preliminary phase, documents were already preprocessed and segmented into informative chunks. The input to the model included the optimized prompt, a selection of relevant documents retrieved via vector search, and few-shot examples where applicable. Subsequently, the sample was extended to cover 20% of the dataset comprising resolutions from the years 2023 and 2024, and finally up to 80% of the same corpus. This was done following a stratified sampling strategy proportional to the thematic groups present, thus ensuring data representativeness and enabling the evaluation of system robustness under more realistic conditions.

In the final integration phase, the MultiVector Retriever mechanism [35] was adopted, with the goal of further improving the quality and relevance of the generated responses. Unlike standard retrieval methods that operate at the chunk level, the MultiVector Retriever was employed in our system to retrieve entire administrative resolutions. Each document was encoded into a set of semantic vectors, one per chunk, and stored collectively as a multivector representation of the full resolution. During retrieval, the relevance of a document was determined by comparing the query to all its constituent vectors and selecting the highest similarity score. This approach proved particularly effective in two main scenarios:

1. Answering complex queries requiring a cross-sectional view of multiple documents;

2. Generating new administrative resolutions based on multiple sources.

This strategy enabled the system to exploit the full structural and semantic richness of administrative documents, ensuring that the model had access to the broader legislative and procedural context when generating responses. However, this also introduced additional computational overhead, both in terms of retrieval time, due to the need to compare against multiple vectors per document, and prompt length, since full resolutions had to be retained in memory and passed to the language model.

Two models were tested for response generation: Gemini version 1.5 Flash and Gemini version 2.0 Flash Lite. Ultimately, the Gemini version 2.0 Flash model [21] was selected due to its optimal balance between quality and performance, and its capability to handle long contexts within low inference times. Additionally, a history window mechanism was configured to store the last six conversational turns, thus preserving dialogue coherence and supporting multi-turn interactions without introducing ambiguity or loss of context.

Overall, the final system architecture integrates the following elements in a synergistic manner:

- Preprocessed documents indexed using vector search techniques;
- A specialized and optimized prompt for the domain of administrative resolutions;
- A retrieval component supporting multiple semantic views (MultiVector Retriever);
- An advanced LLM capable of managing large contexts and multi-turn dialogues (Gemini version 2.0 Flash).

This configuration enabled the generation of responses that were not only factually correct but also consistent with the form and style of official documents, thereby laying the groundwork for an intelligent system capable of assisting in the drafting and validation of complex administrative acts.

4.4.3 Evaluation and Improvement of Generated Responses

During the evaluation phase of the system-generated responses, the queries were divided into two broad categories, each with distinct objectives and characteristics.

1. **Search queries**, focused on extracting and retrieving precise and specific information within the document corpus. These queries aim to test the model's ability to provide accurate, coherent, and contextually grounded answers based solely on the retrieved information.

2. **Generation queries**, intended to assess the model's capacity to produce new structured content, such as the automatic drafting of administrative resolutions. These queries require the model to synthesize and reorganize the contextual information while adhering to the form, style, and specific rules of the technical-administrative domain.

This distinction allowed for targeted calibration of prompt engineering and testing strategies, facilitating a more in-depth analysis of the system's performance across diverse scenarios. Throughout our analysis, greater emphasis was placed on the second category of queries, as the primary objective of the constructed intelligent agent is precisely the generation of deliberative acts.

4.4.4 Request Scheduling Functionality

As part of the system's evolution, a scheduling functionality was introduced to allow users to plan the generation of one or more resolutions in advance. This feature is particularly useful in scenarios where the drafting of documents must be distributed over time or aligned with internal deadlines.

Users are required to input:

- A completion date, indicating the day by which the document(s) should be available;
- One or more requests, each containing the necessary information for the generation of a resolution (e.g., subject, references, objectives, legal constraints).

Once the user confirms the planning, the system takes care of scheduling the processing of each batch automatically. Multiple requests can be scheduled for the same day or across different days. On the specified date, the generated documents will appear in the dedicated area, from which they can be downloaded in Word format (.docx) for further editing or deleted if no longer needed.

The scheduling mechanism is based on two core components:

- **APScheduler** [26] is used to register and manage scheduled jobs dynamically. For each batch, the system calculates the earliest available time slot starting from 08:00 AM on the completion date, with a fixed slot duration of 5 minutes. This ensures that batches are evenly distributed over time and no two are executed simultaneously.
- **queue.Queue** [20] is employed as a thread-safe buffer for request processing. At the scheduled time, each batch is inserted into the queue and subsequently handled by a pool of worker threads that process the requests asynchronously and in FIFO order.

From a methodological perspective, the chosen implementation was guided by the need for a simple yet effective scheduling mechanism that would integrate smoothly with the rest of the system. By leveraging `APScheduler` and a queue-based approach, we were able to introduce asynchronous processing without overcomplicating the architecture. A key assumption underlying this design is that the application remains active during the scheduling window. This constraint is acceptable in our context, as users are not expected to plan batch execution during periods in which the application is offline (such as weekends or holidays). Consequently, the scheduling mechanism is both aligned with real user behavior and operationally sound. To ensure persistence and traceability, all relevant information, including the request content, assigned slots, and scheduling metadata, is stored in a centralized Firestore database. This guarantees durability across restarts and enables consistent tracking of planned operations. Finally, the actual execution of scheduled batches is delegated to a pool of parallel threads, allowing background processing without compromising the responsiveness of the chat interface. This design ensures that users can continue to interact with the agent in real time, even when scheduled jobs are being processed in the background.

This architecture separates scheduling from execution, ensuring system stability, scalability, and timing precision. It also enables the platform to handle a high volume of requests across multiple days without manual intervention or scheduling conflicts.

Chapter 5

Results

This chapter presents the outcomes of the experimental evaluation, highlighting the effectiveness of the semantic retrieval pipeline, the accuracy and robustness of the integrated LLM, and the overall system performance across different testing scenarios.

5.1 Preliminary Document Analysis

In the initial phase of the project, we focused on documents produced in the year 2024, excluding both corrections and corrected documents. First we made an exploratory overview of them in order to analyze their structure and identify recurrent patterns. To facilitate the subsequent steps of analysis and modeling, these resolutions were manually grouped into 21 thematic categories based on content similarity. Figure 5.1 illustrates the distribution of the documents across the 21 generated clusters, while Table 5.1 provides a description of each group. This categorization enabled more effective data management and allowed for a more targeted evaluation of the specific characteristics of each document group.

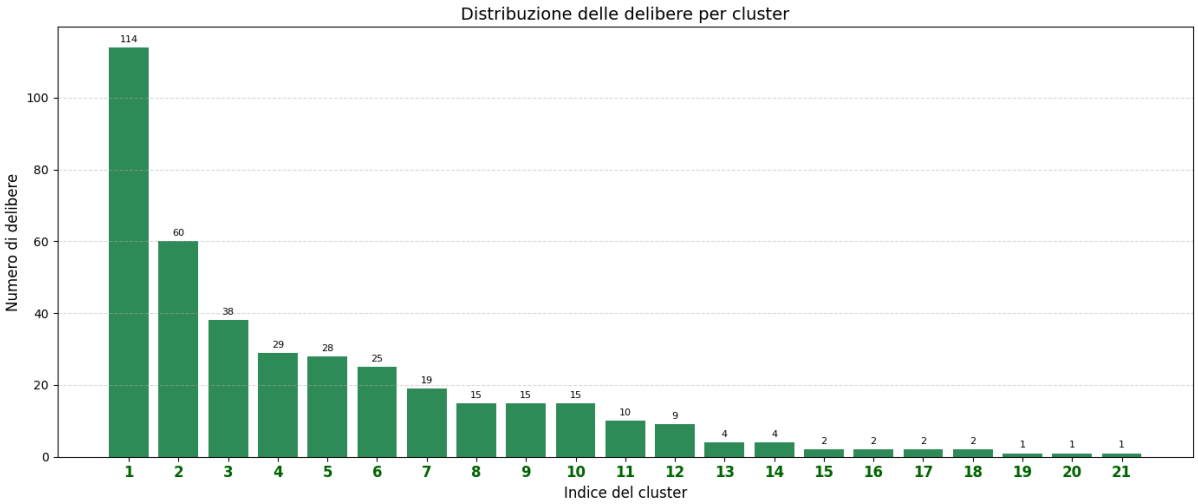


Figure 5.1: Histogram showing the distribution of resolutions across clusters. Each bar indicates the number of resolutions associated with a specific group. The 21 groups are indexed along the x-axis, based on the legend provided in Figure 5.1.

Considering the distribution of documents across clusters, it is observed that the first three thematic groups comprise 114, 60, and 38 resolutions respectively, totaling 212 out of 395 documents. This concentration represents over 53% of the entire dataset. Consequently, particular attention will be given to these clusters during the experimentation and validation phases to

Cluster	Cluster Description
1	Procurement of hospital supplies
2	Contract renewal/extension
3	Hiring and authorization of budget expenditures
4	Contract revision/expansion
5	Outsourcing of services to external companies
6	Acknowledgment of agreements
7	Rectification
8	Supply/farmaceutical procurement
9	Request for quotation/tender announcement
10	Appointment
11	Exercise of option right
12	Donation acceptance
13	Reallocation of expenditures
14	Contract initiation resolution
15	Subcontracting authorization
16	Correspondence
17	Program approval
18	Status reconciliation
19	No contract awarded
20	Contributions
21	Insurance policy stipulation

Table 5.1: Legend of the 21 thematic clusters, from the content analysis of the resolution objects.

ensure high performance on a substantial portion of the corpus, which most frequently reflects real-world use cases.

5.2 Semantic Processing of Documents

Following the pre-processing phase and the application of a content-aware splitting strategy for dividing the text into chunks, as described in the previous chapter, we proceeded with semantic encoding. Specifically, we generated a vector embedding for each chunk of all documents using the `text-embedding-005` model provided by Google Cloud's Vertex AI platform [22].

5.2.1 Embedding Distribution Analysis

Once the embeddings for each chunk were obtained, we conducted an analysis to examine the semantic distribution of the resolutions in the vector space. The objective was to assess the internal coherence of the manually defined clusters introduced in Table 5.1, and to identify potential similarities or anomalies among resolutions belonging to different thematic groups. Principal Component Analysis was employed as a dimensionality reduction technique in order to visualize the spacial distribution and potential groupings of documents.

For this study, a sample of 29 resolutions issued in 2024 was selected, ensuring an even representation across the existing clusters. The previous chapter detailed three input configurations evaluated for each resolution: using all data chunks, all chunks except the final one, and only the final chunk. In the following, we will therefore present the results of these three types of

analyses; this comparison will allow us to better understand the contribution of different parts of the documents to cluster coherence and to detect patterns or outliers that may be specific to certain sections.

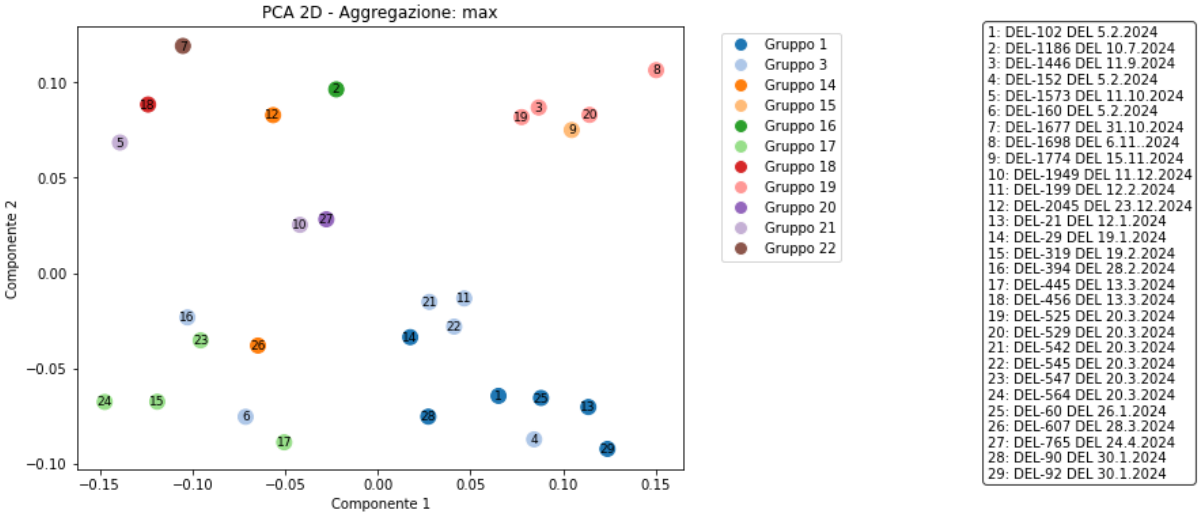


Figure 5.2: Two-dimensional projection obtained by applying PCA to the embeddings aggregated using max pooling, considering all text chunks for each of the 29 resolutions. The Silhouette Score is 0.0563, and the Davies-Bouldin Index is 1.3535.

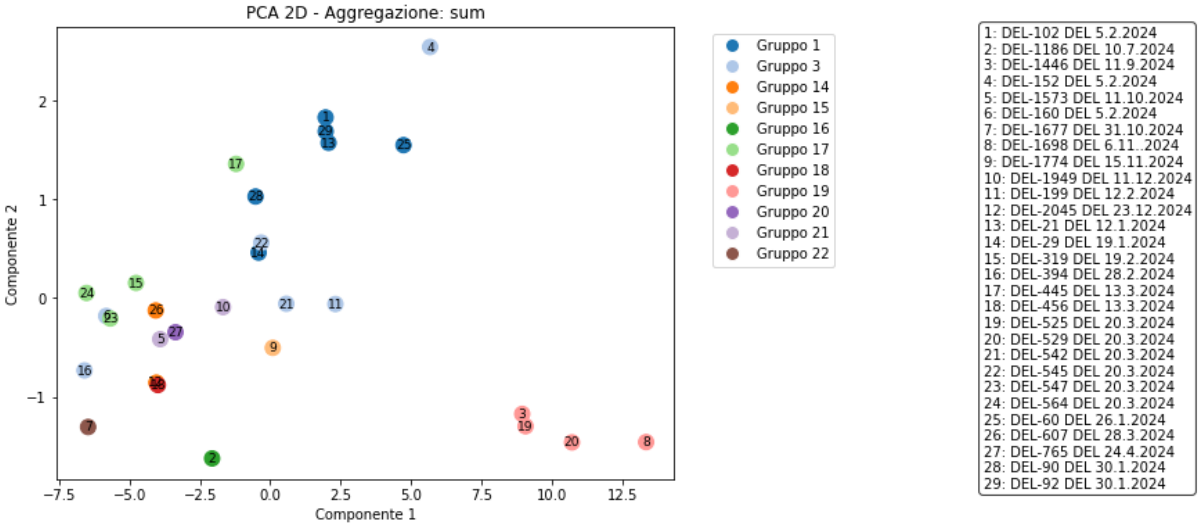


Figure 5.3: Two-dimensional projection obtained by applying PCA to the embeddings aggregated using sum pooling, considering all text chunks for each of the 29 resolutions. The Silhouette Score is 0.0111, and the Davies-Bouldin Index is 1.3887.

In the first stage of the analysis, all chunks from each of the 29 selected resolutions were considered. Their corresponding embeddings were aggregated using the following techniques, previously explained:

- Max pooling (Figure 5.2);
- Sum pooling (Figure 5.3);
- Weighted arithmetic mean, using constant, length-proportional, increasing, and decreasing weights (Figures 5.4, 5.5, 5.6, and 5.7 respectively).

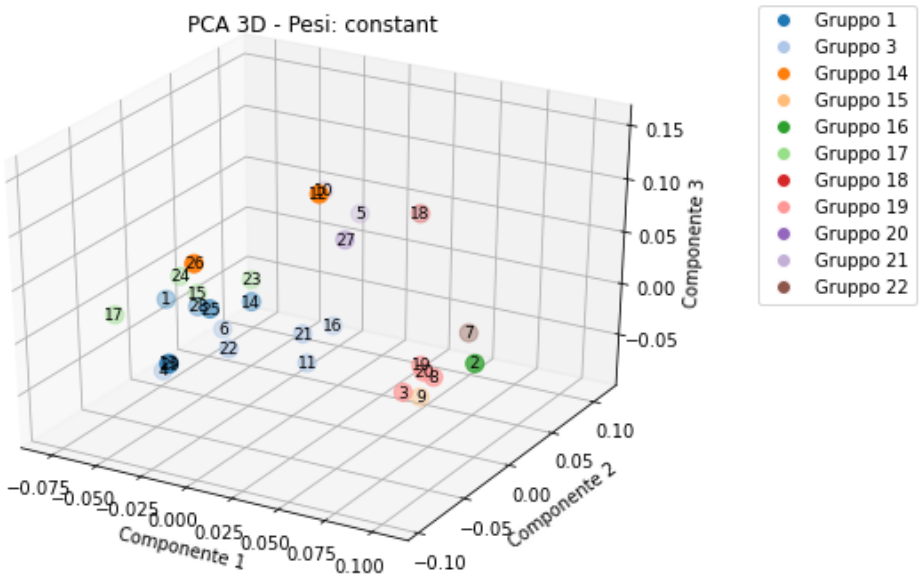


Figure 5.4: Three-dimensional projection obtained by applying PCA to the embeddings aggregated using a weighted arithmetic mean with constant weights equal to 1, considering all text chunks for each of the 29 resolutions. The Silhouette Score is 0.0627, and the Davies-Bouldin Index is 1.2325.

Subsequently, Principal Component Analysis was applied to the aggregated vectors, allowing us to visualize the 29 resolutions as points in 2D or 3D space. Each point is color-coded based on the cluster it belongs to (as defined in Table 5.1) and labeled with the unique identifier of the resolution.

To quantitatively assess the consistency of the manual clustering, we computed the Silhouette Score and the Davies-Bouldin Index.

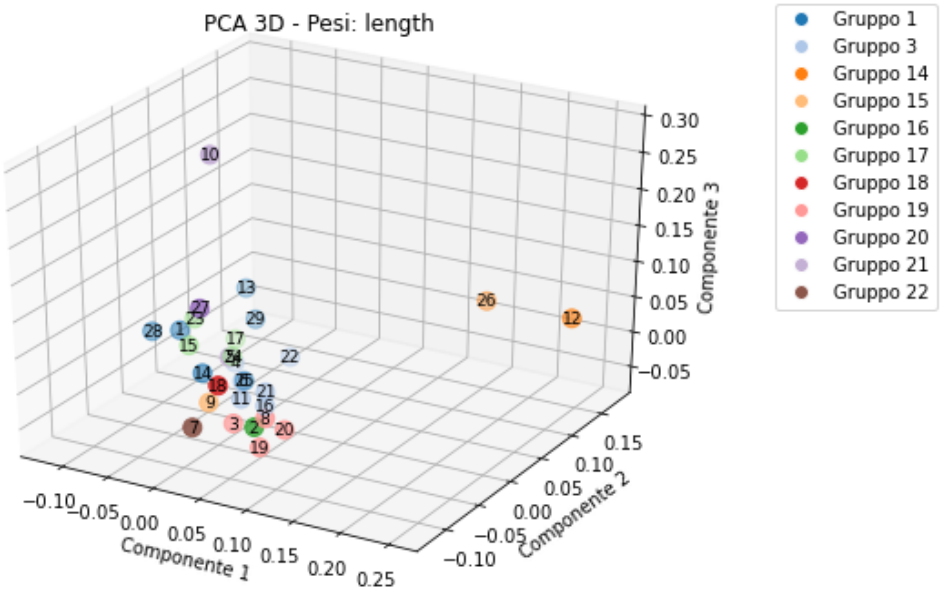


Figure 5.5: Three-dimensional projection obtained by applying PCA to the embeddings aggregated using a weighted arithmetic mean with weights proportional to the chunk length, considering all text chunks for each of the 29 resolutions. The Silhouette Score is 0.0379, and the Davies-Bouldin Index is 1.3368.

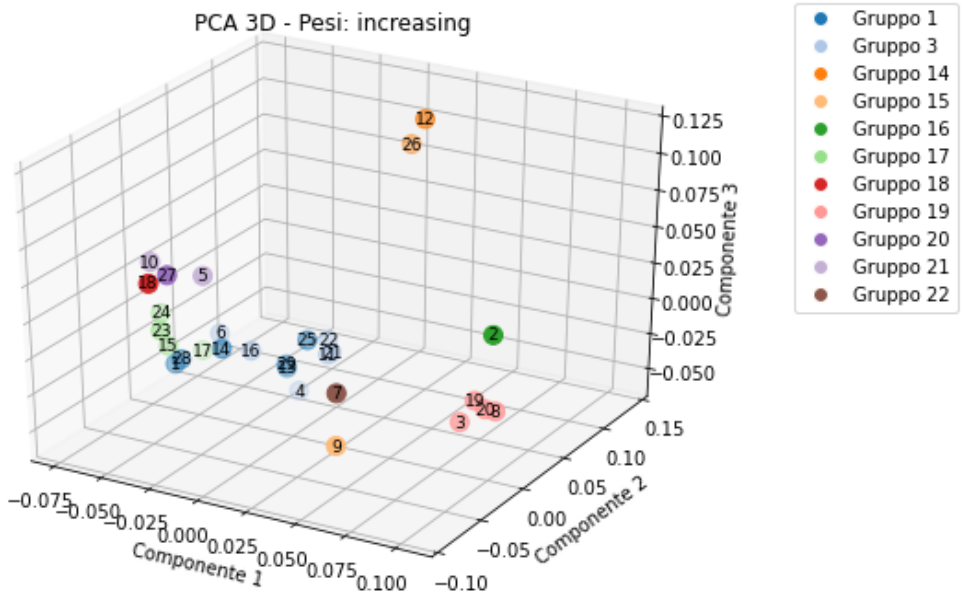


Figure 5.6: Three-dimensional projection obtained by applying PCA to the embeddings aggregated using a weighted arithmetic mean with increasing weights based on the chunk's position in the document, considering all text chunks for each of the 29 resolutions. The Silhouette Score is 0.0985, and the Davies-Bouldin Index is 1.1875.

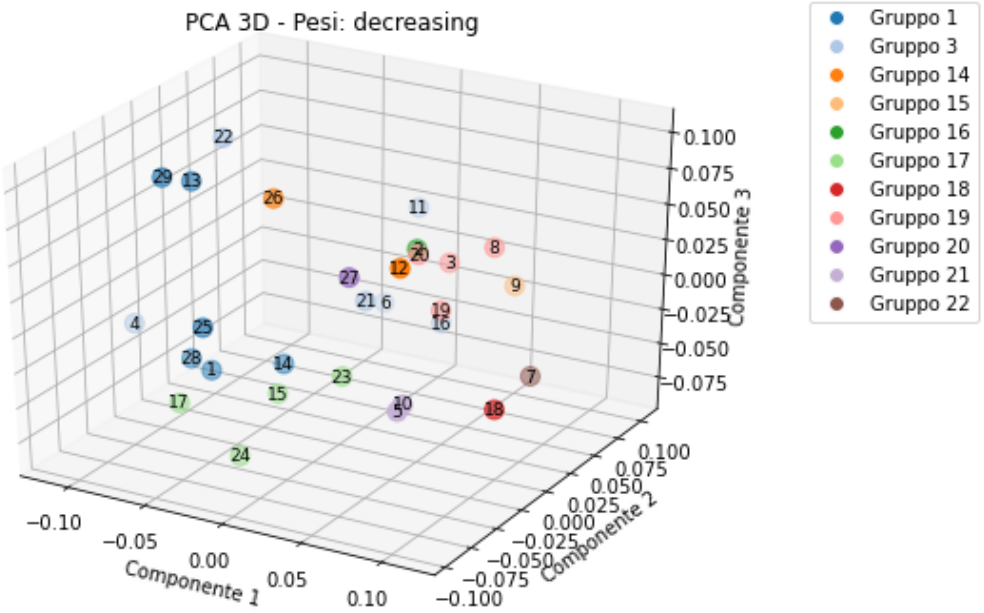


Figure 5.7: Three-dimensional projection obtained by applying PCA to the embeddings aggregated using a weighted arithmetic mean with decreasing weights equal to 1/(chunk position), considering all text chunks for each of the 29 resolutions. The Silhouette Score is 0.0010, and the Davies-Bouldin Index is 1.4630.

In the second phase of the analysis, we considered all chunks of each of the 29 selected resolutions, excluding the last one. This final chunk typically corresponds to the section in which the decision is formally stated, summarizing the content discussed in the preceding parts of the document. The corresponding embedding vectors were aggregated using the following

techniques:

- Max pooling (Figure 5.8);
- Sum pooling (Figure 5.9);
- Weighted arithmetic mean, experimenting with constant weights, weights proportional to chunk length, increasing and decreasing weights (Figures 5.10, 5.11, 5.12, and 5.13).

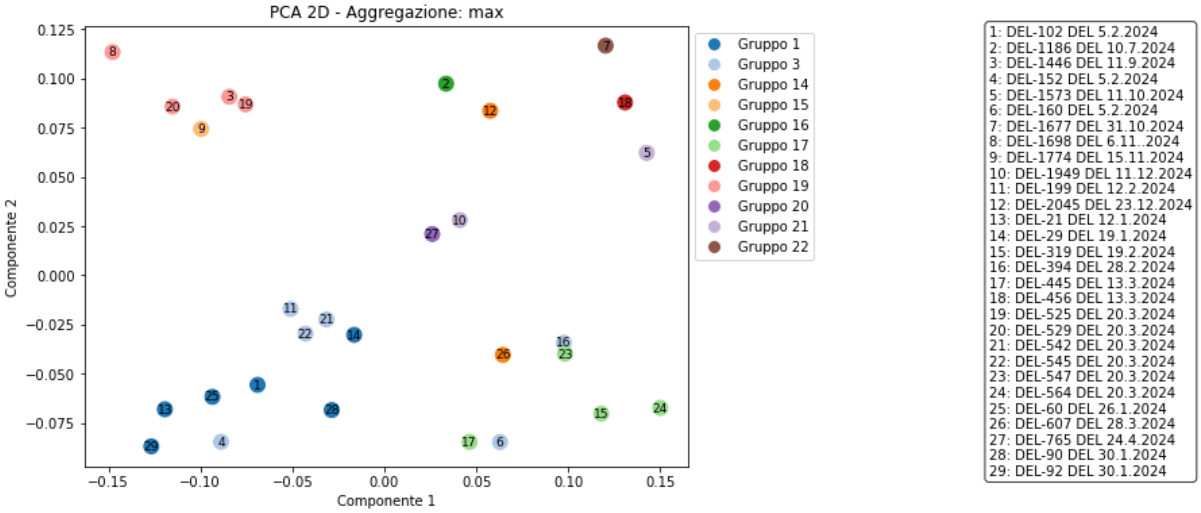


Figure 5.8: Two-dimensional projection obtained by applying PCA to the embeddings aggregated using max pooling, considering all text chunks except the final one for each of the 29 resolutions. The Silhouette Score is 0.0568, while the Davies-Bouldin Index is 1.3452.

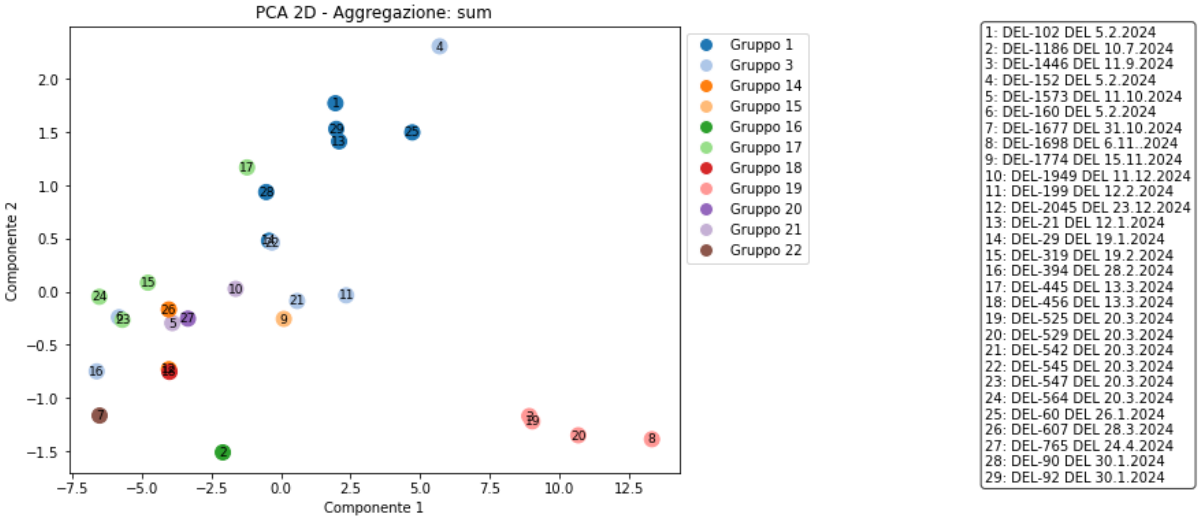


Figure 5.9: Two-dimensional projection obtained by applying PCA to the embeddings aggregated using sum pooling, considering all text chunks except the final one for each of the 29 resolutions. The Silhouette Score is -0.0042, while the Davies-Bouldin Index is 1.4042.

Subsequently, PCA was applied to the aggregated vectors, and the 29 resolutions were visualized as points in either 2D or 3D space. As in the previous phase, each point is assigned a color corresponding to the cluster it belongs to (based on Table 5.1) and a number representing the unique identifier of the resolution.

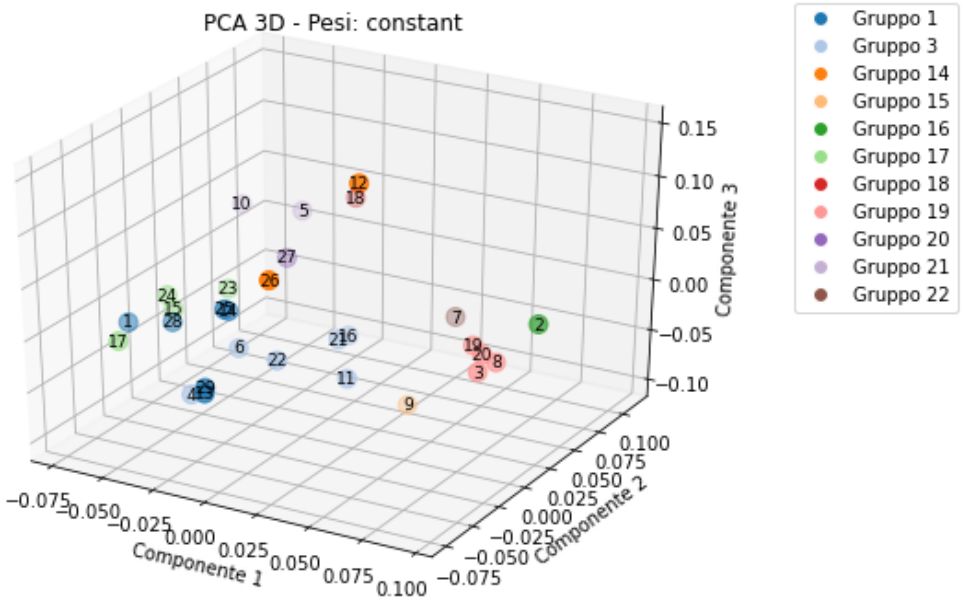


Figure 5.10: Three-dimensional projection obtained by applying PCA to the embeddings aggregated using a weighted arithmetic mean with constant weights equal to 1, considering all text chunks except the final one for each of the 29 resolutions. The Silhouette Score is 0.0620, while the Davies-Bouldin Index is 1.2213.

We also computed the Silhouette Score and the Davies-Bouldin Index to quantify the consistency of our manual clustering.

The rationale behind excluding the final section from the analysis lies in its nature: it does not introduce new information but rather summarizes what has already been stated, potentially introducing redundancy into the semantic representation of the document.

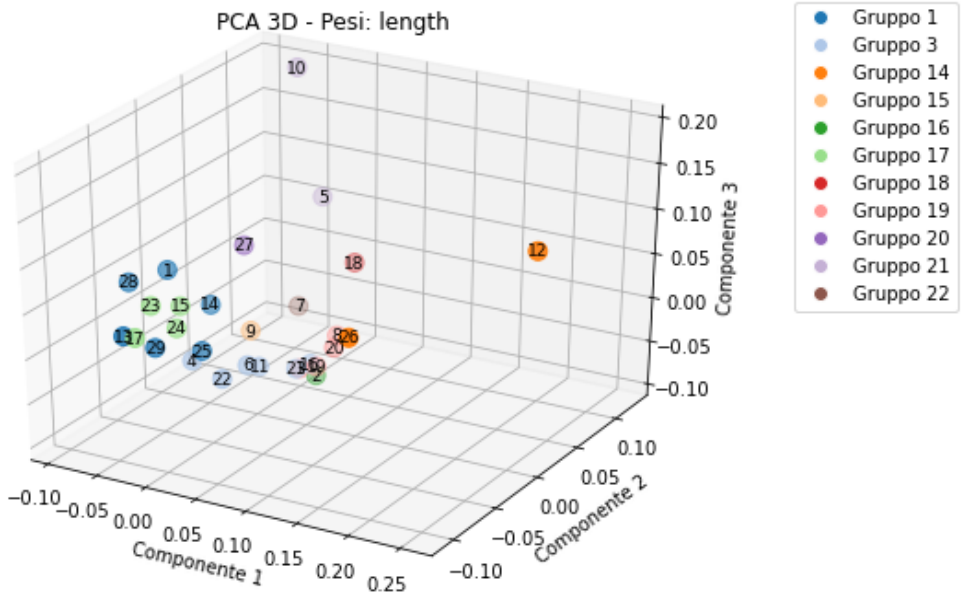


Figure 5.11: Three-dimensional projection obtained by applying PCA to the embeddings aggregated using a weighted arithmetic mean with weights proportional to the corresponding chunk length, considering all text chunks except the final one for each of the 29 resolutions. The Silhouette Score is 0.0408, while the Davies-Bouldin Index is 1.2753.

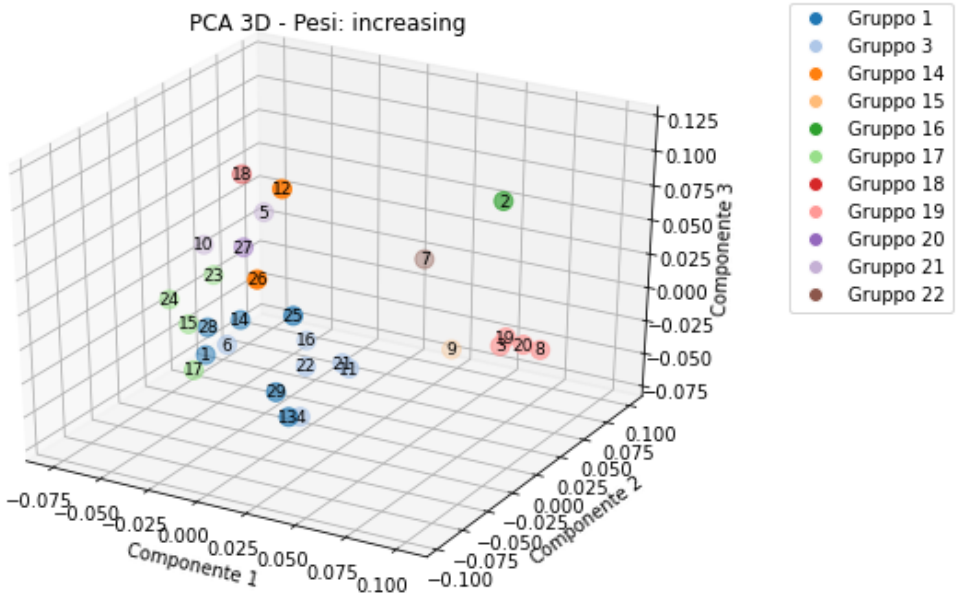


Figure 5.12: Three-dimensional projection obtained by applying PCA to the embeddings aggregated using a weighted arithmetic mean with increasing weights based on the chunk's position within the document, considering all text chunks except the final one for each of the 29 resolutions. The Silhouette Score is 0.1017, while the Davies-Bouldin Index is 1.1553.

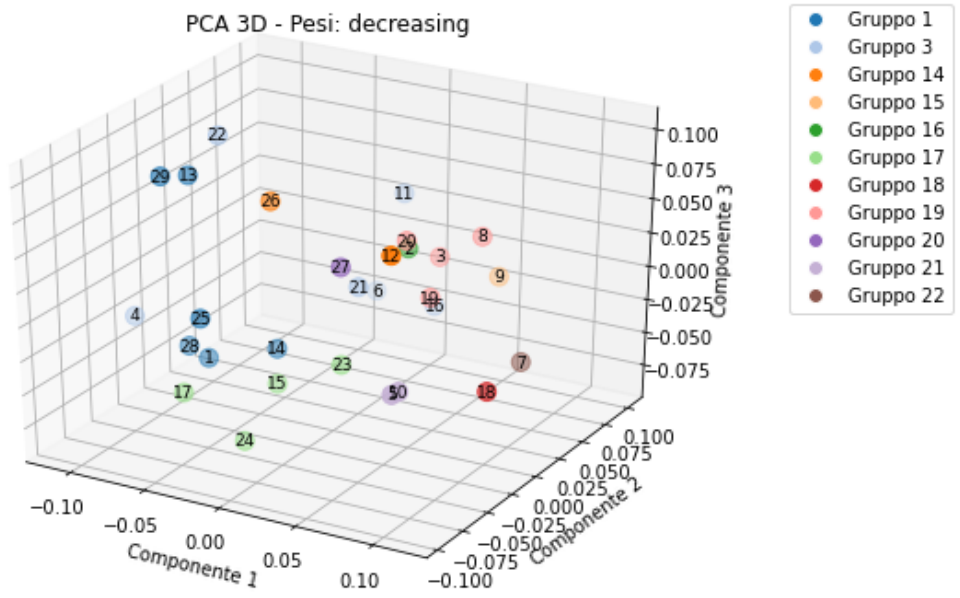


Figure 5.13: Three-dimensional projection obtained by applying PCA to the embeddings aggregated using a weighted arithmetic mean with decreasing weights equal to 1/(chunk position in the document), considering all text chunks except the final one for each of the 29 resolutions. The Silhouette Score is 0.0009, while the Davies-Bouldin Index is 1.4646.

In the final phase of the analysis, we considered only the last chunk of each of the 29 selected resolutions, corresponding to the final section in which the contents previously expressed are formally enacted. Principal Component Analysis was applied to the corresponding embedding vectors. The choice to focus solely on the final section stems from its role as a concise summary of the key information already introduced in the earlier parts of the document. This approach

helps reduce redundancy and simplifies subsequent processing. Figure 5.14 displays the results obtained using this strategy, showing a 2D projection of the 29 resolutions. As in the previous visualizations, each point is associated with a specific color, that indicates the cluster assignment according to Table 5.1, and a unique identifier.

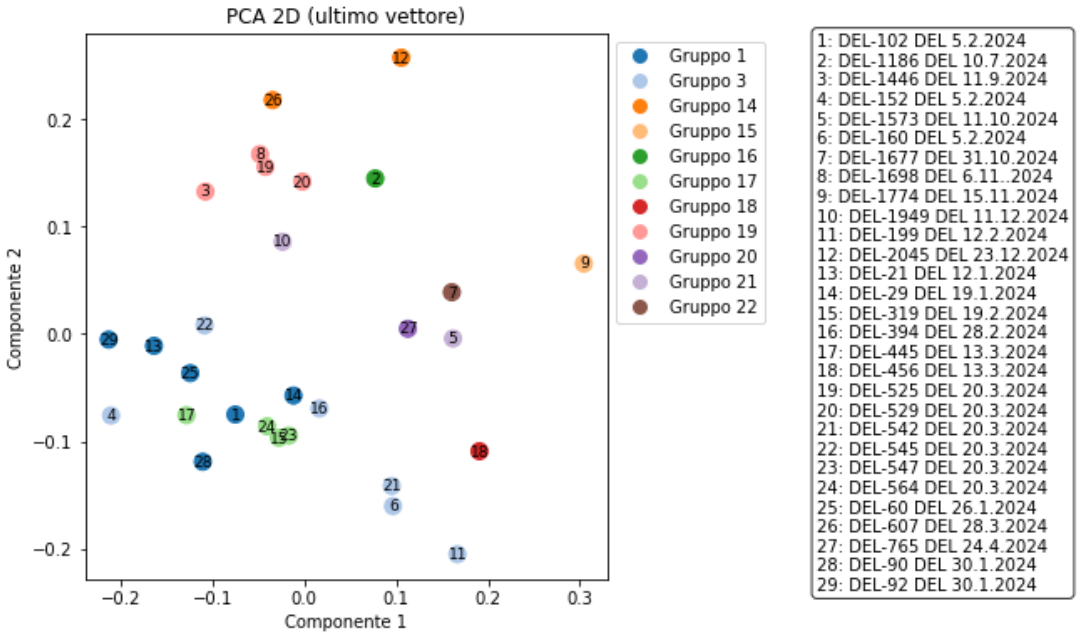


Figure 5.14: Two-dimensional projection obtained by applying PCA to the embeddings corresponding solely to the final chunk of each of the 29 resolutions. The Silhouette Score is 0.0413, while the Davies-Bouldin Index is 1.4050.

In conclusion, the most satisfactory results were obtained with the following configuration:

- **Input:** All chunks except for the final one, corresponding to the final section, in order to capture the descriptive and normative content contained in the premises.
- **Aggregation technique:** Weighted arithmetic mean of the embedding vectors, using increasing weights based on the position of the chunk within the document. This strategy proved effective in emphasizing the central sections, which typically contain the most relevant legal and regulatory information.

This configuration, shown in Figure 5.12, highlighted a well-defined visual separation between distinct groups, confirming both the validity of the manual cluster assignment and the semantic relevance of the resolutions, even in the absence of formally structured sections such as the final one.

5.3 Semantic Retrieval in the RAG Architecture

This phase focused on the design and evaluation of a semantic retrieval pipeline within the framework of a Retrieval-Augmented Generation architecture. In this paradigm, natural language queries are first converted into vector representations and used to retrieve semantically relevant documents, which are subsequently provided as context to a generative language model to enhance the quality of the generated responses.

The retrieval system was initially implemented using the open-source FAISS library [18] and later replicated on Google Cloud's Vertex AI Vector Search [24] to evaluate its scalability and

production readiness. Experiments were conducted on a corpus of 29 administrative resolutions, which were segmented into textual chunks and embedded into vector space.

To assess system performance, three query types were considered, as explained in the previous chapter. Evaluation relied on cosine similarity-based metrics, including query-document and intra-document similarity, as well as supervised measures such as Context Precision and Context Recall from the RAGAS framework [60]. While these latter metrics offered fine-grained insight into content relevance, their dependency on manually curated ground truths limited their scalability, leading to a preference for unsupervised alternatives in subsequent phases.

5.3.1 Experiments with FAISS

The entire procedure was initially implemented using FAISS, integrated with LangChain [36]. The documents were converted into embedding vectors, and for each of the three query sets, the top-$k = 4$ most semantically similar chunks were retrieved. The model's behavior was evaluated using the similarity-based metrics described earlier.

We present some of the results obtained by testing several user-relevant queries, listed in Table 5.2. For each of them, we evaluated the metrics based on cosine similarity previously introduced. Table 5.3 reports the similarity values between each query and the retrieved documents, with the relative statistics, while Table 5.4 shows the intra-document similarity values and the corresponding statistics.

Query Index	Query Content
1	Which resolutions include the acquisition of financial coverage?
2	Provide an example of a resolution involving managers with the role of director.
3	What is the total cost of the supply awarded with resolution number 29 of 19/01/2024?
4	What are the references to tenders or public calls in resolution number 1186 of 10/07/2024?

Table 5.2: User-defined test queries used to evaluate the retrieval capabilities of the system.

Query Index	Average Q-D Similarity	Maximum Q-D Similarity	Minimum Q-D Similarity
1	0.8752	0.8804	0.8658
2	0.8089	0.8134	0.8074
3	0.8810	0.8840	0.8735
4	0.8215	0.8274	0.8126

Table 5.3: For each query, the top 4 most similar documents were retrieved. The table reports the average, maximum, and minimum query-document (Q-D) cosine similarity values for each of the 4 queries listed in Table 5.2.

In addition to the aggregated metrics shown in Table 5.3, descriptive statistics were computed to provide a more detailed assessment of the system's retrieval effectiveness. Specifically, we analyzed the similarity between each query in Table 5.2 and its single most relevant retrieved document. This analysis offers insight into the peak semantic alignment achievable for each user query. The computed statistics are as follows.

- The mean cosine similarity across all queries is 0.8513, indicating generally high semantic correspondence.

Query Index	Average D-D Similarity	Maximum D-D Similarity	Minimum D-D Similarity
1	0.9491	0.9699	0.9217
2	0.9486	1.0000	0.8971
3	0.9082	0.9690	0.8689
4	0.8425	0.8731	0.8048

Table 5.4: For each query, the top 4 most similar documents were retrieved. The table reports the average, maximum, and minimum document-document (D-D) similarity values among the retrieved documents for each of the 4 queries in Table 5.2.

- The standard deviation is 0.0313, reflecting relatively low variability in similarity scores.
- The maximum similarity observed is 0.8840, while the minimum is 0.8074.

These values confirm that, even in the least favorable case, the system maintains a strong semantic match between query and top-ranked document.

The similarity metrics calculated between the queries and the retrieved documents reveal several noteworthy observations regarding the quality of the semantic retrieval process:

1. **High semantic coherence between queries and retrieved documents:** The average values of cosine similarity consistently exceeded 0.85, indicating a strong semantic alignment, even when the queries were expressed in natural language or contained a high degree of abstraction. This suggests that the system is capable of effectively capturing the latent meaning of user requests, going beyond mere lexical matching.
2. **Low dispersion among the top retrieved documents:** The intra-document similarity analysis revealed a high degree of internal homogeneity, with retrieved documents exhibiting very similar content. This internal coherence is a positive indicator of the system's reliability, as it suggests that the documents identified as most relevant share a common semantic core with respect to the original query.

5.3.2 Experiments with Vertex AI Vector Search

The tests previously conducted using FAISS were replicated on Vertex AI Vector Search, maintaining the same set of documents (specifically, we considered the 29 resolutions from the year 2024) and the same queries, grouped into the three categories described earlier.

In the preliminary phase, the following parameters were considered for querying the vector store:

- Approximate number of neighbors: 10
- Search algorithm: Tree-AH algorithm
- Distance metric: Dot Product Distance
- Normalization: None or Unit L2 Normalization
- Metadata filtering: Disabled

In addition to these parameters, performance was also evaluated based on the embedding model, the embedding dimensionality, and the chunking strategy employed. These aspects will be further investigated in subsequent optimization phases.

The `similarity_search_with_score` function from LangChain was adopted, which returns an internal score based on the retrieval service's chosen similarity metric. However, it was found

that the scores returned by Vertex AI undergo a linear transformation and cannot be interpreted as absolute distances, but rather as relative probabilities. For this reason, it is not possible to directly compare the numerical values with those obtained using FAISS. Consequently, the comparison focused on the value ranges, and the semantic quality of the retrieved documents.

Table 5.5 shows the results obtained by testing the user-interest queries presented in Table 5.2, using cosine similarity to compute the average, maximum, and minimum similarity between each query and the retrieved documents.

Query Index	Average Q-D Similarity	Maximum Q-D Similarity	Minimum Q-D Similarity
1	0.7412	0.7456	0.7357
2	0.6846	0.6954	0.6796
3	0.7634	0.7726	0.7542
4	0.6963	0.7069	0.6889

Table 5.5: For each query listed in Table 5.2, the top 4 most similar documents were retrieved. The table reports the average, maximum, and minimum query-document (Q-D) cosine similarity values for each of the 4 queries.

5.3.3 Evaluation of Retrieval Performance

The results we obtained indicate that the system is capable of returning semantically relevant documents for all categories of queries considered. In particular:

- Queries extracted directly from chunks result in accurate retrievals, as expected.
- Generalized queries maintain a high similarity, demonstrating the system's ability to capture meaning even when it is expressed in more abstract terms.
- Realistic queries formulated in natural language are effectively understood by the system, yielding high average similarity scores and coherent documents.

Vertex AI demonstrated greater semantic consistency in the selection of documents, particularly for the user-interest queries, suggesting a more effective internal handling of large-scale retrieval. The adopted similarity metrics proved to be valid tools for analyzing the quality of the results in terms of both relevance and coherence.

5.4 LLM Model Configuration

Following the deployment of the semantic retrieval component, the system was extended with a Large Language Model to support complex query answering and automated document generation. This phase involved the design of a carefully crafted prompt that constrained the model's output to rely solely on the user's query, the retrieved context, and the conversation history, while enforcing a formal and domain-specific tone. An iterative optimization process was employed to reduce hallucinations and ensure compliance with administrative standards.

The LLM was integrated incrementally, beginning with a reduced document subset and expanding to a stratified sample representative of the full dataset. A MultiVector Retrieval strategy was adopted to retrieve entire resolutions rather than individual chunks, enhancing semantic coverage at the cost of increased computational demand. For generation tasks, the Gemini 2.0 Flash model was selected due to its efficiency and support for multi-turn interactions.

Evaluation was structured around two query categories: search-oriented queries, aimed at retrieving precise information, and generation-oriented queries, focused on producing new administrative resolutions. The latter received greater emphasis, reflecting the system's core objective.

5.4.1 Evaluation of Search Queries

The initial testing phase was conducted to validate the model's responses to information retrieval requests from existing resolutions. This phase revealed that responses generated from overly generic search queries were often incomplete, imprecise, or insufficiently informative. In some cases, the model relied on a single document chunk, producing content that was not fully representative and sometimes improperly including real names.

In response to these issues, a process of query reformulation was initiated, orienting queries towards more precise and specific questions, for example, regarding names, amounts, dates, or comparisons between different resolutions. Concurrently, the prompt was enriched with concrete examples (few-shot prompting approach), which helped the model better contextualize its generation while respecting the expected formal and content conventions.

Further improvement was achieved by reducing the model's temperature parameter to 0.1, which stabilized responses and limited undesired creativity. These interventions resulted in a significant increase in accuracy and relevance, enabling the system to generate detailed, faithful responses aligned with user expectations.

During testing with specific search queries aimed at retrieving exact resolutions through synthesis requests or full content extraction, a notable limitation was observed in the model's ability to correctly identify such documents. Detailed analysis showed that this limitation was primarily due to the retrieval component rather than the text generation module.

To address this, an optimization of the key parameters governing the behavior of the vector-based semantic search engine was undertaken. Following a series of empirical experiments, the following configuration values were adopted:

- **Number of nearest neighbors considered in the search**: set to 2200. This parameter defines the number of candidate embeddings analyzed during semantic matching. A higher value enables the system to explore a larger portion of the vector space, increasing the likelihood of identifying semantically relevant documents, at the expense of greater computational time.

- **Percentage of leaf nodes explored**: configured to 0.8. This indicates the fraction of leaf nodes in the vector index visited during the search. Higher values improve system recall at the cost of efficiency. The choice of 0.8 represents a balanced trade-off between accuracy and performance.

- **Number of embeddings per leaf node**: fixed at 100. This parameter sets the granularity of indexing, determining how many semantic vectors are stored in each leaf node. Controlled distribution helps maintain stable local density and ensures consistent performance even on heterogeneous datasets.

- **Number of documents retrieved by the MultiVector Retriever**: specified as 15. After semantic exploration, the system returns the 15 documents most similar to the input query. This value was selected empirically to provide a sufficient number of candidates to satisfy user information needs without introducing excessive noise.

These parameters were calibrated and validated on a subset comprising 80% of the dataset, consisting of resolutions from the years 2023 and 2024, demonstrating a significant improvement in the system's ability to retrieve relevant documents. Notably, it emerged that as the dataset size increases, it becomes necessary to proportionally increase the approximate number of neighbors considered to explore a larger number of candidate documents and thus enhance retrieval effectiveness.

Nonetheless, the system does not guarantee exhaustive retrieval in all cases, making it necessary to enrich queries with additional contextual information such as the proposal number

or the subject of the resolution, beyond standard fields like number and date. Enriching queries with these elements helps reduce ambiguity and increases retrieval precision.

One of the most important challenges involved protecting sensitive data and thus managing data confidentiality and security. Initially, the prompt enforced a complete redaction of proper names, entities, companies, numeric values, and official references. However, this approach was overly restrictive and compromised the informational completeness of responses. Following further iterations, a more effective compromise was established: maintaining censorship on the names of natural and legal persons, while allowing the display of economic data, references to tables, and past communications when explicitly present in the document context.

This strategy enhanced the informativeness of responses while maintaining a good balance between data security and practical utility for the user. The results obtained during the second testing phase, conducted on a dataset where all names had already been redacted using placeholders (`NAME`), confirmed improvements in coherence, contextual adherence, and information protection.

5.4.2 Evaluation of Generation Queries

In parallel with the testing of search queries, the system was extended to allow the automatic generation of new resolutions based on a structured context. The objective was to test the LLM's ability to faithfully reproduce the formal structure and technical-administrative language of official documents. The model was therefore required not only to understand the internal logic of the text but also to adapt to a specialized linguistic register, respecting the standard form of real resolutions.

Automatic generation involved constructing well-defined sections: proposal number, names of responsible parties, subject of the resolution, preambles with regulatory references, and finally, the actual deliberative section containing approved actions. Again, the model was constrained to the provided context and was never allowed to generate arbitrary content. The inclusion of placeholders and the application of pre-established prompt rules effectively controlled text production.

An iterative approach played a fundamental role, allowing the precise definition of the logical and textual structure of generated resolutions. Through progressive refinement of prompts and templates, the system was guided to produce documents conforming to formal and administrative standards, gradually acquiring greater consistency in section arrangement and the use of technical language.

In particular, the controlled use of placeholders proved an effective strategy for representing sensitive or as-yet-unknown elements at the time of initial generation. Such placeholders serve as structural markers which, during the iterative process or upon explicit user request, can be automatically replaced with specific information provided. This mechanism both maintains the formal correctness of the resolution in the absence of complete data and supports user-guided insertion of critical content such as proposal numbers, responsible parties'names, or expense amounts.

Another aspect addressed during the design phase was the recognition of the thematic domain of the resolution to be generated. Through careful semantic analysis of the provided context, the system was guided to automatically recall the most common and relevant regulations and legal references pertinent to that domain, for example, frequently cited legal articles or internal organizational regulations. At the same time, generation remained sufficiently flexible to allow end users to integrate additional specific regulatory references, promoting a balance between automatic accuracy and human control. In summary, the adopted approach enabled not only the generation of formally correct documents but also their dynamic customization, facilitating template adaptation to contextual needs and actually available data.

It is worth highlighting a relevant distinction that emerged during the two types of testing conducted. In the case of search queries, expanding the dataset from the initial sample of 29

resolutions to a larger portion corresponding to 80% of the full corpus from 2023 and 2024 yielded a tangible increase in the quality of system-generated responses. Specifically, there was a significant improvement in completeness, informational coherence, and contextualization relative to the posed query. This outcome is attributable to the greater variety and semantic coverage introduced by the new documents, which enabled the retrieval engine to draw from a richer and more diverse pool of relevant content.

Conversely, in the tests relating to automatic generation of new resolutions, dataset expansion did not produce evident benefits in terms of accuracy or text quality. The model demonstrated comparable performance starting from the smaller sample of 29 resolutions, presumably due to the structured and recurrent nature of administrative documents. In this context, the presence of well-formatted representative examples proved sufficient for learning and correctly reproducing the required formal structure.

Nonetheless, to ensure methodological consistency and maximize potential coverage, the 80% dataset portion was retained also for generation tests. This choice ensured coherence between the two experimental domains and allowed validation of system behavior on a broader and more representative basis, although not strictly necessary for generation purposes.

Another phase of the workflow involved transforming model-generated resolutions into Word documents formatted according to the style of official resolutions. This task required designing a post-processing script capable of interpreting and converting annotations in the raw text into formatting consistent with the original documents. Starting from output containing textual markers (e.g., `**bold**`, `*list`), the script applied custom rules to reconstruct the visual structure of the resolution. Interventions included inserting the entity's logo, defining margins, spacings, and paragraph styles, as well as adding page numbers and headers. The final result is a coherent, readable `.docx` file ready for possible manual revision.

5.5 Output Examples

To provide a concrete overview of the functionalities offered by the application, this section presents representative examples of the responses produced in the different operational modes, which are drawn from real cases effectively processed by the implemented system.

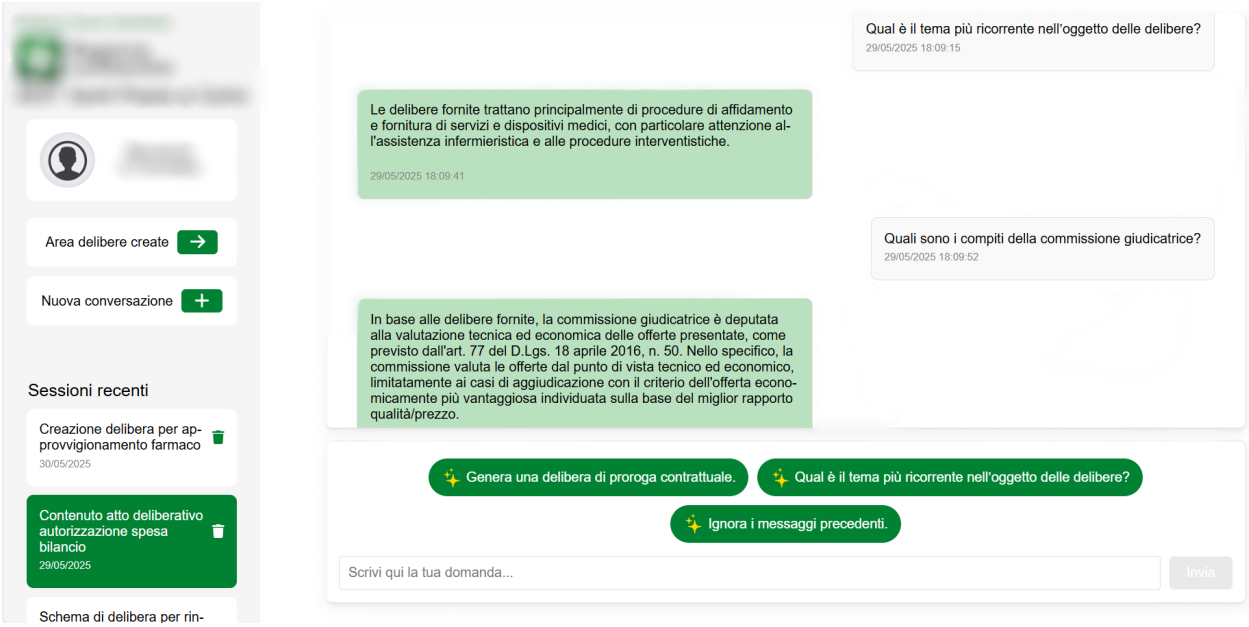


Figure 5.15: Example of a response to search queries.

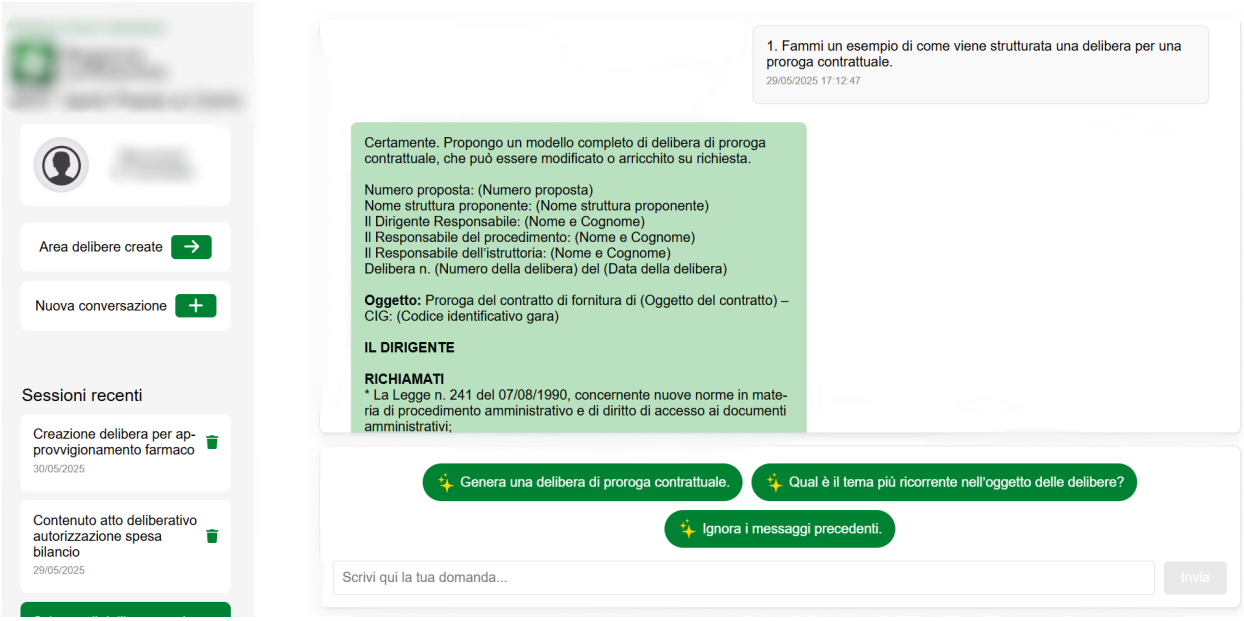


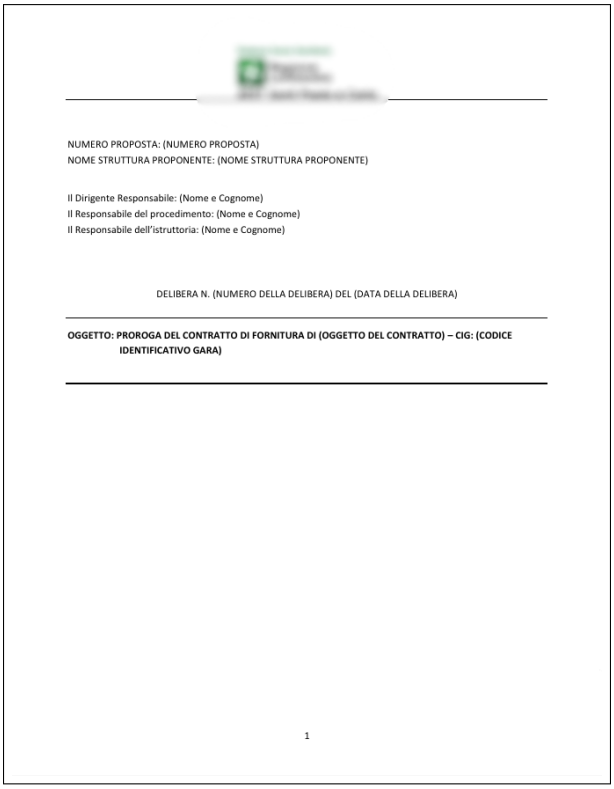
Figure 5.16: Example of automatic generation of a resolution.

- **Search queries**: Figure 5.15 shows the user interface following the submission of specific informational requests. The system returns concise and coherent answers, enriched with relevant textual elements.
- **Resolution generation**: Figure 5.16 illustrates the response generated from a request for the automatic creation of a document. The interface displays the initial part of the text produced by the LLM in conversational mode.
- **Formatted document**: Figures 5.17 and 5.18 report the final output in Word format, automatically generated by the system. The document complies with the editorial conventions of the organization, featuring well-defined sections, coherent layout, and the correct insertion of static and dynamic data.

In the resolution generated in Figure 5.16, some placeholders can be observed within the text. This behavior is expected, as the initial request was formulated generically and lacked specific details. The user thus has the possibility to manually customize the template by enriching it with desired information, or alternatively to continue interacting with the system by providing additional details to obtain an updated and complete version of the document.

A similar behavior can also be observed in Figures 5.17 and 5.18. The presence of placeholders indicates the flexible nature of the generation process, allowing the user to insert all missing information. This feature ensures that the document can be progressively refined until it reaches the desired level of completeness and adherence to institutional standards. Moreover, the figures highlight the structural organization of the document, which follows the conventions discussed in the previous chapter. Specifically, the resolution is articulated into clearly separated sections: an introductory part containing the proposal number, the parties involved, and the subject of the resolution; a central section dedicated to the legal framework, in which relevant laws, regulations, and administrative acts are referenced; and a final deliberative section, where the specific actions and decisions formally adopted by the institution are reported. This tripartite division, in line with organizational practice, not only facilitates readability but also guarantees compliance with procedural and editorial requirements.

Through these examples, it becomes evident how the system supports both automation and user control, striking a balance between efficiency in document drafting and the flexibility necessary to accommodate case-specific details.



NUMERO PROPOSTA: (NUMERO PROPOSTA)
NOME STRUTTURA PROPONENTE: (NOME STRUTTURA PROPONENTE)

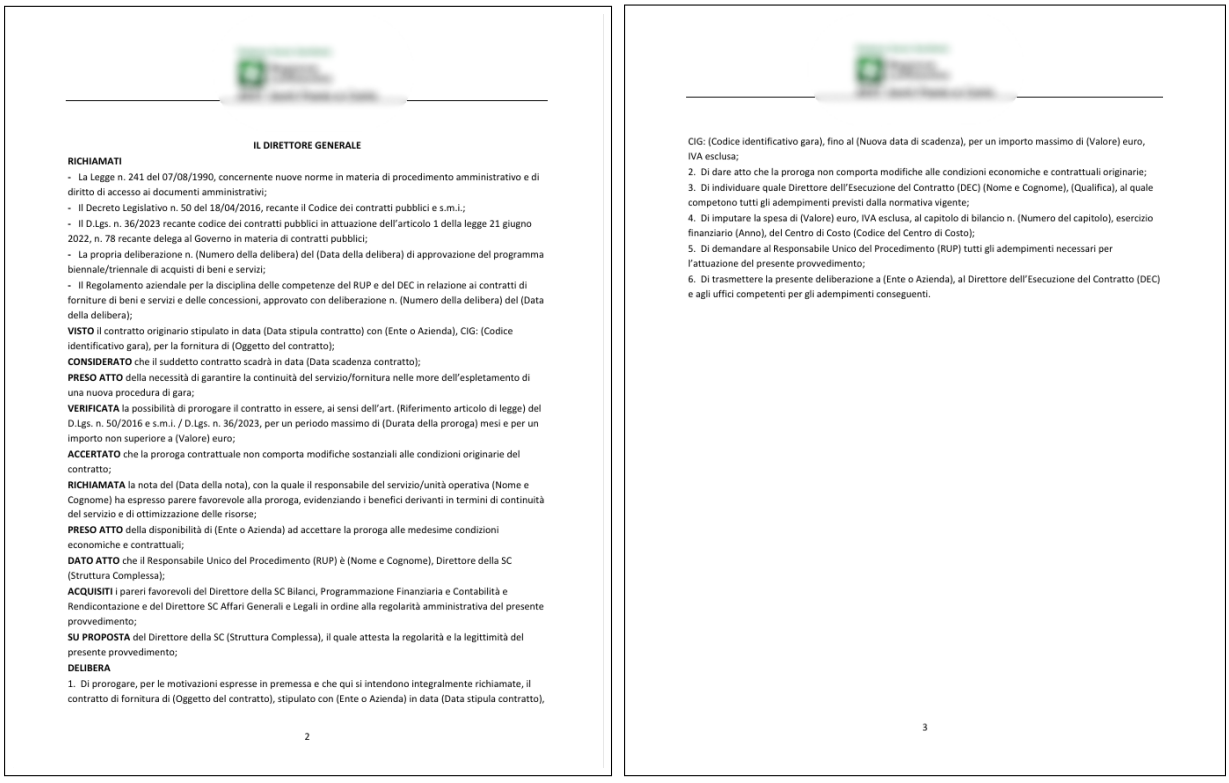
Il Dirigente Responsabile: (Nome e Cognome)
Il Responsabile del procedimento: (Nome e Cognome)
Il Responsabile dell'istruttoria: (Nome e Cognome)

DELIBERA N. (NUMERO DELLA DELIBERA) DEL (DATA DELLA DELIBERA)

OGGETTO: PROROGA DEL CONTRATTO DI FORNITURA DI (OGGETTO DEL CONTRATTO) – CIG: (CODICE IDENTIFICATIVO GARA)

1

Figure 5.17: First page of the generated and formatted resolution.



IL DIRETTORE GENERALE

RICHIAMATI
- La Legge n. 241 del 07/08/1990, concernente nuove norme in materia di procedimento amministrativo e di diritto di accesso ai documenti amministrativi;
- Il Decreto Legislativo n. 50 del 18/04/2016, recante il Codice dei contratti pubblici e s.m.i.;
- Il D.Lgs. n. 36/2023 recante codice dei contratti pubblici in attuazione dell'articolo 1 della legge 21 giugno 2022, n. 78 recante delega al Governo in materia di contratti pubblici;
- La propria deliberazione n. (Numero della delibera) del (Data della delibera) di approvazione del programma biennale/triennale di acquisti di beni e servizi;
- Il Regolamento aziendale per la disciplina delle competenze del RUP e del DEC in relazione ai contratti di forniture di beni e servizi e delle concessioni, approvato con deliberazione n. (Numero della delibera) del (Data della delibera);

VISTO il contratto originario stipulato in data (Data stipula contratto) con (Ente o Azienda), CIG: (Codice identificativo gara), per la fornitura di (Oggetto del contratto);
CONSIDERATO che il suddetto contratto scadrà in data (Data scadenza contratto);
PRESO ATTO della necessità di garantire la continuità del servizio/fornitura nelle more dell'espletamento di una nuova procedura di gara;
VERIFICATA la possibilità di prorogare il contratto in essere, ai sensi dell'art. (Riferimento articolo di legge) del D.Lgs. n. 50/2016 e s.m.i. / D.Lgs. n. 36/2023, per un periodo massimo di (Durata della proroga) mesi e per un importo non superiore a (Valore) euro;
ACCERTATO che la proroga contrattuale non comporta modifiche sostanziali alle condizioni originarie del contratto;
RICHIAMATA la nota del (Data della nota), con la quale il responsabile del servizio/unità operativa (Nome e Cognome) ha espresso parere favorevole alla proroga, evidenziando i benefici derivanti in termini di continuità del servizio e di ottimizzazione delle risorse;
PRESO ATTO della disponibilità di (Ente o Azienda) ad accettare la proroga alle medesime condizioni economiche e contrattuali;
DATO ATTO che il Responsabile Unico del Procedimento (RUP) è (Nome e Cognome), Direttore della SC (Struttura Complessa);
ACQUISITI i pareri favorevoli del Direttore della SC Bilanci, Programmazione Finanziaria e Contabilità e Rendicontazione e del Direttore SC Affari Generali e Legali in ordine alla regolarità amministrativa del presente provvedimento;
SU PROPOSTA del Direttore della SC (Struttura Complessa), il quale attesta la regolarità e la legittimità del presente provvedimento;
DELIBERA

1. Di prorogare, per le motivazioni espresse in premessa e che qui si intendono integralmente richiamate, il contratto di fornitura di (Oggetto del contratto), stipulato con (Ente o Azienda) in data (Data stipula contratto),

2

CIG: (Codice identificativo gara), fino al (Nuova data di scadenza), per un importo massimo di (Valore) euro, IVA esclusa;
2. Di dare atto che la proroga non comporta modifiche alle condizioni economiche e contrattuali originarie;
3. Di individuare quale Direttore dell'Esecuzione del Contratto (DEC) (Nome e Cognome), (Qualifica), al quale competono tutti gli adempimenti previsti dalla normativa vigente;
4. Di imputare la spesa di (Valore) euro, IVA esclusa, al capitolo di bilancio n. (Numero del capitolo), esercizio finanziario (Anno), del Centro di Costo (Codice del Centro di Costo);
5. Di demandare al Responsabile Unico del Procedimento (RUP) tutti gli adempimenti necessari per l'attuazione del presente provvedimento;
6. Di trasmettere la presente deliberazione a (Ente o Azienda), al Direttore dell'Esecuzione del Contratto (DEC) e agli uffici competenti per gli adempimenti conseguenti.

3

Figure 5.18: The left side shows the central section of the generated and formatted resolution, while the right side displays its final section.

5.6 Testing and Validation

This section presents the testing and validation activities carried out on the developed system, aimed at evaluating the effectiveness of its main functionalities: semantic search and automatic generation of resolutions. The evaluations were conducted both on a reduced subset of documents (29 resolutions) for initial qualitative and visual validation, and on a larger dataset (first covering 20% and then 80% of the total) to verify the system's robustness and generalizability.

5.6.1 Types of Tests

To assess the performance of the developed system, a structured testing phase was conducted based on three distinct categories of user interactions:

1. **Direct generation**: Consists of submitting a single query containing most of the relevant information for generating a resolution (such as number, date, proposal number, subject). This pathway simulates a one-shot generation process, aimed at testing the system's ability to correctly learn and replicate the resolution structure from detailed input.

2. **Incremental generation**: Involves a multi-turn interaction, where information is progressively provided throughout the conversation. This pathway allows assessing the system's capability to maintain and utilize dialogue history to produce a coherent and complete resolution.

3. **Search or information retrieval**: Includes specific informational requests or queries for retrieving particular resolutions. This category tests the system's effectiveness in the semantic retrieval phase and in recalling targeted content.

Example queries for each of the three mentioned categories are presented below.

1. Examples of direct generation queries:

   - Create a resolution to stipulate an insurance policy necessary for carrying out the Esperiment trial for a total amount of 1,000 euros with the company *Example Insurance S.p.A.*. Insert proposal number 123 and assign *Name Surname A* as the responsible manager. The resolution has number 567 and date 26/05/2025. Insert *Name Surname B* as procedure manager and *Name Surname C* as investigation manager.
   - Draft a resolution approving a four-year program 2026–2030 for purchases of goods and services amounting to two million euros. Insert proposal number 123 and assign *Name Surname A* as responsible manager. The resolution has number 567 and date 26/05/2025. Insert *Name Surname B* as procedure manager and *Name Surname C* as investigation manager.

2. Examples of incremental generation queries:

   - (a) Generate a resolution regarding the assignment of hospital material supply.
     (b) The assignment is awarded to *Example Insurance S.p.A.* for the supply of airway humidification systems from 15/05/2025 to 15/06/2025 for an amount of 130,000 euros.
     (c) Add proposal number 123 and responsible manager *Name Surname A*.
     (d) The resolution has number 567 and date 26/05/2025.
     (e) Add procedure manager *Name Surname B* and investigation manager *Name Surname C*.
   - (a) I need a sample resolution for a contract renewal.

(b) The contract concerns the in-service supply of an ultrasound system intended for the *Example Department* of the *Example Hospital*. The renewal period will be from 10/07/2025 to 17/08/2025 for a total amount of 60,000 euros.
(c) Add proposal number 123 and responsible manager *Name Surname A*.
(d) The resolution has number 567 and date 26/05/2025.
(e) Add procedure manager *Name Surname B* and investigation manager *Name Surname C*.

3. Examples of search or information retrieval queries:

- What is the most recurring theme in the subject of the resolutions?
- What hospital material was supplied in the year 2024?
- What do the resolutions that authorize budget expenditures specify?

5.6.2 Evaluation of System Accuracy and Robustness

For each of the three categories, 10 distinct queries were prepared; each direct generation and search query was tested 6 times, while each incremental generation query was tested 3 times. For each query, the number of correct answers was recorded, and a score between 0 and 6 was assigned according to the number of satisfactory responses obtained. During the internal evaluation phase, the most recurrent errors were also noted, including:

- Omission or incorrect insertion of requested information;
- Incorrect positioning of information within the document;
- Loss of context or dialogue history in incremental generation;
- Failure to retrieve the correct resolution during search queries.

Identifying these issues enabled targeted corrections in prompt engineering, progressively improving the system's reliability and accuracy.

In aggregating the results, a unit weight (1.0) was assigned to direct and incremental generation queries, and a reduced weight (0.5) to search queries, as generation represents the system's primary objective. To obtain an overall metric, the weighted average of the scores was calculated and normalized on a 100-point scale. The final value obtained was 82.58, indicative of the system's strong capability to generate coherent and correct documents across a variety of use scenarios.

Following this initial evaluation, a series of targeted enhancements were applied to the system, particularly in the areas of prompt design, conversational memory handling, and output post-processing. Once these improvements were implemented, the entire test suite was re-executed under identical conditions. The updated evaluation yielded a normalized score of 87.04. This increase of approximately 4.5 points demonstrates a significant improvement in the system's performance. From a technical standpoint, the gains were primarily observed in the incremental generation and complex query categories, where the improvements in context retention and semantic control contributed to higher output quality. Furthermore, the reduction of formatting errors and better handling of placeholders during document assembly led to more accurate and publication-ready results. These findings confirm that even small refinements in the generation pipeline can yield measurable improvements in both functional and formal dimensions.

On the other hand, the system underwent a functional testing phase through the controlled execution of all planned test queries. The observed behavior confirmed the reliability of the entire pipeline, both in terms of textual generation and management of semantic constraints.

In particular, the incremental approach proved to be the most effective, as it allows greater controllability over the generated content, enabling the user to guide the document composition in multiple steps, thereby reducing the risk of errors and ensuring closer adherence to expected specifications.

From an editorial quality perspective, the system demonstrated high accuracy in formatting and reproducing the formal structure of resolutions, respecting the required constraints. Moreover, the model effectively manages the use of placeholders for sensitive or missing data, ensuring that they are properly replaced when the user provides further details. Finally, a good adaptability of the system to user requests was observed, with the possibility of inserting additional sections or modifying the standard document structure, confirming the robustness of the system even in dynamic or not fully predefined scenarios.

5.6.3 Scheduling Module Testing

During the testing phase, the direct generation queries were used to validate the correct functioning of the automatic scheduling module. Specifically, batches were created with deadlines distributed over multiple days, including deadlines falling immediately after weekends and holidays, in order to cover realistic temporal scenarios and verify that scheduling occurred without errors.

The tests confirmed that the system correctly assigns scheduling slots starting at 08:00 on the day of the deadline, avoiding overlaps and consistently respecting the 5-minute interval between consecutive batches. Even in cases where multiple batches were scheduled for the same day, the assignment logic ensured that each was positioned in the first available slot.

Finally, the execution of scheduled jobs took place as expected, with each batch correctly inserted into the processing queue and handled by the thread pool in the intended order. No errors or unexpected behaviors were observed, confirming the robustness and reliability of the scheduling component.

Chapter 6

User Interface and Interaction Modalities

This chapter presents the design of the user interface and the different interaction modalities supported by the system. It illustrates how users can query documents, generate new resolutions, and schedule requests, emphasizing usability and accessibility.

6.1 User Dashboard

The main interface of the system is shown in Figure 6.1. It is divided into several functional areas, each designed to facilitate interaction with the agent and the management of administrative resolutions.

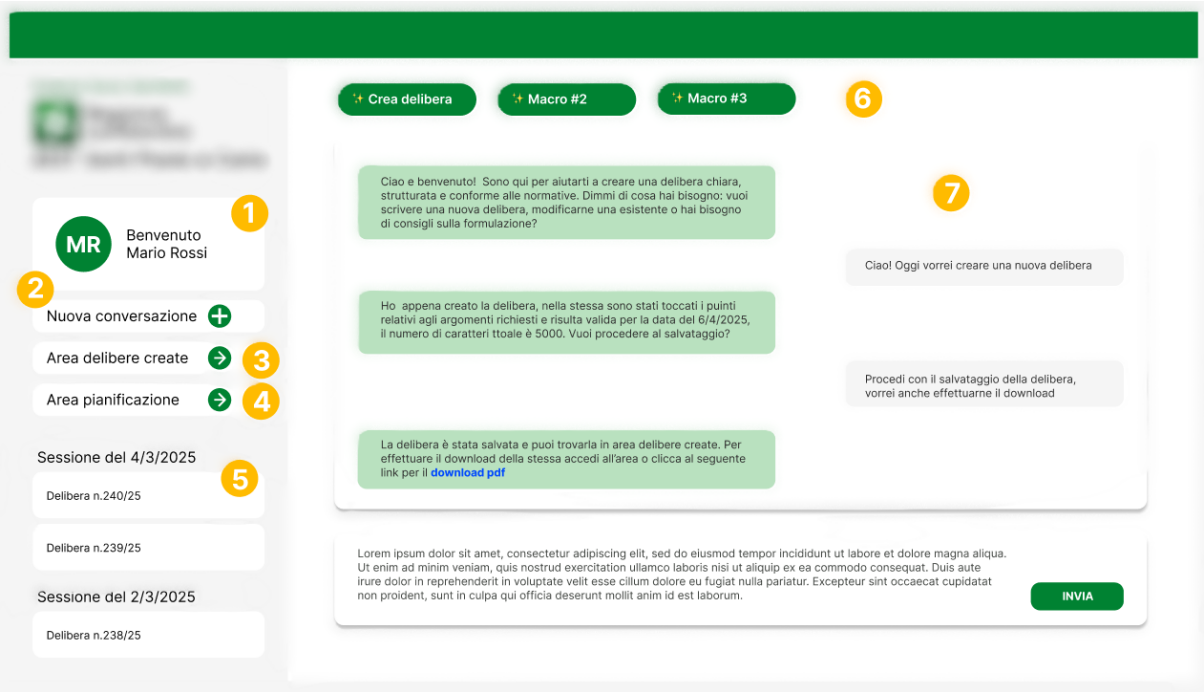


Figure 6.1: Main interface for interaction with the agent

1. **User profile**: Displays the name and identity of the user currently authenticated within the system.
2. **New conversation**: Allows the user to initiate a new dialogue session with the agent, useful for generating a new resolution or submitting an informational request.

3. **Created resolutions area**: Provides access to the section dedicated to searching, viewing, and managing previously drafted resolutions.

4. **Planning area:** Provides access to the section where users can schedule the creation of resolutions for a specific date.

5. **Session history**: Contains the list of previous conversations, each identified by a brief and representative title.

6. **Macro buttons**: Offer quick access to predefined functionalities, such as clearing the session history.

7. **Chat area with the agent**: Serves as the main space for textual interaction with the system, where the user can submit drafting requests or inquiries about the content of the resolutions.

6.2 Resolution Management and Search

Figure 6.2 illustrates the second interface of the system, dedicated to the management and search of previously generated resolutions. This section can be accessed from the main dashboard by clicking on the "Created Resolutions Area". It enables users to browse the generated documents, apply search filters, and perform management actions such as downloading or deleting files.

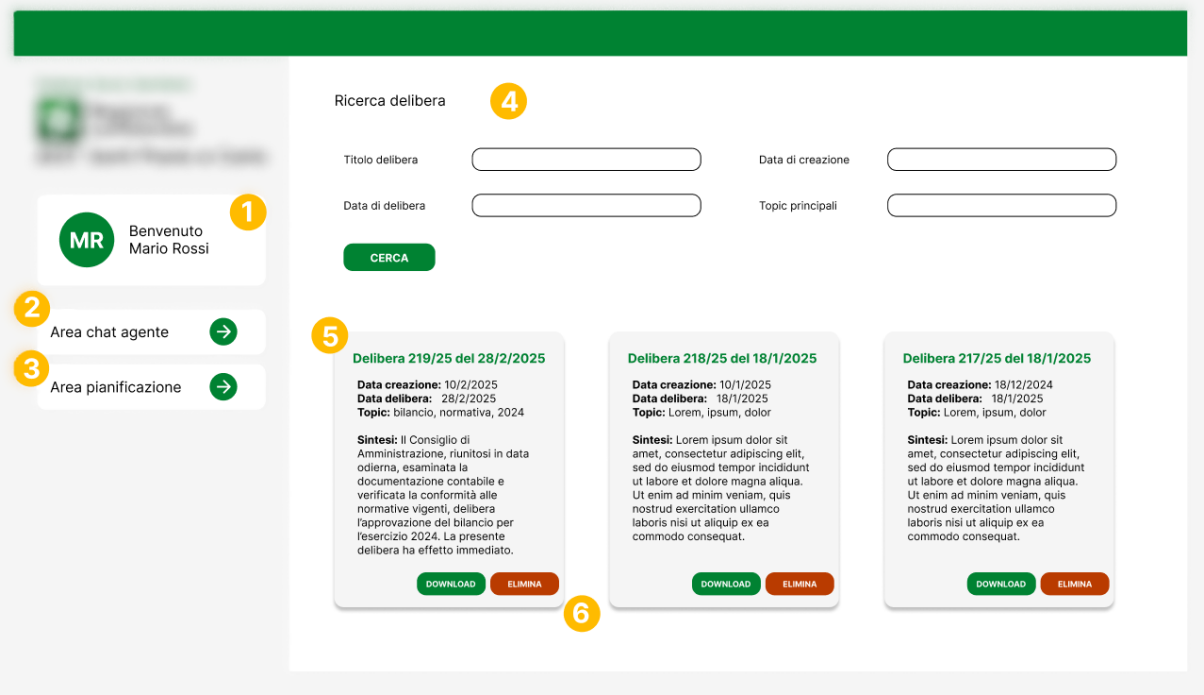


Figure 6.2: Interface for resolution search and management

1. **User profile**: Displays the name and identity of the currently authenticated user.

2. **Agent chat area**: Provides a shortcut to return to conversational mode for continuing the interaction with the agent.

3. **Planning area:** Provides access to the section where users can schedule the creation of resolutions for a specific date.

4. **Search filter**: Allows the user to search for specific resolutions using keywords, creation dates, or relevant topics.
5. **Resolution cards**: Each card offers a concise representation of a resolution, including the title, main topics, date, and an automatically generated summary of the content.
6. **Available actions**: Each resolution can be downloaded in Word format (.docx) or removed from the archive via the respective buttons.

6.3 Resolution Scheduling

This area provides users with a dedicated interface to plan the creation of resolutions in advance. By entering the text of the requested documents and specifying a desired completion date, users can efficiently organize upcoming tasks and ensure that all resolutions are prepared in a timely manner. As illustrated in Figure 6.3, this interface enables users to enter the content of resolutions to be drafted, specify the intended completion date for each resolution and automatically display scheduled resolutions in the "Created Resolutions Area" on the specified date, clearly distinguished by a badge.

This functionality streamlines the planning process, improves document management, and ensures that all requested resolutions are readily accessible and easily identifiable on their scheduled day.

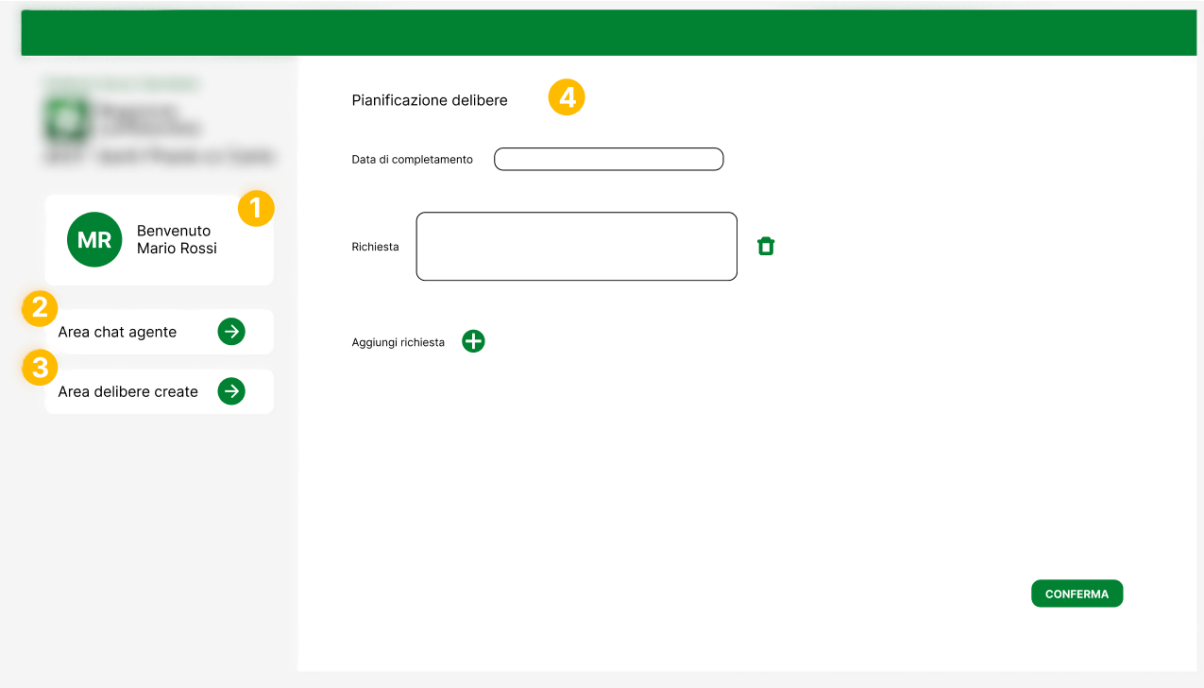


Figure 6.3: Interface for resolution scheduling.

1. **User profile**: Displays the name and identity of the currently authenticated user.
2. **Agent chat area**: Provides a shortcut to return to conversational mode for continuing the interaction with the agent.
3. **Created resolutions area**: Provides access to the section dedicated to searching, viewing, and managing previously drafted resolutions.
4. **Schedulation area**: Allows the user to specify a completion date and enter the text of the requested resolutions.

6.4 Creation and Management of a New Request

The user can formulate a new request by entering a textual prompt into the dedicated input dialog. This prompt may consist of an open-ended question, an administrative requirement, or a general indication regarding an act to be drafted.

Upon submission:

1. The system locally saves the request within the user's database;
2. The semantic retrieval engine is activated to identify relevant documents;
3. The output is generated by the LLM and subsequently saved as part of the conversation.

The system's response is displayed on the same screen as a formatted text block.

If the generated response corresponds to a new administrative act, the user can initiate the procedure to generate a Word document, during which the system:

1. Integrates the generated content into a template compliant with the organization's standards;
2. Applies the desired formatting (headings, centered paragraphs, logos, informational boxes);
3. Saves the file in a dedicated storage and makes it available for download.

Each request and response is recorded in a database dedicated to the user. This allows consultation of the history, reuse of past acts, and analysis of usage behavior by system administrators.

6.4.1 Facilitated Interaction via Macro Buttons

On the screen dedicated to creating a new request, the user can interact with the system not only through free-text input but also via a set of predefined buttons, each representing a conversational *macro*, as illustrated in Figure 6.1. These buttons enable quick activation of specific functionalities, thereby reducing cognitive load and accelerating the initiation of interactions.

In particular, the button labeled *Ignore Previous Messages* allows the user to reset the active conversation state, forcing the system to disregard the entire preceding contextual history and start anew. This functionality is useful when initiating a new request within the same conversation that is completely independent from prior interactions, thus ensuring a neutral generation not influenced by historical context. This mechanism implements controlled session management, thereby enhancing the user experience.

6.4.2 Creation of a Resolution

The creation of a new resolution occurs through interaction with the conversational agent within the dialogue area of the main interface. The user can choose between two operational modes, both aimed at the automatic generation of a document that complies with the required technical and administrative standards.

1. **Direct mode**. In this mode, the user submits a request that already includes all the necessary information for drafting the resolution. The system immediately generates a complete document, ready for validation or saving. This option is particularly useful when the user has already defined all relevant elements, such as the subject of the resolution, normative references, involved parties, and actions to be resolved.

2. **Incremental mode**. The user may adopt an incremental approach, progressively specifying the various pieces of information to be included in the document, such as the proposal number, procedure managers, and the document's subject. This mode offers more precise control and detailed customization of the content, making it especially suitable for a more deliberate and accurate management of the generated resolution.

Once generation is complete, the text of the resolution is displayed directly within the dialogue window. If satisfied with the result, the user can click the "Save to Resolutions Area" button, which allows archiving the document in the dedicated "Created Resolutions Area". Within this section, each saved document is presented as a card displaying the title, an automatically generated summary, the creation date, and the associated topic. Each resolution can subsequently be:

- Deleted, if no longer needed;
- Downloaded in `.docx` format for further editing (such as manually replacing placeholders with the correct information).

It is important to emphasize that the conversation with the agent can continue even after the initial resolution generation, allowing the user to make further modifications, add new sections, or update existing ones in an iterative process.

6.4.3 Information Retrieval within Resolutions

In addition to the automatic generation of documents, the system enables the user to perform semantic searches within the document corpus. This functionality is useful for quickly retrieving relevant information contained in archived resolutions, both for informational purposes and to support the drafting of new texts.

The user can formulate questions or requests in natural language within the same dialogue area with the agent. The system interprets the request and provides an answer based on the existing documents. The conversation with the agent can continue after each result to further explore the retrieved information or request its reformulation.

The system supports a wide range of informational requests, including:

- Searching for resolutions addressing a specific topic, entity, or type of intervention.
- Extracting normative references used in similar contexts.
- Identifying recurring data, such as costs, dates, involved parties, or resolution outcomes.
- Comparing resolutions that present structural or thematic similarities.

6.4.4 Placeholder Management

In both approaches, if certain information is missing, incomplete, or sensitive (such as names, figures, identification codes, or protected data), the system automatically inserts *placeholders* within the generated text. These typically appear as short labels enclosed in parentheses (e.g., `(NAME)`, `(DATE)`, `(AMOUNT)`) and explicitly and recognizably represent missing or to-be-confirmed information, so that they can be easily identified and replaced during the review or manual compilation phase of the document.

These placeholders serve a dual purpose:

1. Ensuring the privacy and security of sensitive data;

2. Indicating to the user the points in the document that require manual completion or verification.

As soon as the user provides the missing information through the dialogue area, the system can automatically replace the placeholders with the supplied data, updating the document in real time.

Chapter 7

Conclusions

This thesis has presented the design, implementation, and evaluation of an advanced system for the semantic search and automatic generation of administrative resolutions. The developed system demonstrated a high level of effectiveness in generating coherent and accurate documents across a variety of use cases, achieving an overall weighted accuracy score of 87.04% based on rigorous internal testing.

The experimental results highlight the system's ability to replicate the formal structure and content requirements of administrative resolutions, while maintaining flexibility to accommodate user-driven modifications and incremental input. The incremental generation approach, in particular, proved to be the most effective interaction modality, allowing for finer control over the output and reducing the risk of errors by enabling step-by-step content refinement.

From a critical standpoint, the system exhibits notable strengths including robustness in textual generation, adherence to formal formatting standards, and effective management of placeholders for sensitive or missing data. Moreover, the capacity to adapt dynamically to user requests, such as inserting additional sections or modifying the document structure, confirms its practical utility in real-world scenarios characterized by evolving requirements.

Despite these promising outcomes, some limitations persist, primarily in the document retrieval phase. In certain cases, the semantic search fails to precisely retrieve the expected resolution, especially when queries are formulated with vague or nonspecific language. Nonetheless, this limitation is considered partial, since the system's main objective is guided generation of new documents rather than information retrieval. Still, improvements in indexing mechanisms and context enrichment could further enhance retrieval accuracy.

During the experimental phase, and notably through consultations with potential end-users, several directions for system enhancement have been identified that directly address operational needs encountered in practical environments. The key prospective developments include:

- **Document-based guided drafting**: Introduction of the capability to upload one or more existing resolutions as reference documents to generate new ones similar in structure and content. This mode is particularly beneficial for recurring cases requiring documents analogous to previously produced ones, resulting in significant time savings and improved consistency.
- **Automated reporting and analysis**: Development of an analytical module capable of extracting statistics and relevant indicators from the content of resolutions. Examples include the number of approved technical extensions within a period, volume of hospital supplies delivered, or other pertinent trends. Such functionality would provide strategic insights useful for monitoring and planning.
- **Normative validation via web search**: Integration of a web search module enabling the system to verify the validity and currency of legal regulations cited within the resolutions. This would allow the language model to confirm the existence and correctness of

normative sources, thereby increasing the reliability of the generated text.

These future enhancements aim to further strengthen the system's efficiency, customization, and operational intelligence, transforming it into a tool increasingly aligned with the real needs of public administration and capable of supporting document drafting, validation, and monitoring activities with greater effectiveness.

In conclusion, this work represents a significant step towards adopting advanced AI-driven solutions in administrative document management, laying a foundation for continuous improvement and practical application in dynamic and complex organizational contexts.

Bibliography

[1] Jay Alammar. The illustrated transformer. https://jalammar.github.io/illustrated-transformer/, 2018. Accessed: 2025-08-16.

[2] Felipe Almeida and Geraldo Xexéo. Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*, 2019.

[3] Appinventiv. Transformer vs rnn in nlp: A comparative analysis. https://appinventiv.com/blog/transformer-vs-rnn/, 2023.

[4] Baeldung. From rnns to transformers. https://www.baeldung.com/cs/rnns-transformers-nlp, 2023.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. In *IEEE Transactions on Neural Networks*, volume 5, pages 157–166. IEEE, 1994.

[6] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Yulin Chen, Rui Zhang, Ananya Kumar, Wei Li, James Thompson, Maria Hernandez, and Rajesh Gupta. Capabilities of gpt-5 across critical domains: Is it the next breakthrough? *arXiv preprint arXiv:2508.19259*, 2025.

[10] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. ACL, 2014.

[11] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022.

[12] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning*, 2014.

[13] Wikipedia Contributors. Prompt engineering, 2025.

[14] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019*

Conference of the North American Chapter of the Association for Computational Linguistics. ACL, 2019.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[17] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[18] Facebook AI Research. Faiss: A library for efficient similarity search and clustering of dense vectors. https://github.com/facebookresearch/faiss, 2025. Accessed: 2025-08-19.

[19] Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*, 2025.

[20] Python Software Foundation. queue â a synchronized queue class. https://docs.python.org/3/library/queue.html, 2025. Accessed: 2025-08-19.

[21] Google AI. Gemini models: Palm api for generative ai. https://developers.generativeai.google/products, 2025. Accessed: 2025-08-19.

[22] Google Cloud. Vertex ai embeddings models. https://cloud.google.com/vertex-ai/docs/generative-ai/embeddings/get-text-embeddings, 2024. Accessed: 2025-08-17.

[23] Google Cloud. Google cloud documentation, 2025. Accessed: 2025-08-17.

[24] Google Cloud. Infrastruttura per un'applicazione di ai generativa compatibile con rag che utilizza vertex ai e vector search. https://cloud.google.com/architecture/gen-ai-rag-vertex-ai-vector-search?hl=it, 2025. Accessed: 2025-08-18.

[25] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 6645–6649. IEEE, 2013.

[26] Jonathan Haas. Apscheduler: Advanced python scheduler. https://apscheduler.readthedocs.io/, 2025. Accessed: 2025-08-19.

[27] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea preprints*, 1(3):1–26, 2023.

[28] Sepp Hochreiter and JÃ¼rgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[29] Matthew Honnibal and Ines Montani. spacy: Industrial-strength natural language processing in python. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (System Demonstrations)*, 2017.

[30] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79, 2024.

[31] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2021.

[32] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[33] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[34] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[35] LangChain. How to retrieve using multiple vectors per document, 2023. Accessed: 2025-08-19.

[36] LangChain AI. Langchain. https://github.com/langchain-ai/langchain, 2024. Accessed: 2025-08-16.

[37] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880. ACL, 2020.

[38] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich KÃ¼ttler, Mike Lewis, Wen-tau Yih, Tim RocktÃ¤schel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in neural information processing systems*, 2020.

[39] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Holger KÃ¼ttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, 2020.

[40] Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024.

[41] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 2019.

[42] Tomas Mikolov. *Recurrent neural network based language model*. PhD thesis, Brno University of Technology, 2010.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[44] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 2023.

[45] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[46] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318. ACL, 2002.

[47] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[48] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559–572, 1901.

[49] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[50] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2227–2237. Association for Computational Linguistics, 2018.

[51] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237, 2018.

[52] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. In *OpenAI Technical Report*, 2018.

[53] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

[54] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

[55] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. In *Journal of Machine Learning Research*, volume 21, pages 1–67, 2020.

[56] Peter J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[57] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition, 2010.

[58] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

[59] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[60] Luis Espinosa-Anke Steven Schockaert Shahul Es, Jithin James. Ragas: Automated evaluation of retrieval augmented generation. https://docs.ragas.io/en/stable/, 2025. Accessed: 2025-08-19.

[61] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*, 2025.

[62] Arjun Singh and colleagues. Agentic rag: Advanced retrieval-augmented generation for interactive and adaptive systems. In *Proceedings of the 2024 Conference on Empirical Methods in NLP (EMNLP)*, 2024.

[63] Ray Smith. An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, pages 629–633. IEEE, 2007.

[64] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, volume 27, 2014.

[65] François Torregrossa, Robin Allesiardo, Vincent Claveau, Nihel Kooli, and Guillaume Gravier. A survey on training and evaluation of word embeddings. *International journal of data science and analytics*, 11(2):85–103, 2021.

[66] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. OâReilly Media, 2022.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, 2017.

[68] L. Wang et al. A survey on large language model based autonomous agents. *Journal of Computer Science and Technology*, 39(1):1–24, 2024.

[69] Weaviate Team. What is agentic rag. https://weaviate.io/blog/what-is-agentic-rag, 2024. Accessed: 2025-08-16.

[70] Weaviate Team. Advanced rag techniques. Internal slide deck provided by Weaviate, accessible with account credentials. Not publicly available., 2025.

[71] Weaviate Team. Agentic architectures for retrieval-intensive applications. Internal slide deck provided by Weaviate, accessible with account credentials. Not publicly available., 2025.

[72] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45,

2020.

[73] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2nd edition, 2009.

[74] Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[75] W. Zhai. A survey of task planning with large language models. *iComputing*, 2025, 2025.

[76] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.