الجمهورية الجزائرية الديمقراطية الشعبية وزارة التعليم
العالي والبحث العلمي
جامعة سعيدة د. مولاي الطاهر
كلية الرياضيات و الإعلام الآلي و الاتصالات السلكية و اللاسلكية
قسم: الإعلام الآلي

Mémoire de Master en informatique

Spécialité : Intelligence Artificielle : Principes et Applications

T h è m e

Boosting LLMs With external knowledge integration

▪ **Présenté par :**
Nour El Houda KADDOURI

▪ **Dirigé par :**
Dr Ahmed ZAHAF

Année universitaire 2025-2026

Acknowledgements

All praise is due to **Allah**, by whose grace good deeds are accomplished, who illuminated my path and granted me the strength to complete this modest work after a journey marked by effort and challenges.

With deep gratitude and heartfelt appreciation, I extend my sincere thanks to **my beloved parents**, who have been my constant support at every step and my source of strength in moments of weakness. To you, I dedicate the fruit of this effort. May Allah grant you long lives, bless you with good health, and reward you abundantly for the values of patience, faith, and perseverance you have instilled in me.

I would also like to express my profound appreciation to **my supervisor, Dr Ahmed ZAHAF** , who has been far more than an academic mentor—he has been a true role model and a source of inspiration. He taught me not only how to seek knowledge, but also how to be patient in its pursuit. His valuable guidance, sincere advice, and continuous support had a significant impact on overcoming difficulties and completing this research. I am deeply grateful for his trust and encouragement, and I pray for his continued success and contribution to knowledge.

I am equally grateful to **all members of my dear family**, whose quiet support and unwavering encouragement have always been with me. My sincere thanks also go to all my teachers, from primary school to university, who have each left a meaningful mark on my educational journey and contributed to shaping the path that brought me to this moment.

Finally, I owe a word of appreciation **to myself** —for enduring when the path was difficult, for persevering when doubt began to arise, and for believing in my ability to succeed despite all circumstances.

Dedication

I dedicate this work to **my beloved parents** and my family for their endless support and encouragement.

To the memory of **my beloved grandfather**, who will always live in my heart, with love and gratitude for his lasting presence and influence in my life. May Allah forgive him, have mercy on his soul, and grant him a place in Paradise.

I dedicate this work to **my beloved sisters Cha¨ımaa** and **houda**. Thank you for your endless support, encouragement, and kindness. Your presence has always been a source of strength and motivation. I love you.

Table of Contents

List of Figures

List of Tables

Introduction

Artificial intelligence and Natural Language Processing (NLP) have undergone a profound transformation over the past decade, driven by the emergence of Large Language Models (LLMs) and advanced reasoning architectures. These systems have demonstrated remarkable capabilities across a wide range of tasks, including text classification, machine translation, question answering, and conversational dialogue [Alammar and Grootendorst, 2024]. However, de-spite their impressive performance, LLMs remain limited by hallucinations, static training knowledge, and insufficient relational reasoning—particularly for knowledge-intensive tasks that require structured and up-to-date information [Alansari and Luqman, 2026].

To address these limitations, Retrieval-Augmented Generation (RAG) has emerged as a powerful paradigm that enriches language model responses by dynamically retrieving relevant information from external knowledge sources. More recently, graph-based extensions such as GraphRAG and Knowledge Graph RAG (KG-RAG), combined with Graph Neural Networks (GNNs) More recently, have further improved semantic reasoning by modeling explicit relationships between entities. These advances open new perspectives for building more accurate, explainable, and knowledge-grounded question answering systems [Lewis et al., 2020].

This work investigates the design, evaluation, and deployment of a **Knowledge Graph RAG system** that integrates multiple reasoning approaches for question answering. The system is built around two complementary interaction modes—Multiple Choice Question (MCQ) answering and Free Question answering—and combines large language models with structured knowledge retrieval from ConceptNet [Speer et al., 2017] and graph-based reasoning via a Graph Neural Network architecture [Zhou et al., 2020].

The work is organized into four main chapters:

Chapter 1 presents the theoretical foundations of language models. It covers the evolution from traditional NLP techniques such as Bag-of-Words and word embeddings to modern Transformer-based architectures. It examines the two dominant paradigms—Masked Language Models (MLMs) such as BERT and Causal Language Models (CLMs) such as GPT—and explores fine-tuning strategies including Supervised Fine-Tuning (SFT), Reinforcement Learning from Human Feedback (RLHF), and Direct Preference Optimization (DPO).

Chapter 2 provides a comprehensive state of the art on Retrieval-Augmented Generation systems. It introduces the three RAG paradigms (Naive, Advanced, and Modular RAG), discusses multimodal RAG extensions, and presents graph-based retrieval approaches including GraphRAG, KG-RAG, and the role of Graph Neural Networks in structured knowledge reasoning.

Chapter 3describes the design and evaluation of our system. Based on insights from state-of-the-art approaches, we propose a system that leverages four distinct retrieval-augmented generation (RAG) methods to exploit the ConceptNet knowledge graph : a Zero-shot LLM Baseline [Touvron et al., 2023], two

KG-RAG [Baek et al., 2023] pipelines with different entity extraction strategies (LLM-based and NLP-based), and a GNN-augmented architecture (QAGNN + RoBERTa) [Yasunaga et al., 2021]. The results of the experimental setup and comparative evaluation of the four methods are analyzed in terms of accuracy, efficiency, and reasoning paradigm.

Chapter 4 presents the deployment of the complete **Knowledge Graph RAG system** as a local web application. It describes the FastAPI backend, the React.js frontend, and the five interaction scenarios available to the user, illustrated with screenshots of the running application.

1 Foundations of Language Models

1.1 Introduction

The field of Natural Language Processing (NLP) has undergone a profound transformation over the past decades, evolving from traditional symbolic and statistical approaches to modern deep learning-based methods. Early techniques, such as Bag-of-Words (BoW) and n-gram models, provided simple yet effective ways to represent text as numerical features. However, these approaches were limited in their ability to capture semantic meaning and contextual dependencies. The advent of neural networks, including Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Convolutional Neural Net-works (CNNs), marked a significant step forward by enabling the modeling of sequential and local patterns in language.

More recently, the emergence of Large Language Models (LLMs) has revolutionized NLP by enabling the learning of rich linguistic, semantic, and contextual representations from large-scale textual corpora. These models have led to substantial improvements across a wide range of applications, including text classification, machine translation, question answering, information retrieval, and conversational systems.

At the core of these advancements lies the Transformer architecture, which introduced the self-attention mechanism (*self-attention*). Unlike previous sequential models, self-attention allows for the efficient modeling of long-range dependencies within text while enabling parallel computation. This innovation has significantly improved both performance and scalability, establishing Transformers as the dominant architecture in modern NLP.

Building upon this foundation, two primary paradigms of language modeling have emerged: Masked Language Models (MLMs), such as *BERT*, and Causal Language Models (CLMs), such as *GPT*. MLMs leverage bidirectional context to learn deep representations of language, making them particularly effective for understanding tasks. In contrast, CLMs operate in an autoregressive manner, generating text sequentially based on preceding tokens, which makes them well-suited for generative applications.

Despite their strong generalization capabilities, pretrained language models are not inherently optimized for specific downstream tasks or user requirements. This limitation has motivated the development of fine-tuning strategies, which adapt pretrained models to particular domains, tasks, and behavioral expectations. Traditional approaches typically rely on task-specific classification heads; however, these methods introduce architectural rigidity, require labeled data, and lack zero-shot generalization capabilities.

To address these limitations, recent research has explored more unified and efficient paradigms, including the generative use of the native MLM head. In this approach, tasks are reformulated as masked token prediction problems, allowing models to produce outputs as single tokens (*Answer Token Prediction*) without the need for additional task-specific components. When combined with instruction tuning, this paradigm enhances parameter efficiency, supports

zero-shot generalization, and enables a unified framework for handling multiple tasks.

In parallel, modern LLM training pipelines have evolved into multi-stage processes that include pretraining, supervised fine-tuning (Supervised Fine-Tuning, SFT), and preference-based optimization techniques such as Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO). These methods not only improve task performance but also align model behavior with human expectations in terms of usefulness, safety, and clarity.

This chapter provides a comprehensive overview of the foundations of language models and their evolution toward modern LLMs. It begins with fundamental concepts in artificial intelligence and NLP, followed by an examination of deep learning approaches for language processing. It then introduces the Transformer architecture and its key components, before analyzing the differences between MLMs and CLMs. Finally, it explores both traditional and emerging adaptation strategies, highlighting recent advances in generative classification. The objective is to demonstrate how language models have evolved from simple statistical representations into powerful, efficient, and human-aligned systems capable of addressing complex real-world tasks.

Recent advances in Natural Language Processing (NLP) have been largely driven by the emergence of Large Language Models (LLMs), which are capable of learning complex linguistic, semantic, and contextual representations from large-scale textual corpora. These models have significantly transformed the field by enabling substantial improvements across a wide range of applications, including text classification, machine translation, question answering, information retrieval, and conversational systems.

At the core of these advancements lies the Transformer architecture, which introduced the self-attention mechanism (*self-attention*). This mechanism allows models to effectively capture long-range dependencies within textual sequences while enabling efficient parallel computation, thereby overcoming the limitations of earlier sequential models. Building upon this architecture, two dominant paradigms have emerged: Masked Language Models (MLMs), such as *BERT*, which leverage bidirectional context for deep language understanding, and Causal Language Models (CLMs), such as *GPT*, which generate text in an autoregressive manner.

While pretrained language models encode rich linguistic knowledge, they are not inherently optimized for specific downstream tasks or user requirements. This limitation has led to the development of fine-tuning strategies, which adapt pretrained models to particular domains, tasks, and behavioral expectations. Traditional approaches rely on task-specific classification heads; however, these methods suffer from several limitations, including lack of zero-shot capability, architectural rigidity, and limited generative flexibility.

More recently, a new paradigm has emerged that repurposes the native MLM head for generative classification. This approach reformulates tasks as masked token prediction problems and enables models to directly produce answers as single tokens (*Answer Token Prediction*), eliminating the need for additional task-specific heads. Combined with instruction tuning techniques, this strategy

improves parameter efficiency, supports zero-shot generalization, and provides a unified architecture capable of handling multiple tasks.

In parallel, modern LLM training pipelines have evolved into multi-stage processes that include pretraining, supervised fine-tuning (Supervised Fine-Tuning, SFT), and preference-based optimization methods such as Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO). These techniques enhance not only task performance but also alignment with human expectations in terms of usefulness, safety, and clarity.

This chapter provides a comprehensive overview of Transformer-based language models and their adaptation strategies. It presents the fundamental components of the Transformer architecture, examines the differences between MLMs and CLMs, explores both traditional and emerging approaches to model adaptation, and highlights recent advances in generative classification. The objective is to illustrate how language models evolve from general-purpose systems into efficient, task-aware, and human-aligned solutions.

1.2 Artificial Intelligence and Natural Language Processing

1.2.1 Artificial Intelligence

What Is Language AI?

The term artificial intelligence (AI) is often used to describe computer systems dedicated to performing tasks close to human intelligence, such as speech recognition, language translation, and visual perception. It is the intelligence of software as opposed to the intelligence of humans.[Alammar and Grootendorst, 2024]

A Recent History of Language AI

The history of Language AI encompasses many developments and models aiming

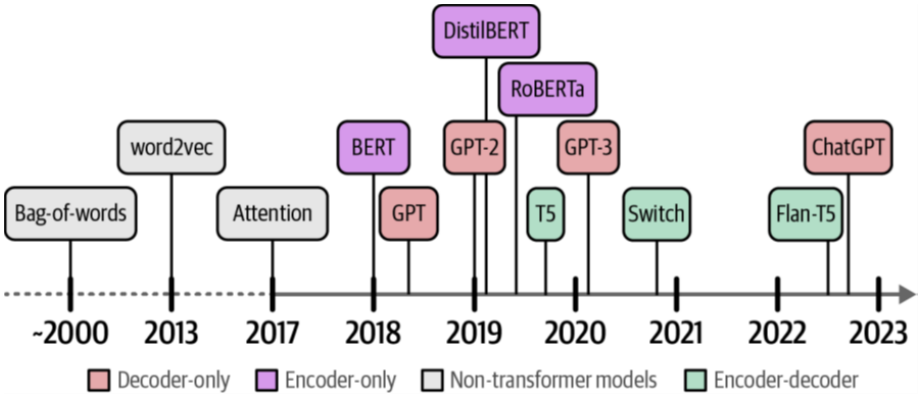

Figure 1 – A peek into the history of Language AI. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

to represent and generate language, as illustrated in Figure 1-1.

Language AI is capable of many tasks by processing textual input. Language AI is capable of performing a wide range of tasks by processing textual input. These tasks include text classification, sentiment analysis, machine trans-lation, question answering, and text generation. Such capabilities demonstrate the ability of AI systems to extract meaning, understand context, and generate coherent responses from human language.

Language, however, is a complex and unstructured form of data for computers. When represented as sequences of characters or numerical values, it may lose its semantic meaning. Therefore, a significant effort in the development of Language AI has focused on transforming text into structured representations that machines can effectively process [Alammar and Grootendorst, 2024].

1.2.2 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a branch of artificial intelligence that aims to enable computers to understand, interpret, and generate human language autonomously and contextually. It is an interdisciplinary field involving **linguistics**, **computer science**, **cognitive science**, **psycholinguistics**, and **logic**, in order to process language, which is a complex and unstructured form of data.

NLP relies on techniques for text representation and statistical or neural modeling, transforming linguistic data into formats that machines can understand. It addresses a wide range of problems, such as:

- **Text classification and analysis** (e.g., sentiment detection)
- **Machine translation**
- **Question answering**
- **Text generation**

In modern language models, NLP serves as the foundation for developing systems that leverage advanced architectures such as **transformers**, **pre-trained language models**, and **multimodal approaches**, ensuring performance that approaches human-level understanding.[Alammar and Grootendorst, 2024]

1.2.3 Challenges in NLP

Natural Language Processing (**NLP**) is a powerful tool in artificial intelligence, but it faces several inherent challenges due to the complexity, ambiguity, and variability of human language. Understanding these challenges is essential for designing effective NLP systems. The main challenges include:

- **Ambiguity:** One of the primary challenges in NLP is ambiguity, which occurs when a word, phrase, or sentence can have multiple interpretations. Ambiguity can be lexical, where a single word has multiple meanings (e.g., the word "*bank*" can refer to a financial institution or the edge of a river), or syntactic, where the structure of a sentence allows multiple

interpretations (e.g., “I saw the man with the telescope”). Resolving ambiguity requires context-aware modeling and sophisticated disambiguation algorithms, which remain a significant hurdle for NLP systems.

— **Contextual Understanding:** Language is highly dependent on context. The meaning of a word or sentence often changes depending on the surrounding text, the speaker’s intention, or even cultural and situational factors. For example, the word “*charge*” can mean “to attack,” “to bill,” or “to energize,” depending on context. Capturing context at mul-tiple levels—sentence, paragraph, or document—is a major challenge for NLP, particularly for tasks such as machine translation, dialogue systems, and summarization. Recent approaches using transformers and attention mechanisms have improved context handling, but limitations remain in understanding long-range dependencies or nuanced contexts.

— **Semantics and Meaning Representation:** Beyond syntax, NLP systems must capture the semantic meaning of words, phrases, and sentences. Human language is often vague, figurative, or implicit, containing idioms, metaphors, and implied meanings that are difficult for machines to interpret. For example, the sentence “He kicked the bucket” literally refers to kicking a bucket but figuratively means “he died.” Developing representations that encode such semantics accurately is a major challenge. Modern techniques like word embeddings (*Word2Vec, GloVe*) and contextual embeddings (*BERT, GPT*) help capture semantic relationships, but fully understanding nuanced meaning remains an open research problem.

— **Variability and Diversity of Language:** Natural language is highly variable across different speakers, regions, and domains. Differences in dialects, writing styles, abbreviations, and even errors (spelling mistakes, typos) make it difficult for NLP systems to generalize. Moreover, multilin-gual NLP adds another layer of complexity, as each language has its own grammar, syntax, and semantic structures. Designing systems that are robust across languages and domains is an ongoing challenge.

— **Knowledge and Commonsense Reasoning:** Many NLP tasks require not just text understanding but also world knowledge or commonsense reasoning. For example, answering a question like “Can a penguin fly?” requires factual knowledge about penguins. Integrating external knowledge sources, such as knowledge graphs or retrieval-augmented models, is crucial but challenging to do efficiently and accurately.

In summary, while NLP has achieved remarkable progress with modern models, these challenges—ambiguity, context dependence, semantic complexity, linguistic variability, and knowledge integration—remain central obstacles. Overcoming them is essential for building systems that can truly understand and interact with human language in a natural and intelligent way.[Alammar and Grootendorst, 2024]

1.3 Deep Learning for NLP

1.3.1 Deep Learning

Deep Learning is a subfield of machine learning that uses **deep neural networks** composed of multiple layers to learn hierarchical representations from raw data. Unlike traditional machine learning, which often requires manually engineered features, Deep Learning enables the automatic learning of **complex representations** that are well-suited to the data and the tasks to be solved.[Alammar and Grootendorst, 2024] Common architectures include:

- **RNN (Recurrent Neural Networks)**: suitable for sequence processing, but limited in capturing long-term dependencies.
- **LSTM and GRU**: RNN variants that better handle long-range dependencies in text.
- **CNN (Convolutional Neural Networks)**: effective for detecting local patterns in sentences, such as frequent phrases or expressions.
- **Transformers**: architectures based on the **attention mechanism**, capable of processing the entire text at once and capturing global context.
- **GNN (Graph Neural Networks)**: neural architectures designed to process graph-structured data by learning node representations through neighborhood aggregation. They are particularly effective for modeling relationships between entities and are widely used in knowledge graphs, recommendation systems, and Retrieval-Augmented Generation (RAG) frameworks.

1.3.2 Why Deep Learning in NLP

The history of Language AI begins with simple word representation techniques such as bag-of-words, a method for converting unstructured text into structured data. Bag-of-words was first introduced around the 1950s and gained widespread use in the 2000s. Later, this approach was extended to n-grams, which capture sequences of words to retain some contextual information.[Alammar and Grootendorst, 2024]

1.3.3 Bag-of-Words Representation

The Bag-of-Words (BoW) model is a simple technique in Natural Language Processing (NLP) used to convert text into numerical data that a machine can understand.[Alammar and Grootendorst, 2024]

Bag-of-words works as follows:

1 Tokenization

Tokenization is the process of splitting a text into small units called tokens (usually words).

Example:
Original sentence: ”I love machine learning”
After tokenization: [”I”, ”love”, ”machine”, ”learning”]

Notes:
- Tokens can be words, characters, or sentences.
- Text is often cleaned (lowercase, remove punctuation, etc.).

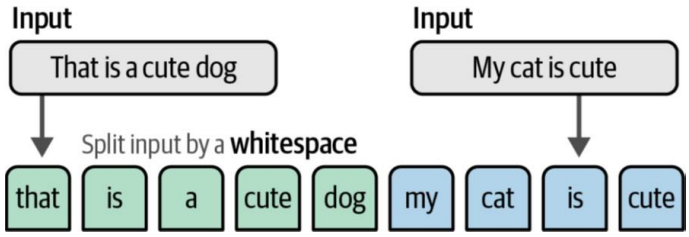

Figure 2 – Tokenization example. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

2 Build the Vocabulary

Create a list of all unique words that appear in the dataset.

Example:
Sentences:
”I love machine learning”
”I love AI”

Vocabulary: [”I”, ”love”, ”machine”, ”learning”, ”AI”]

Notes:
- Each word appears only once.
- The order of words must remain fixed.

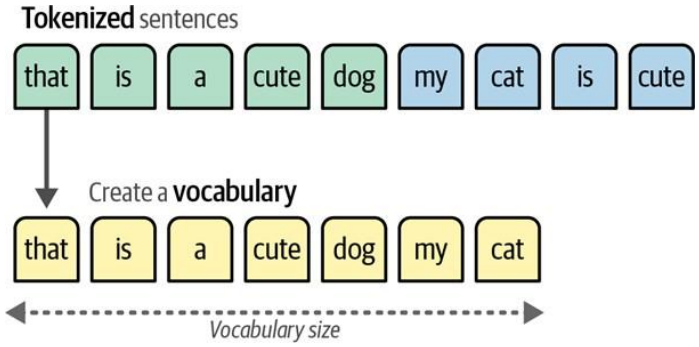

Figure 3 – Vocabulary construction. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

3 Counting

Count how many times each word from the vocabulary appears in the text.

Example:
Vocabulary: [”I”, ”love”, ”machine”, ”learning”, ”AI”]
Sentence: ”I love AI”

Counting:
”I” = 1, ”love” = 1, ”machine” = 0, ”learning” = 0, ”AI” = 1

Result: [1, 1, 0, 0, 1]

4 Vector of Numbers

Convert the counts into a numerical vector representation of the text.

Example:
Vocabulary: [”I”, ”love”, ”machine”, ”learning”, ”AI”]

Sentence 1: ”I love machine learning”
Vector: [1, 1, 1, 1, 0]

Sentence 2: ”I love AI”
Vector: [1, 1, 0, 0, 1]

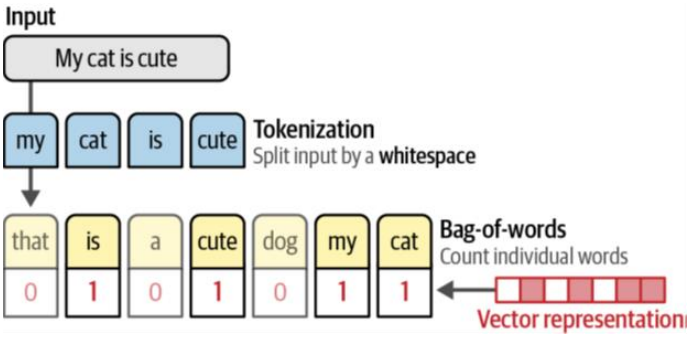

Figure 4 – Vector representation. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

One-Hot Encoding is a technique used to represent individual words as numerical vectors. Each word in the vocabulary is associated with a unique vector where all elements are zero except one element set to one, corresponding to the index of the word in the vocabulary.

For example, given a vocabulary "I", "love", "AI":

"I" → [1, 0, 0] "love" → [0, 1, 0] "AI" → [0, 0, 1]

This representation is simple and allows words to be processed by machine learning models. However, it has important limitations: It

produces high-dimensional sparse vectors.

It does not capture semantic relationships between words.

Each word is treated as independent.

Although Bag-of-Words is a simple and effective representation method, it ignores the semantic meaning of words and their context. To address this limitation, **Word2Vec** was introduced in 2013 as one of the first approaches to learn meaningful word embeddings from large text corpora.[Alammar and Grootendorst, 2024]

1.3.4 Deep Learning in NLP

Deep Learning has significantly transformed Natural Language Processing (NLP) by enabling models to learn meaningful representations of text directly from large-scale data. Unlike traditional approaches such as Bag-of-Words or One-Hot Encoding, which represent words as sparse and independent vectors, Deep Learning methods aim to capture the semantic relationships between words.

One of the earliest and most influential approaches in this direction is Word2Vec, introduced in 2013. Word2Vec leverages neural networks, which consist of interconnected layers of nodes that process information. As illustrated in Figure 1-6, neural networks can have multiple layers, where each connection has a weight depending on the input. These weights are often referred to as the parameters of the model.[Alammar and Grootendorst, 2024]

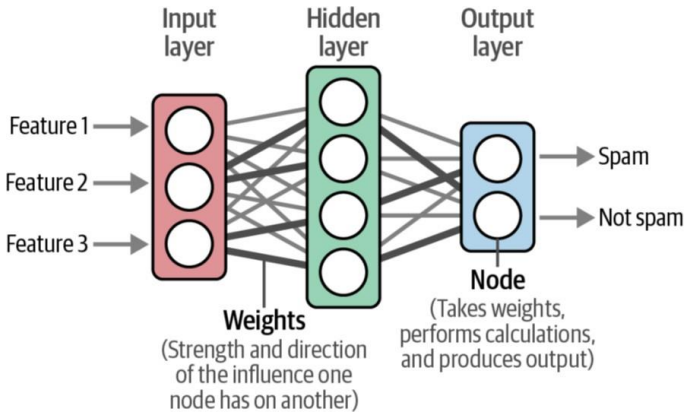

Figure 5 – Enter Caption. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

Figure 1-6. A neural network consists of interconnected layers of nodes where each connection is a linear equation. Using these neural networks, Word2Vec generates word embeddings by analyzing which other words tend to appear next to a given word in a sentence. Initially, each word in the vocabulary is assigned a random vector (e.g., 50 dimensions). During training, as illustrated in Figure 1-7, the model takes pairs of words from the training data and predicts whether they are likely to be neighbors in a sentence. The embeddings are up-dated iteratively to reflect these predictions and align with the observed contexts.

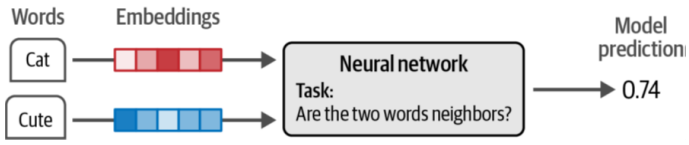

Figure 6 – Enter Caption. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

The resulting embeddings capture the meaning of words. For example, consider the words “apple” and “baby”. Embeddings represent the properties of words: ”baby” might score high on properties like “newborn” and “human”, while ”apple” scores low on these properties. As illustrated in Figure 1-8, em-beddings can have multiple properties to represent the semantic aspects of a word. Since the size of embeddings is fixed, these properties are carefully chosen to provide a mental representation of the word.

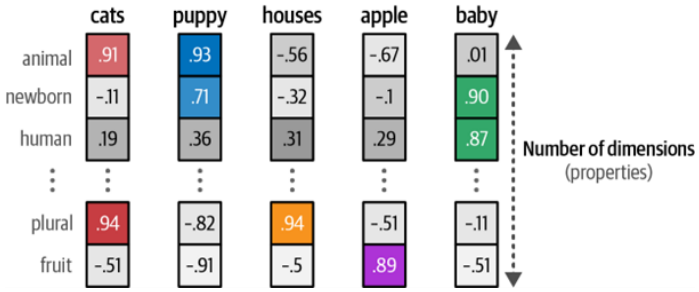

Figure 7 – Enter Caption.Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

This advancement marked a major shift in NLP, moving from simple statistical representations to learned semantic representations. It also laid the foundation for more advanced Deep Learning models, such as GloVe, BERT, and GPT, which further enhance contextual understanding and language generation.

1.4 Word Embeddings

Embeddings are numerical vector representations of words (or sentences) that capture their meaning. Instead of treating words as just symbols, embeddings represent semantic properties such as context and similarity.[Alammar and Grootendorst, 2024]

1.4.1 How are embeddings created?

1. Collect a large text dataset
2. Tokenization
3. Initialize vectors randomly
4. Train using context
5. Neural network optimization: The network adjusts vector values during training to minimize prediction errors.

1.4.2 Why are embeddings important?

— They capture semantic meaning

— They allow models to compare words mathematically

— They are the foundation of modern NLP models (Word2Vec, GloVe, BERT, GPT)

1.4.3 Types of Embeddings

[Alammar and Grootendorst, 2024]

1. **Word Embeddings**

— Represent individual words as numerical vectors

— Capture semantic meaning and relationships between words

— Example models: Word2Vec, GloVe

Example:
”king” → [0.25, 0.80, 0.10]
”queen” → [0.27, 0.78, 0.12]

Interesting relation:
king - man + woman ≈ queen

Applications:

— Machine translation

— Text classification

— Semantic search

Limitation:
A word has only one vector representation, meaning it cannot handle context (e.g., ”bank” as a financial institution vs river bank).

2. **Sentence Embeddings**

— Represent an entire sentence as a single vector

— Capture the global meaning of a sentence

— Used to compare sentence similarity

— Example model: Sentence-BERT

Example:
Sentence 1: ”I love machine learning”
Sentence 2: ”I enjoy studying AI”

These sentences have similar embeddings because they share similar meanings.

Applications:

— Semantic search

— Chatbots

— Text similarity comparison

Advantages:

— Captures full sentence meaning

— More accurate than word embeddings

3. **Document Embeddings**

— Represent full documents or long texts as a single vector

— Capture overall topic or content of a document

Simple method: Bag-of-Words

Document: ”cat eats fish”
Vocabulary: [”cat”, ”eats”, ”fish”, ”dog”]
Vector: [1, 1, 1, 0]

Limitation:

— Does not capture meaning

— Ignores word order

Advanced methods:

— Doc2Vec

— BERT

These models capture context and better understand the global meaning of a document.

Applications:

— Document classification

— Text summarization

— Information retrieval

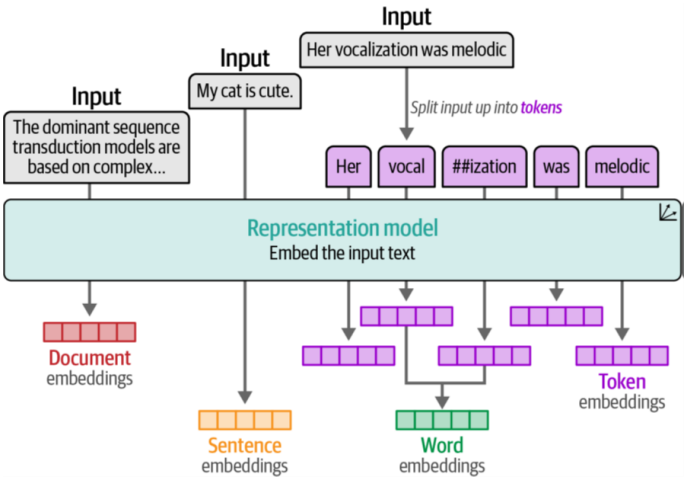

Figure 8 – Embeddings can be created for different types of input. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

1.4.4 Embeddings for Recommendation Systems

Embeddings allow a system to understand relationships between items, not just labels. In recommendation systems:

— Items that frequently appear together are considered similar

— This similarity can be learned automatically from user behavior

1.4.5 How does this work for songs?

The key idea is:

— Song = word

— Playlist = sentence

Using the Word2Vec algorithm:

1. The model scans many playlists
2. It learns which songs often appear together
3. Each song is converted into a vector (embedding)
4. Songs that co-occur in playlists get similar vectors

[Alammar and Grootendorst, 2024]

1.4.6 What does the training actually learn?

The model does not understand music itself. It learns patterns from data:

— If two songs appear in many of the same playlists, their embeddings become similar

— If they rarely appear together, their embeddings are far apart

This captures musical taste and genre indirectly.

1.4.7 How are recommendations generated?

To recommend songs:

1. Take the embedding of a given song
2. Compute similarity (usually cosine similarity) with all other songs
3. Return the closest vectors

These closest vectors correspond to songs with:

— Similar genre

— Similar audience

— Similar listening context

The concept of embeddings demonstrates how deep learning models can capture semantic relationships, whether between words in a text or items in a recommendation system. Before we can generate these embeddings effectively, however, the raw text or sequence data must be properly processed. This brings us to **tokenization**, the essential first step in preparing textual data for language models, which will be discussed in the next section.[Alammar and Grootendorst, 2024]

1.5 Tokenization Techniques

1.5.1 Tokenization

Tokenization is the process of breaking text into smaller pieces called **tokens**.

1.5.2 Tokenizer Properties: How Does a Tokenizer Break Down Text?

A tokenizer decides how to split text based on three main factors: the method, design choices, and the training data.

1.5.3 Tokenization Method

The tokenization method is chosen when the model is designed. Common methods include[Alammar and Grootendorst, 2024]:

1. Byte Pair Encoding (BPE) – used by GPT models

- Starts with characters
- Repeatedly merges the most frequent character pairs
- Produces tokens that can be full words or parts of words

Example: playing → play + ing

BPE efficiently handles new or rare words by merging frequent subword units.

2. WordPiece – used by BERT

- Similar to BPE but chooses splits that reduce ambiguity
- Considers the frequency of word pieces in training data

Example: unhappiness → un + happy + ness

WordPiece ensures that splits maximize probability and make sense in context.

1.5.4 Tokenizer Design Choices

After choosing the method, several design decisions are made:

Vocabulary Size

- Smaller vocabulary → more splitting (e.g., apolog + izing)
- Larger vocabulary → fewer splits (e.g., apologizing)

Special Tokens

— Tokens like ¡s¿ (start), ¡/s¿ (end), ¡pad¿ (padding), ¡—assistant—¿ (role)

— Help structure how the model reads and generates text

1.5.5 Training Data

The tokenizer is trained on a dataset, which strongly affects how it splits text:

— English text → optimized for natural language

— Code → optimized for programming tokens

— Multilingual data → different token patterns

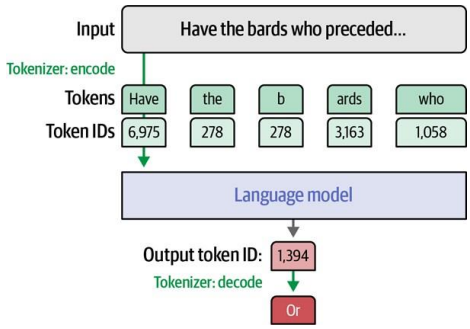

Figure 9 – Tokenizers are also used to process the output of the model by converting the output token ID into the word or token associated with that ID. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

1.5.6 1.5.5 Tokenization Types

Tokenization is the process of converting raw text into smaller units called tokens. Different tokenization strategies exist depending on the model design, efficiency, and language coverage.[Alammar and Grootendorst, 2024]

1. **Word Tokens**

— Each token corresponds to a complete word.

— It is the most intuitive and simple form of tokenization.

— The vocabulary consists of all unique words in the dataset.

Example:
Sentence: ”I love machine learning”
Tokens: [”I”, ”love”, ”machine”, ”learning”]

Advantages:

— Easy to understand and implement

— Preserves word-level meaning

Limitations:

— Large vocabulary size

— Cannot handle unseen or rare words (Out-of-Vocabulary problem)

— Poor handling of morphological variations (e.g., run, running, ran)

2. **Subword Tokens**

— Words are split into smaller meaningful units (subwords).

— A token can be a full word or part of a word.

— This is the most widely used method in modern Large Language Models (LLMs).

Example:
Word: ”unhappiness”
Tokens: [”un”, ”happi”, ”ness”]

Used in models such as:

— BERT

— GPT models

— LLaMA

Advantages:

— Handles unknown words efficiently

— Reduces vocabulary size

— Captures morphological structure of words

Limitations:

— Slightly more complex than word-level tokenization

3. **Character Tokens**

— Each token corresponds to a single character.

— The text is broken down into letters, numbers, and symbols.

Example:
Word: ”AI”
Tokens: [”A”, ”I”]

Advantages:

— Very flexible and robust to unknown words

— Small and fixed alphabet size

— Works well for noisy or misspelled text

Limitations:

— Very long sequences

— High computational cost during training

— Harder to capture semantic meaning

4. **Byte Tokens**

— Each token represents a byte (Unicode encoding level).

— Can represent any text, including all languages, symbols, and emojis.

— Works at the lowest level of text representation.

Example:
Word: ”Hi”
Tokens (UTF-8 bytes): [72, 105]

Special symbols such as smiley faces, graphical icons, and expressive char-acters are also encoded using UTF-8 bytes. This allows byte tokenization to handle them correctly without requiring any special preprocessing.

Advantages:

— Fully universal (supports all languages and special characters)

— No Out-of-Vocabulary problem

— Very robust to any input format, including emojis and noisy text

Limitations:

— Very long sequences

— Higher computational cost

— Less semantic structure compared to subword methods

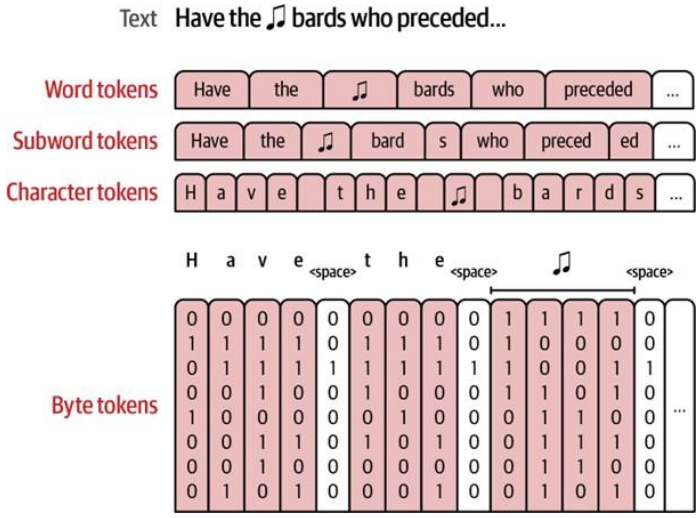

Figure 10 – There are multiple methods of tokenization that break down the text to different sizes of components (words, subwords, characters, and bytes. Source: Alammar & Grootendorst (2024) [Alammar and Grootendorst, 2024].

1.5.7 Token Embeddings

What are Embeddings? Embeddings are numerical vectors representing tokens (words or parts of words). They allow a model to understand semantic meaning and relationships between tokens.[Alammar and Grootendorst, 2024]

Why they matter:

— Models cannot process text directly—they work with numbers.

— Embeddings preserve meaning in numeric form.

— Enable text understanding, generation, and reasoning.

1.5.8 Contextualized Word Embeddings with Language Models

1. **Tokenization:** Split sentences into tokens.
2. **Token Embeddings:** Convert each token into a numeric vector.
3. **Language Model:** Analyze all tokens together.
4. **Contextualized Embeddings:** Word meaning changes based on context.

Example:

— river bank → side of a river

— money bank → financial institution

1.6 Word Embeddings Beyond LLMs

— Useful in recommendation systems, semantic search, and robotics

— Word2Vec converts words into embeddings based on context

— Embeddings can represent relationships:

  — king ∼ queen, coffee ∼ tea, coffee far from airplane

1.6.1 Word2Vec Algorithm and Training

1. Reads large text corpora
2. Picks a word and examines nearby words
3. Adjusts vectors so similar words are close together and unrelated words are far apart
4. Result: each word has a vector representing its meaning

1.7 Transformers

Before the introduction of the Transformer, the dominant models for sequence processing were Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks. These models generate a sequence of hidden states h_t, where each state depends on the previous hidden state h_{t-1} and the input at position t. This strictly sequential computation prevents parallelization, which becomes a major limitation when dealing with long sequences [Vaswani et al., 2017].

Recurrent models such as LSTMs and GRUs have long been established as state-of-the-art approaches for sequence modeling and transduction tasks such as language modeling and machine translation. However, a fundamental limitation remains: the sequential dependency between hidden states prevents efficient parallel computation during training, making long sequences computationally expensive [Vaswani et al., 2017].

1.7.1 Attention Mechanisms and Previous Limitations

Attention mechanisms allow models to capture dependencies between positions regardless of their distance in the sequence. However, in most existing architectures, attention is used in combination with recurrent networks [Vaswani et al., 2017].

Several models, such as the Extended Neural GPU, ByteNet, and ConvS2S, attempted to reduce sequential computation using convolutional architectures. While these models enable parallel computation, the number of operations required to connect distant positions increases with distance, making long-range dependency learning more difficult [Vaswani et al., 2017].

1.7.2 The Transformer Model

In this work, we propose the Transformer, a neural network architecture based entirely on attention mechanisms. It removes recurrence and convolution entirely and relies solely on attention to model global dependencies between input and output sequences [Vaswani et al., 2017].

The Transformer enables significantly higher parallelization and achieves superior translation performance while requiring substantially less training time compared to previous models [Vaswani et al., 2017].

On the WMT 2014 English-to-German translation task, the model achieves a BLEU score of 28.4, improving over previous best results by more than 2 BLEU points. On the WMT 2014 English-to-French task, it achieves a state-of-the-art BLEU score of 41.0 after 3.5 days of training on eight GPUs [Vaswani et al., 2017].

1.7.3 Overall Architecture

The Transformer follows an encoder–decoder architecture. The encoder maps an input sequence into continuous representations, while the decoder generates the output sequence in an auto-regressive manner, producing one token at a time [Vaswani et al., 2017].

The architecture is composed entirely of stacked self-attention layers and position-wise feed-forward networks, without recurrence or convolution.

1.7.4 Encoder and Decoder Stacks

Encoder The encoder consists of a stack of six identical layers. Each layer contains two sub-layers: a multi-head self-attention mechanism and a position-wise feed-forward network [Vaswani et al., 2017].

Each sub-layer is wrapped with a residual connection followed by layer normalization [Vaswani et al., 2017]:

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

All layers produce representations of dimension $d_{model} = 512$.

Decoder The decoder is also composed of six identical layers, each containing three sub-layers [Vaswani et al., 2017]:

- Masked multi-head self-attention
- Encoder–decoder attention
- Position-wise feed-forward network

The first sub-layer prevents attending to future positions, ensuring the auto-regressive property.

1.7.5 Attention Mechanism

The attention mechanism maps a query and a set of key–value pairs to an output computed as a weighted sum of the values [Vaswani et al., 2017].

1.7.6 Scaled Dot-Product Attention

The Transformer uses scaled dot-product attention:

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

This scaling stabilizes gradients by preventing large dot-product values from saturating the softmax function [Vaswani et al., 2017].

1.7.7 Multi-Head Attention

Instead of a single attention function, queries, keys, and values are projected into multiple subspaces and attention is performed in parallel [Vaswani et al., 2017]:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

This allows the model to jointly attend to information from different representation subspaces.

1.7.8 Types of Transformers

- Encoder-based (BERT, RoBERTa): bidirectional understanding models
- Decoder-based (GPT, GPT-3): autoregressive generation models
- Encoder–decoder (T5, BART): translation and summarization
- Vision Transformers (ViT)
- Multimodal models (CLIP, DALL-E)
- Efficient variants (Longformer, Sparse Transformers)

1.7.9 Applications

- Natural Language Processing: machine translation, text generation
- Computer Vision: image classification (ViT)
- Biology: protein folding (AlphaFold)
- Multimodal generation (DALL-E)

1.7.10 Advantages and Limitations

Advantages

— High parallelization

— Strong performance on sequence tasks

— Flexible architecture

Limitations

— High computational cost

— Requires large datasets

1.7.11 Conclusion

Transformers represent a major advancement in AI. They outperform sequential models, thanks to the attention mechanism, and have revolutionized NLP, vision, and multimodal applications. Ongoing research focuses on reducing complexity, improving efficiency, and expanding use in fields like healthcare, science, and intelligent systems [Vaswani et al., 2017].

1.8 Masked Language Models (MLMs) – BERT and Beyond

1.8.1 Introduction to Masked Language Models

Masked Language Models (MLMs) represent a fundamental paradigm in self-supervised pretraining for Natural Language Processing. Introduced prominently with BERT (Bidirectional Encoder Representations from Transformers) by Devlin et al. (2019), MLMs are encoder-only Transformer architectures designed to learn deep bidirectional representations by predicting randomly masked tokens in input sequences.

Core Principle: During pretraining, a proportion of input tokens (typically 15–40%) is replaced with a special [MASK] token. The model's objective is to reconstruct the original tokens based on their surrounding context in a single forward pass. This denoising objective forces the model to develop rich contextual understanding, as each prediction depends on both left and right context—unlike autoregressive (decoder-only) models that process text sequentially.[Alammar and Grootendorst, 2024]

1.8.2 Architectural Foundations

Masked Language Models (MLMs) such as BERT are built upon the Transformer Encoder architecture. Unlike traditional sequential models (e.g., RNNs or LSTMs), the Transformer processes all tokens in parallel, leading to improved efficiency and better modeling of long-range dependencies.[Alammar and Grootendorst, 2024]

1. Transformer Encoder Structure A Transformer Encoder consists of a stack of identical layers. Each layer is composed of several key components that work together to encode contextual information.

2. Self-Attention Mechanism The self-attention mechanism is the core component of the Transformer.

Main idea: Each token attends to all other tokens in the sequence to build a contextual representation.

How it works: For each token, three vectors are computed:

— Query (Q)

— Key (K)

— Value (V)

The attention operation is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

Intuition:

— Compute similarity between tokens (Q vs K)

— Assign importance weights using softmax

— Aggregate information using V

Example: In the sentence "The bank is near the river", the word *bank* attends strongly to *river*, helping the model infer the correct meaning.

Multi-Head Attention:

— Multiple attention heads are used instead of a single one

— Each head captures different relationships (syntax, semantics, etc.)

— Outputs are concatenated and projected

3. Positional Encodings Since the Transformer has no inherent notion of word order, positional encodings are added to input embeddings.

Purpose: To inject sequence order information into the model.

Types:

— Sinusoidal (fixed):

$$PE(pos, i) = \sin\left(\frac{pos}{10000^{2i/d}}\right) \quad (2)$$

— Learned positional embeddings

Importance: Allows the model to distinguish between sentences such as "dog bites man" and "man bites dog".

4. Layer Normalization and Residual Connections These components ensure stable and efficient training of deep networks.

Residual Connections:

$$\text{Output} = \text{Layer}(x) + x \tag{3}$$

— Prevent vanishing gradients

— Facilitate training of deep architectures

Layer Normalization:

— Normalizes activations (mean = 0, variance = 1)

— Stabilizes training

— Speeds up convergence

5. Feed-Forward Networks (FFN) After self-attention, each token is passed through a fully connected feed-forward network.

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{4}$$

Characteristics:

— Applied independently to each token

— Introduces non-linearity

— Increases model capacity

6. Transformer Layer Pipeline Each encoder layer follows this sequence:

1. Self-Attention
2. Add & Layer Normalization
3. Feed-Forward Network
4. Add & Layer Normalization

This block is repeated multiple times (e.g., 12 layers in BERT-base).

7. Why This Architecture is Powerful

— Captures long-range dependencies

— Builds deep contextual understanding

— Enables parallel computation

— Scales effectively to large models

8. Summary

Component	Role
Self-Attention	Captures global context
Positional Encoding	Injects word order
Residual + LayerNorm	Stabilizes training
Feed-Forward Network	Applies non-linear transformations

Key Components of BERT-style MLMs:

Component	**Function**
Token Embeddings	Map discrete tokens to a continuous vector space.
Segment Embeddings	Distinguish sentence pairs (e.g., for NLI tasks).
Positional Embeddings	Encode the position of each token within the sequence.
MLM Head	Linear projection followed by softmax to predict masked tokens.
Next Sentence Prediction (NSP)	Binary classification task for sentence-pair coherence (used in BERT pretraining).

1.8.3 Traditional Downstream Adaptation: Classification Heads

Standard Fine-tuning Pipeline: For tasks like text classification, senti-ment analysis, or named entity recognition, MLMs are typically adapted via [Alammar and Grootendorst, 2024]:

1. Input Preparation: Tokenize and format input with special tokens ([CLS], [SEP]).
2. Forward Pass: Obtain contextualized representations from the final encoder layer.
3. Pooling: Extract a fixed-size representation (commonly the [CLS] token).
4. Classification Head: Apply a task-specific linear layer to map pooled representations to label probabilities.

Limitations of Classification-Head Approaches:

— Zero-shot incapability: Requires labeled data and task-specific fine-tuning.

— Prompt sensitivity: Performance varies with input formatting.

— Architectural rigidity: Each new task demands a new head and retraining.

— Limited generative flexibility: Cannot produce open-ended outputs or explanations.

1.8.4 Emerging Paradigm: Generative Use of the MLM Head

Recent research (e.g., Clavi´e et al., 2025) explores repurposing the native MLM head for generative classification, eliminating the need for task-specific heads.

Core Idea: Answer Token Prediction Instead of masking random tokens, the training objective is modified to:

- Format tasks as Cloze-style questions with a single [MASK] representing the answer.
- Train the model to predict a single verbalizer token (e.g., “Positive”, “A”, “B”) corresponding to the target label.

Training Strategy: Masked Instruction Tuning

- Data Source: Filtered subsets of instruction datasets (e.g., FLAN) containing single-token answers.
- Objective Mix: 80% Answer Token Prediction (ATP), 20% “Dummy MLM” (standard random masking with placeholder labels for regularization).
- Templating: Minimal, reusable templates for training and inference to ensure generalization.

Inference Mechanism: At test time:

1. The model receives a formatted prompt with [MASK] at the answer position.
2. Performs a single forward pass.
3. Outputs probability distribution over the vocabulary at the masked position.
4. Maps the highest-probability token(s) to the target label space.

1.8.5 Advantages of MLM-Based Generative Classification

Advantage	Explanation
Zero-shot capability	Leverages pretrained knowledge without task-specific fine-tuning
Parameter efficiency	No additional classification head; uses existing MLM head
Unified architecture	Single model handles multiple tasks via prompting
Interpretability	Output probabilities directly reflect model confidence per token
Computational efficiency	Single forward pass vs. autoregressive generation requiring multiple steps

1.8.6 Comparison with Decoder-Only LLMs

Aspect	Encoder-Only MLMs	Decoder-Only LLMs
Context Access	Bidirectional (full context)	Unidirectional (causal)
Inference Speed	Single forward pass	Autoregressive (sequential)
Zero-shot Flexibility	Emerging via MLM head	Native via instruction tuning
Parameter Efficiency	High performance at ¡0.5B params	Often requires ¿1B params for comparable
Task Adaptation	Prompt-based or fine-tuned head	Prompt-based or LoRA/PEFT fine-tuning
Best Use Cases	Classification, NLU, retrieval	Generation, reasoning, dialogue

1.8.7 Conclusion and Future Directions

Masked Language Models, pioneered by BERT, have evolved from context encoders with task-specific heads into versatile generative classifiers via instruction tuning of their native MLM head. This paradigm shift offers[Alammar and Grootendorst, 2024]:

— Unified architecture: One model, multiple tasks, no architectural modifications.

— Efficiency: Competitive performance at a fraction of the parameters of decoder-only LLMs.

— Accessibility: Enables zero-shot capabilities for resource-constrained applications.

Future work should explore:

— Extending to multi-token answer generation via iterative masking or beam search.

— Systematic studies on verbalizer design and prompt optimization.

— Integration with retrieval-augmented generation (RAG) for knowledge-intensive tasks.

— Application to multimodal and structured data domains.

1.9 Causal Language Models (e.g., GPT)

Causal Language Models (CLMs), also known as autoregressive language models, constitute a fundamental paradigm in Natural Language Processing. In contrast to Masked Language Models (MLMs), which leverage bidirectional context, CLMs model the probability of a token based solely on preceding tokens [Alammar and Grootendorst, 2024].

1.9.1 Core Principle

The training objective of a causal language model is to maximize the conditional probability:

$$P(x_t \mid x_1, x_2, \ldots, x_{t-1}) \tag{5}$$

This formulation implies that text generation is inherently sequential, where each token depends only on previously generated tokens. This behavior is enforced through a **causal mask** applied within the self-attention mechanism, preventing access to future tokens.

1.9.2 Architecture

Causal language models such as GPT are based on the Transformer decoder architecture, which includes:

— Masked self-attention (causal masking)

— Positional encodings

— Feed-forward neural networks

— Residual connections and layer normalization

Unlike encoder-based models (e.g., BERT), GPT processes sequences from left to right, making it particularly suitable for generative tasks.

1.9.3 Pretraining Objective

During pretraining, the model is trained on large-scale corpora using the next-token prediction objective, typically optimized with cross-entropy loss.

This enables the model to acquire:

— Syntactic knowledge

— Semantic understanding

— Contextual coherence

— Implicit world knowledge

1.9.4 Applications

Causal language models are widely used in:

— Text generation

— Conversational systems

— Story completion

— Code generation

— Generative question answering

1.9.5 Limitations

— Sequential generation leads to slower inference

— Lack of bidirectional context

— High computational cost for long sequences

1.10 LLMs Fine-Tuning

Large Language Models (LLMs) are initially trained on large-scale unlabeled corpora to learn general linguistic representations. However, this pretraining stage alone is insufficient for directly deploying models in real-world applications. Fine-tuning is therefore introduced as a crucial step that adapts pretrained models to specific downstream tasks, domains, or behavioral requirements.

Fine-tuning transforms a general-purpose language model into a **task-aware and instruction-following system** by adjusting its parameters using task-specific data. This process is a form of transfer learning, where knowledge acquired during pretraining is reused and refined.[Alammar and Grootendorst, 2024]

1.10.1 From Pretraining to Adaptation

Pretrained language models encode rich syntactic, semantic, and contextual knowledge learned from large corpora. Despite this capability, they are not inherently optimized for specific applications such as text classification, summarization, question answering, or dialogue generation.

The main limitation of pretrained models lies in their lack of **task specialization and instruction alignment**. Consequently, they may produce outputs that are fluent but not necessarily relevant, structured, or aligned with user intent.

Fine-tuning addresses this limitation by adapting the pretrained parameters to a supervised dataset relevant to a target task. This process allows the model to specialize while preserving general linguistic knowledge acquired during pretraining.[Alammar and Grootendorst, 2024]

1.10.2 Supervised Fine-Tuning (SFT)

Supervised Fine-Tuning (SFT) is the most widely used approach for adapting pretrained language models. It relies on a dataset composed of **paired input-output examples**, where each input corresponds to a prompt or instruction, and each output represents the expected response.

Although the underlying training objective remains next-token prediction, the optimization process is guided by labeled examples, enabling the model to learn structured and task-specific behaviors.

Formally, the training objective is based on the **cross-entropy loss function**, defined as:

$$L = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}, x) \tag{6}$$

where x represents the input sequence and y_t denotes the target token at time step t. The model parameters are optimized using backpropagation and gradient-based optimization methods.

Through this process, the model learns to maximize the likelihood of generating correct output sequences conditioned on given inputs.

SFT can be implemented in two main forms[Alammar and Grootendorst, 2024]:

— **Full fine-tuning**, where all model parameters are updated

— **Parameter-efficient fine-tuning (PEFT)**, where only a subset of parameters is trained (e.g., LoRA, adapters, prompt tuning)

1.10.3 Effects of Fine-Tuning

Fine-tuning significantly enhances the functional capabilities of pretrained language models. After this stage, the model exhibits improved behavior in several aspects.

First, the model becomes **instruction-following**, meaning it is able to interpret and execute natural language instructions more effectively. Second,

it becomes **task-aware**, producing outputs that are better aligned with the expected structure of specific tasks such as classification, summarization, or question answering.

In addition, fine-tuning improves performance in **domain-specific scenarios**, where specialized knowledge or terminology is required. This includes fields such as medicine, law, and finance.

Finally, fine-tuning enhances the overall **accuracy, relevance, and consis-tency** of generated outputs, although it does not fully guarantee alignment with human preferences or safety requirements[Alammar and Grootendorst, 2024].

Summary

In summary, fine-tuning plays a fundamental role in bridging the gap between general language modeling and practical applications. By adapting pretrained representations to task-specific datasets, it enables Large Language Models to evolve from general linguistic systems into specialized, instruction-following models suitable for real-world deployment.

1.10.4 Advanced LLM Fine-Tuning (Modern Pipeline)

Modern Large Language Models (LLMs) go beyond traditional training approaches by adopting a multi-stage pipeline. This pipeline is designed to improve not only performance, but also alignment with human values, safety, and response quality[Alammar and Grootendorst, 2024].

Pretraining

Pretraining is the foundational stage where the model learns general language representations from massive unlabeled corpora (e.g., books, articles, web data).

Objective: The model is trained using next-token prediction, i.e., predicting the next word given previous context.

$$L = -\sum_{t} \log P(x_t \mid x_1, x_2, \ldots, x_{t-1}) \tag{7}$$

Key characteristics:

— Self-supervised learning (no manual labels required)

— Learns grammar, syntax, semantics, and world knowledge

— Produces a general-purpose language model

Limitation:

— Outputs may be correct but not helpful or aligned with user intent

— No guarantee of safety or instruction-following behavior

Example: Given the prompt: “The capital of France is”, the model predicts “Paris” based on statistical patterns.

1.10.2 Supervised Fine-Tuning (SFT) In this stage, the pretrained model is fine-tuned on high-quality labeled datasets consisting of instruction-response pairs.

Objective: Teach the model how to follow instructions and produce structured, useful responses.

Training data format:

— Input: Instruction or question

— Output: Desired response written by humans

Example:

— Instruction: "Explain photosynthesis"

— Response: A clear, structured explanation

Benefits:

— Improves coherence and readability

— Enables instruction-following behavior

— Reduces irrelevant or random outputs

Limitation:

— Learns to imitate data, not necessarily human preferences

— May still produce unsafe or suboptimal responses

1.10.3 Preference Tuning Preference tuning aligns the model with human expectations by optimizing not just correctness, but also usefulness, safety, and quality.

Reinforcement Learning from Human Feedback (RLHF) RLHF introduces a reward-based optimization process.

Pipeline:

1. Generate multiple responses for a prompt
2. Humans rank responses (best to worst)
3. Train a reward model to predict human preferences
4. Optimize the LLM using reinforcement learning (e.g., PPO)

Key components:

— Reward Model (RM): Scores outputs

— Policy Model: The LLM being optimized

— Optimization Algorithm: Proximal Policy Optimization (PPO)

Advantages:

— Strong alignment with human values

— Improves helpfulness and safety

Challenges:

— Complex and computationally expensive

— Requires human annotation

Direct Preference Optimization (DPO) DPO is a simpler alternative to RLHF that directly optimizes preference comparisons.

Principle: Instead of training a reward model, DPO directly learns from pairs of responses:

- Preferred response
- Rejected response

Objective:

$$L_{DPO} = -\log \sigma(\beta(\log P(y_{chosen}) - \log P(y_{rejected}))) \quad (8)$$

Advantages:

- No need for a separate reward model
- More stable training
- Simpler implementation

Limitation:

- Still depends on high-quality preference data

1.10.4 Importance of Preference Tuning Preference tuning is critical to transform a raw language model into a usable assistant.

It enables the model to learn:

- Helpfulness (useful and relevant answers)
- Safety (avoiding harmful or toxic outputs)
- Clarity (well-structured explanations)
- Style (polite, professional, or conversational tone)

Example: Two answers may be correct, but preference tuning selects the one that is:

- clearer
- safer
- more user-friendly

1.10.5 Key Insight The modern LLM training pipeline can be summarized as a progressive refinement process:

- Pretraining: learning general language representations
- Supervised Fine-Tuning: learning task-specific and instruction-following behavior
- Preference Tuning: learning human-aligned responses

Key insight: This multi-stage process transforms language models from simple probabilistic predictors into intelligent, interactive, and human-aligned systems capable of real-world applications.

1.11 Conclusion

In this chapter, we have presented a comprehensive overview of the foundations and evolution of language models in Natural Language Processing (NLP). We began by tracing the progression from traditional text representation meth-ods such as Bag-of-Words and n-gram models to early neural architectures, including RNNs, LSTMs, and CNNs, which enabled more effective sequential and contextual modeling of language.

We then introduced Large Language Models (LLMs), which represent a major milestone in NLP due to their ability to learn rich linguistic, semantic, and contextual representations from large-scale corpora. Central to their success is the Transformer architecture, which relies on the self-attention mechanism to efficiently capture long-range dependencies while enabling parallel computation. This innovation has established Transformers as the dominant framework for modern language understanding and generation.

Furthermore, we examined the two main paradigms of language modeling: Masked Language Models (MLMs), such as BERT, which exploit bidirectional context for understanding tasks, and Causal Language Models (CLMs), such as GPT, which generate text in an autoregressive manner. We also discussed their respective strengths, limitations, and roles in different NLP applications.

In addition, this chapter highlighted the importance of model adaptation through fine-tuning strategies. While traditional approaches rely on task-specific classification heads, recent advances have introduced more unified and efficient paradigms, including generative use of the MLM head and instruction tuning. These approaches improve parameter efficiency, enable zero-shot generalization, and reduce architectural complexity.

Finally, we explored modern training pipelines for LLMs, which combine pretraining, supervised fine-tuning (SFT), and preference-based optimization techniques such as Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO). These methods play a crucial role in aligning model behavior with human expectations in terms of usefulness, safety, and clarity.

Despite these advances, Large Language Models still face several limitations such as hallucinations, outdated knowledge, and limited interpretability of their reasoning processes. To address these challenges, **Retrieval-Augmented Generation (RAG)** has emerged as a powerful solution. It enhances LLMs by retrieving relevant information from external knowledge bases, thereby improving the accuracy and reliability of generated responses, especially for knowledge-intensive tasks. Moreover, RAG enables continuous knowledge updates and the integration of domain-specific information, making language models more adaptable and robust.

Overall, this chapter has established the theoretical foundations necessary for understanding modern language models and their evolution toward intelligent, scalable, and human-aligned systems. Building upon these foundations, the next chapter will focus on the state of the art in Multimodal Retrieval-Augmented Generation (RAG) systems, which extend language models by integrating external knowledge sources and multimodal information to further enhance their reasoning and generation capabilities.

2 Retieval Augmented Generation : State of the Art

2.1 Introduction and Overview of RAG

Large Language Models (LLMs), such as GPT, have demonstrated impressive capabilities in natural language processing. However, they still face several important limitations, including generating incorrect information or "hallucinations" when answering questions outside their training data, as well as the inability to access up-to-date or domain-specific knowledge due to their reliance on static training data.

To address these challenges, **Retrieval-Augmented Generation (RAG)** was introduced. RAG combines the strengths of large language models with external knowledge sources. It retrieves relevant documents from external databases and integrates them into the generation process in order to produce more accurate and reliable responses. This approach helps reduce factual errors, incorporate recent or specialized information, and improve the adaptability of models for knowledge-intensive tasks.

A typical application of RAG is illustrated in Figure 2. A user asks ChatGPT about a recent news event. Since the model relies only on pre-trained knowledge, it may not have access to updated information. In this case, RAG retrieves relevant external documents, such as news articles, from external sources. These documents are then combined with the original query to form an enriched prompt, allowing the model to generate an accurate and up-to-date response.

In this chapter, we explore the concept and architecture of Retrieval-Augmented Generation (RAG) in detail. We present its main components, including indexing, retrieval, and generation. We also introduce the main RAG paradigms, namely Naive RAG, Advanced RAG, and Modular RAG, and analyze their characteristics. Furthermore, we discuss the advantages and limitations of RAG, as well as its main applications in real-world systems such as chatbots, decision support systems, and knowledge-based text generation.[Gao et al., 2023]

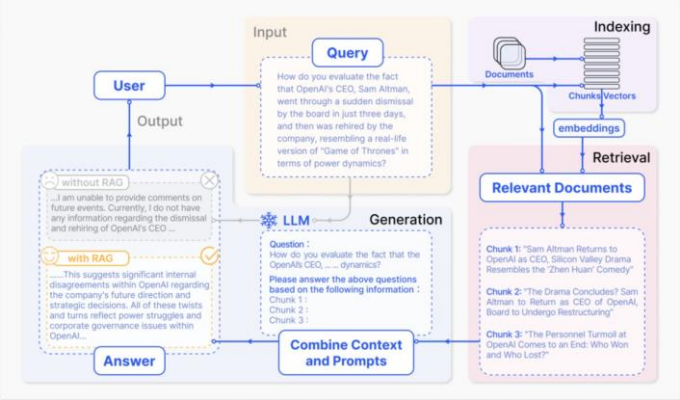

Figure 11 – A representative instance of the RAG process applied to question answering. It mainly consists of 3 steps. 1) Indexing. Documents are split into chunks, encoded into vectors, and stored in a vector database. 2) Retrieval. Retrieve the Top k chunks most relevant to the question based on semantic similarity. 3) Generation. Input the original question and the retrieved chunks together into LLM to generate the final answer. Source: Gao et al. (2024) .

2.2 Definition of RAG

Retrieval-Augmented Generation (RAG) is a framework that enhances large language models (LLMs) by integrating external knowledge sources into the text generation process. Traditional LLMs rely solely on knowledge encoded in their parameters, which can lead to limitations such as hallucinations, outdated information, or lack of domain-specific expertise.

RAG addresses these limitations by retrieving relevant information from external sources such as databases, documents, or knowledge graphs, and combining it with the model's internal knowledge.

The core principle of RAG lies in the synergy between retrieval and generation:

Retrieval: Relevant documents or text chunks are identified using semantic similarity or other search techniques. These sources may include structured data (e.g., knowledge graphs), semi-structured data (e.g., PDFs), or unstructured text. Advanced retrieval techniques include pre-retrieval optimization (such as indexing and query refinement) and post-retrieval processing (such as reranking and context compression).

Generation: The LLM synthesizes the retrieved information along with the user query to produce an accurate and coherent response. This allows the model to generate factually grounded outputs while leveraging its language understanding and reasoning abilities.

Augmentation: RAG systems can further improve performance using techniques such as iterative retrieval, modular pipelines, and task-specific adapters. These strategies ensure that the generated output is relevant, non-redundant, and adapted to different downstream tasks.[Gao et al., 2023]

2.3 Types of RAG

Retrieval-Augmented Generation (RAG) has evolved into three major paradigms: *Naive RAG*, *Advanced RAG*, and *Modular RAG*. These paradigms progressively address the limitations of earlier approaches by improving indexing, retrieval, and generation processes, as well as the integration between them.[Gao et al., 2023]

2.3.1 Naive RAG

Naive RAG represents the simplest and earliest implementation of RAG, commonly referred to as the *Retrieve-Read* framework. It follows a linear pipeline composed of three main stages [Gao et al., 2023]:

— **Indexing**: Raw data is collected from heterogeneous sources such as PDF, HTML, Word, and Markdown files. The data is cleaned, transformed into a unified plain-text format, and split into smaller chunks to address the context length limitations of Large Language Models (LLMs). Each chunk is then encoded into a dense vector representation using embedding models and stored in a vector database to enable efficient similarity search.

— **Retrieval**: When a user submits a query, it is encoded into a vector using the same embedding model. The system computes similarity scores (e.g., cosine similarity) between the query vector and stored document vectors, and retrieves the top-K most relevant chunks. These retrieved documents serve as external context for the model.

— **Generation**: The retrieved context is combined with the user query into a single prompt, which is passed to the LLM to generate the final response. Depending on the configuration, the model may rely strictly on retrieved content or combine it with its internal knowledge.

Despite its simplicity and effectiveness, Naive RAG suffers from several limitations:

— **Retrieval issues**: low precision and recall, missing relevant information, and inclusion of irrelevant content.

— **Generation issues**: hallucinations, weak grounding in retrieved context, and potential incoherence.

— **Augmentation issues**: poor integration of multiple documents, redundancy, and lack of consistency in the generated output.

2.3.2 Advanced RAG

Advanced RAG enhances the basic pipeline by introducing optimization techniques before, during, and after retrieval, while maintaining a sequential architecture.[Gao et al., 2023]

Pre-retrieval optimization focuses on improving both indexing and query formulation:

— Fine-grained text segmentation and sliding window strategies

— Metadata enrichment and structured indexing

— Query rewriting, transformation, and expansion

Retrieval optimization includes:

— Hybrid retrieval (combining dense and sparse methods)

— Multi-query retrieval strategies

— Improved embedding models adapted to the domain

Post-retrieval optimization improves the quality of the retrieved context:

— Reranking retrieved documents based on relevance

— Context compression and summarization to reduce noise

— Filtering redundant or irrelevant information

Additionally, Advanced RAG may employ iterative retrieval mechanisms, where retrieval and generation are performed multiple times to refine the answer. These enhancements significantly improve relevance, reduce noise, and produce more coherent and accurate responses.

2.3.3 Modular RAG

Modular RAG represents the most advanced and flexible paradigm. Unlike previous approaches, it does not rely on a fixed linear pipeline but instead adopts a modular architecture, where components can be independently added, replaced, or optimized.[Gao et al., 2023]

Key modules include:

— **Search Module**: Enables retrieval from multiple sources such as vector databases, search engines, and knowledge graphs.

— **Memory Module**: Stores past interactions and historical knowledge to support multi-turn reasoning and context awareness.

— **Routing Module**: Dynamically selects the most appropriate retrieval strategy or processing pipeline based on the query.

— **Predict Module**: Refines and generates relevant contextual information while reducing noise.

— **Task Adapter Module**: Adapts the system to specific downstream tasks such as question answering, summarization, or domain-specific applications.

Modular RAG also introduces advanced processing patterns and strategies:

— Rewrite → Retrieve → Read (RRR)

— Iterative and recursive retrieval mechanisms

— Hybrid retrieval (dense + sparse)

— Adaptive retrieval strategies (e.g., Self-RAG, FLARE)

This paradigm offers high flexibility, scalability, and adaptability, making it particularly suitable for complex applications, multi-hop reasoning, and enterprise-level systems.

2.4 Data Types, Retrieval Techniques, and Generation Optimization in RAG

2.4.1 Data Types in RAG

Retrieval-Augmented Generation (RAG) systems rely on external knowledge sources, which can be categorized into three main types:

Unstructured Data Unstructured data refers to raw textual information without a predefined format, such as articles, documents, and domain-specific corpora (e.g., medical or legal texts). This type of data is the most commonly used in RAG systems due to its availability and richness.

However, it presents challenges such as [Gao et al., 2023]:

— Difficulty in extracting precise information

— Presence of noise and redundancy

Semi-Structured Data Semi-structured data includes formats such as PDF files that contain both textual content and structured elements like tables and figures.[Gao et al., 2023]

Challenges include:

— Loss of semantic meaning when splitting tables into text chunks

— Difficulty in computing semantic similarity for tabular data

To address these issues, recent approaches use:

— Converting tables into textual descriptions

— Leveraging LLMs to translate text into structured queries (e.g., Text-to-SQL)

Structured Data Structured data is highly organized and includes databases and Knowledge Graphs.[Gao et al., 2023]

Knowledge Graphs are particularly important because they:

— Represent information as entities and relationships

— Enable precise and explainable retrieval

— Support multi-hop reasoning

Remark: This type is highly relevant for graph-based RAG systems using Graph Neural Networks (GNNs), which enhance reasoning over relational data.

2.4.2 Advanced Retrieval Techniques

Hybrid Retrieval Hybrid retrieval combines:

— Sparse retrieval (e.g., BM25), which relies on keyword matching

— Dense retrieval (embeddings), which captures semantic similarity

This combination improves performance by handling both exact matches and semantic variations.

Query Transformation User queries are often not optimal for retrieval. Therefore, several transformation techniques are applied [Gao et al., 2023]:

- **Query Rewriting**: reformulating the query to improve clarity and retrieval effectiveness
- **HyDE (Hypothetical Document Embedding)**: generating a hypothetical answer and using it as a search query
- **Step-back Prompting**: abstracting the query to a higher-level concept to improve retrieval coverage

Query Routing Query routing determines the best retrieval strategy or data source based on the query:

- Metadata-based routing (using keywords or tags)
- Semantic routing (based on meaning)
- Hybrid routing (combining both approaches)

2.4.3 Generation and Context Optimization

After retrieval, the quality of the generated response depends heavily on how the retrieved context is processed.[Gao et al., 2023]

Reranking Retrieved documents are reordered so that the most relevant information appears first. This improves the effectiveness of the input provided to the LLM.

Context Compression Providing too much information can degrade performance due to noise and the ”lost in the middle” problem. Therefore:

- Irrelevant content is removed
- Important information is summarized

Model Fine-tuning Fine-tuning the LLM can further improve:

- Domain adaptation (e.g., medical, legal)
- Output style and structure

2.4.4 Advanced Retrieval Strategies

Iterative Retrieval In this approach, retrieval is performed multiple times, where each step uses previous outputs to refine the search.

Advantage: improves answer quality progressively **Limitation:** may introduce irrelevant information

Recursive Retrieval The problem is decomposed into smaller sub-queries, allowing deeper exploration of the knowledge base.

This is particularly useful for:

— Complex queries

— Multi-step reasoning tasks

Adaptive Retrieval In adaptive retrieval, the model dynamically decides:

— When to retrieve information

— What information to retrieve

This approach represents a step toward intelligent agent-based systems.

2.5 Comparison Between RAG and Fine-tuning

From a methodological perspective, Retrieval-Augmented Generation (RAG) and Fine-tuning represent two fundamentally different paradigms for incorporating knowledge into Large Language Models (LLMs).[Gao et al., 2023]

Retrieval-Augmented Generation (RAG) RAG follows a *non-parametric* approach, where knowledge is stored externally in resources such as vector databases or knowledge graphs. During inference time, relevant information is dynamically retrieved and integrated into the input context.

This implies that:

— Knowledge can be updated in real time without retraining the model

— The model operates as a **retrieval-augmented reasoning system**

Fine-tuning In contrast, Fine-tuning relies on *parametric learning*, where knowledge is embedded the model parameters through training on domain-specific data.

As a result:

— Knowledge is implicitly stored within the model

— Updating knowledge requires **costly retraining**

Criterion	RAG	Fine-tuning
Knowledge Storage	External (non-parametric)	Internal (parametric)
Knowledge Update	Real-time	Requires retraining
Adaptability	High	Limited after training
Computational Cost	Lower (training)	High (training-intensive)
Explainability	High (traceable sources)	Low (implicit knowledge)

Table 1 – Analytical comparison between RAG and Fine-tuning

Analytical Comparison

Discussion RAG is particularly suitable for tasks that require:

— Up-to-date information

— High explainability

On the other hand, Fine-tuning is more appropriate when:

— The goal is to **adapt model behavior** (e.g., style or format)

— High performance is needed in a narrow domain

Conclusion: Recent studies suggest that combining both approaches (a hybrid strategy) often leads to the best performance, where RAG provides external knowledge and Fine-tuning optimizes model behavior.

2.6 Evaluation of RAG Systems

Evaluating RAG systems is a complex task, as it involves not only assessing the generated outputs but also the quality of the retrieval process.[Gao et al., 2023]

2.6.1 Retrieval Quality Evaluation

This stage measures the system's ability to retrieve relevant information.

Hit Rate Measures whether the correct document appears in the top-K retrieved results:

$$Hit@K = \frac{\text{Number of queries with correct result in Top-K}}{\text{Total number of queries}}$$

Mean Reciprocal Rank (MRR) Evaluates the rank position of the first correct result:

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{rank_i}$$

where $rank_i$ is the position of the first correct result for query i.

Normalized Discounted Cumulative Gain (NDCG) Takes into account both ranking order and relevance:

$$NDCG = \frac{DCG}{IDCG}$$

It is particularly useful when multiple relevant documents exist with varying importance.

2.6.2 Generation Quality Evaluation

After retrieval, the generated response must be evaluated.

Accuracy Measures the correctness of the generated answer compared to the ground truth.

Faithfulness Evaluates whether the response is grounded in the retrieved context or contains hallucinations:

- Is the answer supported by the retrieved context?
- Or does it introduce unsupported information?

Relevance Measures how well the generated response answers the user's query.

Discussion The main challenge in RAG systems is balancing:

- Retrieving sufficient relevant information (Recall)
- Avoiding irrelevant or noisy content (Precision)

Any weakness in retrieval directly impacts generation quality.

2.7 Applications of RAG

Retrieval-Augmented Generation (RAG) has gained popularity due to its ability to enhance language models with external knowledge. Its applications span multiple domains [Gao et al., 2023]:

2.7.1 Question Answering Systems

- **Open-domain QA:** Answer questions using a large corpus of documents (e.g., Wikipedia, scientific articles).
- **Customer Support:** Provide accurate answers using company documentation.
- **Medical QA:** Assist doctors by retrieving evidence-based medical information.

2.7.2 Summarization

- **Document Summarization:** Generate summaries using retrieved content from large document collections.
- **News Summarization:** Quickly synthesize news articles with context from multiple sources.

2.7.3 Personalized Assistants

- RAG enables chatbots to incorporate **user-specific data** dynamically.
- Applications include scheduling, recommendations, and task management.

2.7.4 Code Generation and Debugging

— Retrieve relevant code snippets from large repositories.

— Assist developers in generating or debugging code based on prior examples.

2.7.5 Scientific Research Assistance

— Retrieve relevant research papers, datasets, and experimental results.

— Help in drafting literature reviews and generating context-aware insights.

2.7.6 Legal and Compliance

— Retrieve legal clauses or regulations to answer compliance-related questions.

— Support law professionals with fast and accurate information retrieval.

2.7.7 Content Creation and Summarization for Marketing

— Generate content that incorporates knowledge from multiple sources.

— Summarize product reviews or social media feedback.

2.7.8 Knowledge Management in Enterprises

— Retrieve and summarize internal documents, emails, and reports.

— Enable employees to access knowledge efficiently without manual searching.

2.7.9 Key Advantages in Applications

— **Dynamic Knowledge Integration:** LLMs are not limited to training data.

— **Reduced Hallucination:** Providing factual context reduces incorrect generation.

— **Scalability:** Can adapt to new domains without retraining the entire model.

Although traditional RAG systems improve factual grounding through external retrieval, they mainly rely on textual information and remain insufficient for applications involving heterogeneous data such as images, audio, and videos. To overcome these limitations, Multimodal Retrieval-Augmented Generation (MM-RAG) was introduced.

2.8 Multimodal Retrieval-Augmented Generation (MM-RAG)

2.8.1 Introduction et Contexte

Traditional Retrieval-Augmented Generation (RAG) systems mainly rely on textual data to retrieve relevant information and support the response generation

process. Although these systems have demonstrated strong performance in question answering and knowledge-intensive tasks, they remain limited when dealing with heterogeneous real-world data sources such as images, videos, audio recordings, tables, and structured documents.[Gao et al., 2023]

The term *heterogeneous data* refers to different types of information that possess distinct formats and structures. For example:

— text contains linguistic information,
— images contain visual information,
— audio contains acoustic signals,
— videos contain both visual and temporal information.

In many real-world applications, important contextual information is distributed across multiple modalities. A *modality* represents a specific type of data or information source processed by the system. For instance:

— in healthcare systems, diagnosis may depend simultaneously on textual medical reports and radiological images,
— educational platforms may combine textual explanations with diagrams and videos,
— intelligent assistants may process both speech and visual inputs.

Consequently, relying only on textual retrieval often leads to incomplete contextual understanding and lower response quality.

To overcome these limitations, **Multimodal Retrieval-Augmented Generation (MM-RAG)** emerged as an extension of traditional RAG systems. MM-RAG integrates multiple modalities into the retrieval and generation pipeline in order to improve contextual understanding and reasoning capabilities.

The term *retrieval pipeline* refers to the sequence of operations used to:

1. retrieve relevant information,
2. process retrieved data,
3. generate the final response.

Unlike traditional RAG architectures that retrieve only textual documents, MM-RAG systems are capable of retrieving and processing heterogeneous information sources such as:text, image, audio, and video.

The primary objective of MM-RAG is to improve: response relevance, factual consistency, contextual understanding, semantic reasoning.

The term *semantic reasoning* refers to the ability of a system to understand the meaning and relationships between different pieces of information rather than relying only on keyword matching.

By combining multimodal retrieval mechanisms with Large Language Models (LLMs), MM-RAG systems generate richer and more informative responses compared to single-modality systems.

Recently, MM-RAG has attracted increasing interest in both academia and industry due to its applicability in several domains including: healthcare, multimedia analysis, scientific research, education, e-commerce, intelligent conversational systems.

2.8.2 General Architecture of MM-RAG

A typical MM-RAG architecture follows a multi-stage pipeline designed to process heterogeneous data modalities efficiently. The system generally consists of several interconnected components responsible for:

— data preprocessing,
— embedding generation,
— retrieval,
— response generation.

Data Collection and Preprocessing The first stage consists of collecting multimodal data from different sources such as:textual documents, images, videos, audio recordings, web pages, PDF documents, structured databases. Since each modality has different characteristics, preprocessing operations vary according to the data type. The term *preprocessing* refers to the set of operations applied to raw data before it is used by machine learning models. The main objective of preprocessing is to clean, standardize, and transform data into suitable formats for retrieval and embedding generation.[Gao et al., 2023]

Text Preprocessing For textual data, preprocessing operations may include: **Text Cleaning**: consists of removing unnecessary or noisy elements such as special characters, HTML tags, duplicated spaces, and irrelevant symbols. This operation improves text quality and reduces noise during retrieval.[Gao et al., 2023] **Tokenization**: refers to dividing text into smaller units called tokens. A token may represent a word, a sub-word, or a character. For example, *"Artificial intel-ligence improves retrieval"* may be tokenized into: Artificial, intelligence, improves, retrieval. **Stop-word Removal**: Stop-words are very common words such as "the", "is", "and". These words often carry limited semantic meaning. Removing them reduces unnecessary information and improves re-trieval efficiency. **Chunk Segmentation**: consists of dividing long documents into smaller sections called chunks. This operation is necessary because Large Language Models have limited context windows and cannot process extremely long documents efficiently.

Image and Video Preprocessing For images and videos, several preprocessing operations are commonly applied before generating embeddings:

— **Resizing**: changing the dimensions to a fixed size required by deep learning models; reduces computational complexity and ensures consistent input formatting.
— **Normalization**: adjusting pixel values into a standardized numerical range (typically [0, 1]); stabilizes model performance during training and inference.
— **Feature Extraction**: identifying important visual characteristics such as shapes, textures, colors, and object patterns.
— **Frame Sampling**: selecting a subset of frames from a video sequence to reduce computational cost while preserving temporal information.[Gao et al., 2023]

Audio Preprocessing Audio data requires several preprocessing operations:

- **Speech-to-Text Conversion**: transforms spoken language into textual representations using Automatic Speech Recognition (ASR) systems.
- **Spectrogram Extraction**: converts audio signals into frequency-based representations capturing pitch, intensity, and temporal acoustic patterns.
- **Noise Reduction**: removes unwanted background sounds to improve audio quality and retrieval accuracy.

Embedding Generation After preprocessing, each modality is transformed into dense vector representations called *embeddings*. Embeddings are numerical vectors that capture the semantic meaning of data. The objective is to represent semantically similar information using nearby vector representations in high-dimensional spaces.

Text Embeddings: generally generated using transformer-based language models such as BERT [**?**], SentenceTransformers [**?**], or T5-based encoders [**?**].

Visual Embeddings: generated using vision-language models such as CLIP [**?**] and BLIP-2 [**?**]. These models transform images into semantic vector representations aligned with textual embeddings within a shared semantic space, enabling cross-modal retrieval.

Cross-modal Retrieval: refers to the ability to retrieve information from one modality using a query from another modality (e.g., a textual query retrieving relevant images).

Vector Storage and Indexing Once embeddings are generated, they are stored inside specialized systems called *vector databases*. Popular vector databases include FAISS [**?**], Chroma, Pinecone, and Milvus. These systems organize em-beddings using indexing techniques that accelerate similarity search operations.

Retrieval Module During inference, the user query is converted into an embedding representation. The retrieval module computes semantic similarity scores between the query embedding and stored multimodal embeddings. Several retrieval strategies may be employed:

- **Dense Retrieval**: relies on semantic embeddings to identify relevant information based on meaning similarity.
- **Sparse Retrieval**: relies primarily on keyword matching techniques such as BM25 [**?**].
- **Hybrid Retrieval**: combines sparse and dense retrieval to capture both exact keyword matches and semantic similarities.
- **Cross-modal Retrieval**: enables interaction between multiple modalities such as text, images, and audio.

Generation Module The retrieved multimodal context is integrated into the prompt provided to the Large Language Model. The generation module synthesizes retrieved information and produces the final response. Large Language Models combine:

— retrieved external knowledge,
— internal parametric knowledge,

to generate coherent and contextually grounded outputs.

In MM-RAG systems, the generation process may involve:

— **Multimodal Fusion**: combining information from multiple modalities into a unified representation.
— **Contextual Reasoning**: understanding relationships between retrieved information and the user query.
— **Cross-modal Alignment**: synchronizing semantic representations between different modalities.
— **Semantic Aggregation**: combining related information from multiple sources to produce coherent outputs.

2.8.3 Modalities Used in MM-RAG

Text Modality Text remains the most commonly used modality in RAG systems. Textual corpora may include articles, reports, scientific papers, legal documents, and domain-specific datasets. Text embeddings enable semantic retrieval and contextual augmentation for large language models.

Image Modality Image-based retrieval extends the capabilities of traditional RAG systems by incorporating visual understanding. Vision-language models such as BLIP-2 generate descriptive captions and semantic representations from images. Similarly, CLIP aligns image and textual embeddings into a unified semantic space. This modality is particularly useful in medical imaging, visual question answering, multimedia retrieval, and e-commerce recommendation systems.

Audio Modality Audio retrieval systems process speech and acoustic information using speech recognition and audio embedding techniques. Applications include speech assistants, meeting summarization, audio retrieval systems, and multilingual conversational agents.

Video Modality Video-based MM-RAG systems extract temporal and semantic information from video sequences by combining visual frame analysis, temporal embeddings, subtitle retrieval, and speech analysis. Video retrieval is essential in surveillance systems, multimedia understanding, educational platforms, and video summarization.

Code Modality Code-oriented MM-RAG systems retrieve relevant source code, documentation, and programming examples to support code generation, debugging, software maintenance, and automatic documentation generation.

2.8.4 Advantages of MM-RAG

MM-RAG systems offer several important advantages over traditional text-based RAG approaches:

1. **Improved contextual understanding**: multimodal integration combines complementary information sources, enabling richer semantic relationships and more informative responses.
2. **Enhanced retrieval quality**: cross-modal semantic alignment allows textual queries to retrieve relevant visual or auditory information.
3. **Reduced hallucinations**: integrating multiple modalities grounds the generation process in diverse external knowledge sources.
4. **Domain adaptability**: MM-RAG enhances flexibility across domains where heterogeneous information plays a critical role.
5. **Advanced reasoning**: multimodal systems support more sophisticated reasoning capabilities compared to single-modality architectures.

2.8.5 Limitations of MM-RAG

Despite their advantages, MM-RAG systems still face several important limitations[Gao et al., 2023]:

— **Computational complexity**: processing multiple modalities requires large computational resources, memory capacity, and storage infrastructure.
— **Multimodal alignment**: integrating heterogeneous semantic structures into a unified representation space remains challenging.
— **Fusion redundancy**: multimodal fusion mechanisms may introduce noisy contextual information, negatively impacting retrieval precision.
— **Scalability**: handling large-scale multimodal datasets presents significant engineering challenges.
— **Limited relational reasoning**: current MM-RAG systems often fail to model explicit semantic relationships and complex dependencies between entities, particularly for multi-hop reasoning tasks.

2.8.6 Transition Toward Graph-based RAG

Although MM-RAG systems significantly improve contextual understanding through multimodal integration, they remain insufficient for modeling complex semantic relationships and structured reasoning tasks. Traditional retrieval mechanisms primarily rely on semantic similarity between embeddings and often ignore explicit relational structures between entities.

To overcome these limitations, graph-based retrieval approaches such as **GraphRAG** and **Knowledge Graph RAG (KG-RAG)** have emerged. These approaches leverage graph structures, entity relationships, and struc-tured knowledge representations to enhance semantic retrieval and relational reasoning capabilities. Furthermore, recent research increasingly combines Graph Neural Networks (GNNs) with graph-based RAG systems to learn richer relational embeddings and improve reasoning performance over complex knowledge graphs.[Gao et al., 2023]

2.9 GraphRAG

2.9.1 Definition and Core Concepts

GraphRAG is an advanced form of Retrieval-Augmented Generation (RAG) that integrates graph structures into the retrieval process. Instead of retrieving isolated text chunks only, GraphRAG organizes information as interconnected entities and relations, allowing the system to perform relational and multi-hop reasoning [Han et al., 2024].

Traditional RAG systems retrieve documents using vector similarity but often ignore semantic relationships between concepts. GraphRAG addresses this limitation by representing knowledge as graphs where [Han et al., 2024]:

— **Nodes** represent entities or concepts,
— **Edges** represent semantic relations,
— **Paths** enable reasoning across multiple connected facts.

Graph Structures and Relations GraphRAG relies on graph-based representations such as Knowledge Graphs, Heterogeneous Graphs, and Property Graphs. In these graphs, entities (persons, locations, events, concepts) are represented as nodes, and relationships (located in, part of, related_to) are represented as edges [Han et al., 2024].

Multi-Hop Retrieval One of the main strengths of GraphRAG is multi-hop retrieval. Instead of retrieving a single document independently, the sys-tem traverses multiple connected nodes in the graph to infer complex answers [Han et al., 2024]:

1. Retrieve an entity,
2. Explore neighboring relations,
3. Follow graph paths,
4. Aggregate relevant evidence.

Reasoning Capabilities GraphRAG enhances reasoning through relational inference, semantic traversal, context propagation, and graph connectivity analysis. By leveraging graph structures, the model can infer hidden relations and provide more coherent responses while reducing hallucinations [Han et al., 2024].

2.10 KG-RAG

2.10.1 Definition and Components

Knowledge Graph-based Retrieval-Augmented Generation (KG-RAG) combines Large Language Models with Knowledge Graphs to improve factual accuracy and semantic reasoning. In KG-RAG systems, retrieval is performed not only from textual embeddings but also from structured graph knowledge [Han et al., 2024].

Knowledge Graph Components A Knowledge Graph generally consists of[Han et al., 2024]:
— **Entities**: real-world objects or concepts,
— **Relations**: semantic connections between entities,
— **Triples**: structured representations in the form (*subject, relation, object*).
Example: (Santiago, capital of, Chile).

Retrieval in Knowledge Graphs KG-RAG retrieves information by traversing graph relations and selecting semantically relevant subgraphs. Compared with standard vector retrieval, graph-based retrieval offers [Han et al., 2024]:
— better contextual understanding,
— explainable retrieval paths,
— improved relational reasoning.

The retrieval process usually involves: entity extraction, graph traversal, neighbor aggregation, and context generation for the LLM.

Semantic Reasoning A major advantage of KG-RAG is semantic reasoning. Using graph relations, the system can infer indirect connections, identify hidden dependencies, support multi-step reasoning, and improve answer consistency. Therefore, KG-RAG is particularly useful for complex question-answering tasks requiring structured reasoning [Han et al., 2024].

2.11 GNN for Knowledge Graphs

2.11.1 Definition of Graph Neural Networks

Graph Neural Networks (GNNs) are deep learning models designed to process graph-structured data. Unlike traditional neural networks, GNNs directly exploit graph connectivity and neighborhood information. They are widely used in Knowledge Graph applications because they can learn semantic representations from entities and relations.

Graph Convolutional Networks (GCN) GCN applies convolution operations on graph nodes by aggregating information from neighboring nodes. Main roles: capture local graph structure, learn contextual node embeddings, and propagate semantic information. GCNs are useful in KG-RAG because they improve relational representation learning [**?**].

Graph Attention Networks (GAT) GAT introduces attention mechanisms into graph learning. Instead of treating all neighbors equally, GAT assigns different importance weights to neighboring nodes. Advantages: better handling of heterogeneous relations, improved semantic focus, and adaptive neighborhood aggregation [**?**].

GraphSAGE GraphSAGE is a scalable GNN architecture that samples neighborhoods instead of processing the entire graph. Main advantages: scalability to large graphs, reduced computational complexity, and inductive learning capabil-ity [**?**]. GraphSAGE is particularly suitable for large-scale KG-RAG systems.

Graph Embeddings Graph Embeddings transform graph nodes and relations into dense vector representations. These embeddings preserve semantic similarity, structural proximity, and relational information. Graph embeddings are essential in KG-RAG because they improve retrieval quality, semantic matching, and reasoning efficiency.

2.12 Datasets and Benchmarks

Several benchmark datasets are commonly used to evaluate GraphRAG and KG-RAG systems:

ComplexWebQuestions
: contains complex multi-hop questions requiring reasoning across multiple facts [**?**].

HotpotQA
: is designed for multi-document and multi-hop reasoning tasks; evaluates the ability to combine evidence from different sources [**?**].

WebQSP
: focuses on question answering over structured Knowledge Graphs [**?**].

MetaQA
: evaluates reasoning over movie-related Knowledge Graphs using multi-hop queries [**?**].

Evaluation Metrics The performance of GraphRAG and KG-RAG systems is commonly measured using:

— **Accuracy**: measures correct predictions,
— **F1-score**: balances precision and recall,
— **Recall**: evaluates retrieval completeness.

2.13 Research Gaps

Although **Retrieval-Augmented Generation (RAG)** has significantly improved the factuality and reliability of Large Language Models (LLMs), several limitations remain unresolved. Traditional RAG approaches mainly depend on retrieving relevant documents from external sources before generating responses. However, these methods still suffer from several challenges, including hallucinations generated by LLMs, weak relational reasoning between retrieved information, and limited understanding of structured semantic knowledge. Furthermore,

traditional RAG systems often struggle with complex multi-hop questions that require reasoning across multiple pieces of information and establishing connections between different entities.

Graph-based Retrieval-Augmented Generation (GraphRAG) has been introduced to address some of these limitations by incorporating graph structures that represent relationships between entities and concepts. Despite its advantages, GraphRAG approaches face several challenges. The construction and management of very large graph structures require significant computational resources. In addition, graph traversal operations can become complex and costly, especially when dealing with large-scale knowledge sources. These limitations affect the scalability and efficiency of GraphRAG systems in real-world applications.

Knowledge Graph-based RAG (KG-RAG) enhances retrieval by exploiting structured knowledge graphs to provide more meaningful and connected information. However, the performance of KG-RAG systems strongly depends on the quality and completeness of the underlying knowledge graphs. Incomplete knowledge graphs, noisy or missing relations, and difficulties in maintaining updated graph information remain major challenges. Moreover, integrating multimodal information, such as text, images, and other data formats, into knowledge graphs introduces additional complexity.

Graph Neural Networks (GNNs) have demonstrated strong capabilities for learning from graph-structured data and performing relational reasoning. Nevertheless, their application in large-scale RAG systems is still limited by several issues. These include the over-smoothing problem, where node representations become indistinguishable after multiple propagation layers, high memory requirements for processing large graphs, and scalability limitations when handling massive graph datasets. Furthermore, training GNN models on large-scale graphs can be computationally expensive, which restricts their deployment in resource-constrained environments.

Therefore, despite the progress achieved by RAG, GraphRAG, KG-RAG, and GNN-based approaches, there remains a need for more efficient and scalable frameworks that combine the reasoning capabilities of graphs with the generative power of LLMs. Addressing these research gaps is essential for developing robust systems capable of providing accurate, explainable, and context-aware responses to complex queries.

2.14 Positioning of Our Work

To address these limitations, our work proposes a **multimodal GraphRAG architecture** integrating Knowledge Graph and Graph Neural Network techniques in order to improve:

— relational reasoning,
— semantic retrieval,
— answer generation quality.

The proposed approach aims to combine graph-based retrieval with graph embeddings and multimodal representations to enhance the performance of

Knowledge Graph Retrieval-Augmented Generation systems.

2.15 Conclusion

In this chapter, we explored the fundamental concepts and evolution of Retrieval-Augmented Generation (RAG), a framework that enhances Large Language Models by incorporating external knowledge sources into the generation process. We presented the main components of RAG systems, including indexing, retrieval, and generation, and discussed how they contribute to improving response accuracy, reducing hallucinations, and providing access to up-to-date information.

We also examined the three main RAG paradigms—Naive RAG, Advanced RAG, and Modular RAG—and analyzed their characteristics, strengths, and limitations. Furthermore, we reviewed the different types of data that can be integrated into RAG systems, ranging from unstructured documents to structured knowledge graphs, as well as advanced retrieval and context optimization techniques designed to improve overall system performance.

In addition, a comparison between RAG and Fine-tuning highlighted the complementary nature of these approaches, while the evaluation metrics presented provide a framework for assessing both retrieval effectiveness and generation quality.

Overall, RAG has become a key technology for building reliable and knowledge-intensive AI applications. Understanding its architectures, retrieval strategies, and evaluation methods provides the necessary theoretical foundation for the development of effective question-answering systems. The next chapter presents the proposed approach, including the system architecture, implementation details, and the evaluation methodology used to assess its performance.

3 Approach and Evaluation

3.1 Objective

: Based on insights from state-of-the-art approaches, we propose a system that leverages four distinct retrieval-augmented generation (RAG) methods to exploit the ConceptNet knowledge graph : a Zero-shot LLM Baseline [Touvron et al., 2023], two KG-RAG [Baek et al., 2023] pipelines with different entity extraction strategies (LLM-based and NLP-based), and a GNN-augmented architecture (QAGNN + RoBERTa) [Yasunaga et al., 2021].

3.2 RAG pipelines

3.2.1 Approach 1: Zero-shot LLM Baseline

Approach 1 serves as the reference baseline. The LLaMA-3.1-8b model is queried directly with the question and its four answer choices, without any external knowledge or contextual augmentation. This setup evaluates the intrinsic zero-shot reasoning capability of the language model.

The prompt follows the structure:

\texttt{System: You are a concise QA assistant. Answer with only the letter (A, B, C or D).}\\
\texttt{User: Question: \{question\}\\
\{choices\}\\
Answer:

Figure12 illustrates the overall architecture of the Zero-shot LLM baseline pipeline.

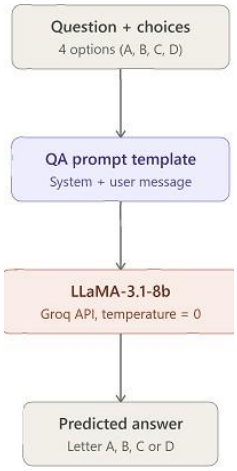

Figure 12 – Zero-shot LLM Baseline (Approach 1). Source: Author's own illustration generated with AI assistance.

Hyperparameter Configuration:

Table 2 – Hyperparameter Configuration — Approach 1 (Zero-shot LLM Baseline)

Hyperparameter	Value
Language Model	LLaMA-3.1-8b-instant (via Groq API)
Temperature	0.0
Max Tokens	20
External Knowledge	None
LLM Calls per Question	1
Evaluation Set Size	500 questions

Results:

Evaluated on the 500-question OpenBookQA validation split, Approach 1 achieves an accuracy of **80.80%**, demonstrating the strong zero-shot reasoning capability of LLaMA-3.1-8b on structured science questions without any external knowledge.

3.2.2 Approach 2: Boosting LLM Reasoning with KG-RAG via LLM-based Entity Extraction

Description

Approach 2 augments the baseline LLM with external knowledge retrieved from ConceptNet. The key novelty of this approach lies in using the LLM itself to extract the relevant entities from the question before querying the knowledge graph. This constitutes a fully LLM-driven Retrieval-Augmented Generation (RAG) pipeline over a structured knowledge graph.

The pipeline proceeds in three steps:

1. **LLM-based Entity Extraction:** The LLM is prompted to return a JSON array of the most important keywords from the question (maximum 4 entities).
2. **Knowledge Graph Retrieval:** The extracted entities are used to query a local SQLite index of ConceptNet, retrieving up to 8 semantic relations per entity.
3. **Knowledge-Augmented QA:** The retrieved relations are injected into the LLM prompt as a knowledge context, and the model produces its final answer.

The entity extraction prompt follows this structure:

System: You are a keyword extractor. Given a question, return ONLY a JSON array of the most important keywords or concepts, lowercase, no duplicates. Maximum 4 items. No explanation, no markdown, just the JSON array.
User: Question: {question}

The knowledge-augmented QA prompt follows this structure:

System: You are a knowledge-augmented QA assistant. Use the knowledge to answer with only the letter (A, B, C or D).
User: Knowledge:\n{context}\n\n Question: {question}\n{choices}\n\nAnswer:

Hyperparameter Configuration:

Table 3 – Hyperparameter Configuration — Approach 2 (KG-RAG via LLM-based Entity Extraction)

Hyperparameter	**Value**
Language Model	LLaMA-3.1-8b-instant (via Groq API)
Temperature	0.0
Max Tokens (QA)	20
Max Tokens (Entity Extraction)	60
Entity Extraction Method	LLM-based (JSON prompt)
Max Entities per Question	4
Max Relations Retrieved	8
Knowledge Source	ConceptNet (local SQLite)
LLM Calls per Question	3 (1 extraction + 1 QA + 1 baseline)
Evaluation Set Size	500 questions
Runtime	≈ 56 minutes

Results:

Approach 2 achieves an accuracy of **78.00%** on the 500-question validation set, representing a decrease of **2.80%** compared to the baseline. Among the 500 questions, only 5 returned no ConceptNet context (fallback to baseline). The runtime of approximately 56 minutes reflects the cost of three sequential LLM calls per question.

3.2.3 Approach 3: Boosting LLM Reasoning with KG-RAG via NLP-based Entity Extraction

Description

Approach 3 follows the same KG-RAG pipeline as Approach 2, with one key difference: entity extraction is performed by a traditional NLP pipeline (spaCy) rather than the LLM. This eliminates one LLM call per question and reduces the total inference time while maintaining the same knowledge augmentation strategy.

The spaCy pipeline applies Part-of-Speech (POS) tagging and lemmatization to identify the most relevant nouns and proper nouns from the question. Stop words and domain-specific noise terms are filtered out, and the remaining entities

are cleaned before being used to query ConceptNet.

Hyperparameter Configuration

Table 4 – Hyperparameter Configuration — Approach 3 (KG-RAG via NLP-based Entity Extraction)

Hyperparameter	Value
Language Model	LLaMA-3.1-8b-instant (via Groq API)
Temperature	0.0
Max Tokens (QA)	20
Entity Extraction Method	NLP-based (spaCy en_core_web_sm)
POS Tags Retained	NOUN, PROPN
Max Entities per Question	4
Max Relations Retrieved	8
Knowledge Source	ConceptNet (local SQLite)
LLM Calls per Question	2 (1 baseline + 1 QA)
Evaluation Set Size	500 questions
Runtime	≈ 30 minutes

Results

Approach 3 achieves an accuracy of **78.00%**, identical to Approach 2, while reducing inference time by approximately **46%** (from 56 to 30 minutes). Among the 500 questions, 8 returned no ConceptNet context (fallback to baseline). This result confirms that NLP-based entity extraction is a more efficient alternative to LLM-based extraction when accuracy is equivalent, making Approach 3 more suitable for deployment.

Figure13 presents the general architecture of the KG-RAG pipeline used in Approaches 2 and 3. Both approaches follow the same overall framework, where external knowledge is retrieved from the ConceptNet knowledge graph and integrated into the LLM prompt to enhance the reasoning process. The main difference lies in the entity extraction step: Approach 2 relies on an LLM-based entity extraction strategy, while Approach 3 uses a traditional NLP-based extraction method with spaCy. This shared architecture highlights the role of knowledge retrieval and augmentation, while allowing a direct comparison between the two entity extraction strategies in terms of accuracy and computational efficiency.

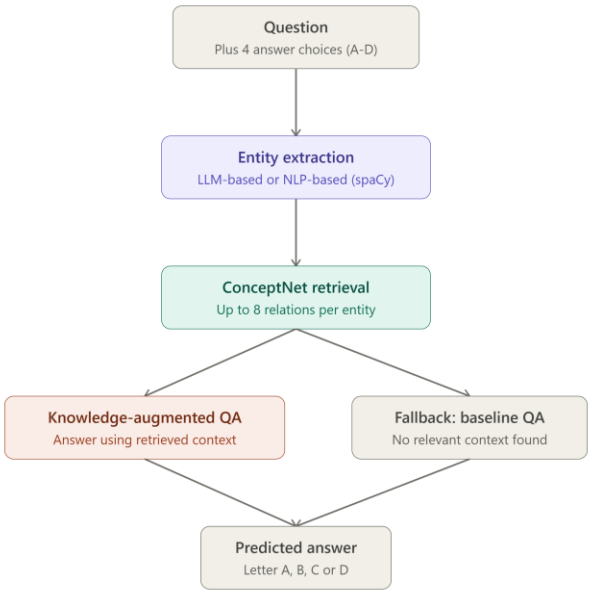

Figure 13 – KG-RAG pipeline (Approaches 2 et 3). Source: Author's own illustration generated with AI assistance.

3.2.4 Approach 4: Boosting LLM Reasoning with KG-RAG via GNN (QAGNN + RoBERTa)

Description

Approach 4 replaces the retrieval-based KG-RAG pipeline with a fully trained Graph Neural Network architecture — QAGNN — that jointly encodes the question using RoBERTa and reasons over a ConceptNet subgraph using a Graph Attention Network (GAT). Unlike Approaches 2 and 3, which use the LLM at inference time with retrieved knowledge, Approach 4 integrates structured knowledge directly into the model architecture through learned graph message passing.

Hyperparameter Configuration

Table 5 – Hyperparameter Configuration — Approach 4 (QAGNN + RoBERTa)

Hyperparameter	Value
Encoder Model	RoBERTa-Base
Encoder Learning Rate	1×10^{-5}
Decoder Learning Rate	3×10^{-4}
Optimizer	AdamW
Number of Epochs	20
Batch Size	32
Mini Batch Size	4
GNN Hidden Dimension	200
Number of GNN Layers (*k*)	5
Attention Heads	2
Dropout	0.2
Weight Decay	0.01
Maximum Nodes per Graph	50
Maximum Sequence Length	128
Number of Relations	38
Random Seed	42

Training Results and Convergence:

The model was trained for 20 epochs with the encoder frozen for the first 4 epochs. Table 6 presents the evolution of validation and test accuracy across key epochs.

Table 6 – QAGNN Training Progression on OpenBookQA

Epoch	Dev Accuracy	Test Accuracy
0	0.3920	0.3660
1	0.4280	0.3740
3	0.4320	0.3820
4	0.4740	0.4420
5	0.5260	0.4880
6	0.5540	0.5280
7	0.5760	0.5500
9	0.5820	0.5560
11	0.5920	0.5540
16 (Best)	**0.6020**	**0.5500**

The best model was obtained at epoch 16, achieving a validation accuracy of **60.20%** and a test accuracy of **55.00%**. Training loss decreased consistently from approximately 1.38 at step 9 to below 0.15 by epoch 19, indicating good convergence behavior.

Figure14 illustrates the overall architecture of the GNN-based reasoning pipeline used in Approach 4 (QAGNN + RoBERTa). Unlike the previous KG-RAG approaches, which rely on retrieving external knowledge and injecting it into an LLM prompt, this approach integrates structured knowledge directly into the model through graph-based reasoning. The question and answer choices are first encoded using the RoBERTa language model to obtain contextual representations. Then, a relevant ConceptNet subgraph is constructed, where nodes represent concepts and edges represent semantic relations. A Graph Attention Network (GAT) performs message passing over this graph to propagate information between connected nodes. Finally, the graph representations are combined with the textual representations from RoBERTa to predict the most likely answer choice. This architecture enables the model to learn reasoning patterns over both textual and structured knowledge representations.

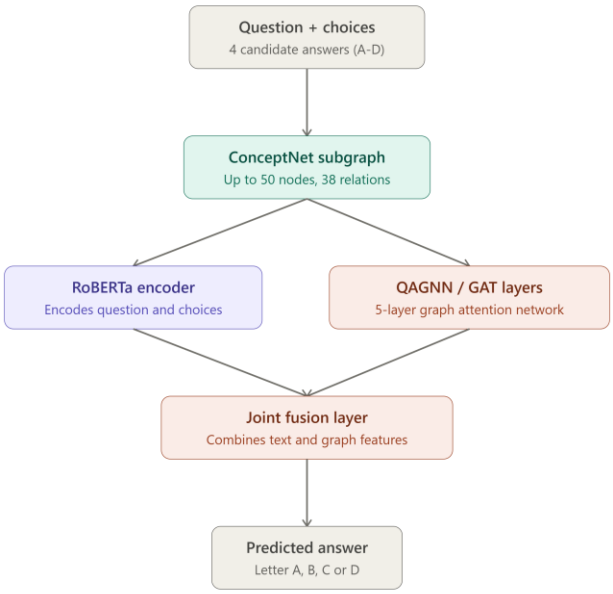

Figure 14 – GNN pipeline (Approach 4 : QAGNN + RoBERTa). Source: Author's own illustration generated with AI assistance.

3.3 Evaluation

3.3.1 Accuracy

Accuracy measures how correct the model predictions are by calculating the proportion of correct results among the total number of results. The formula for Accuracy is:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

where TP denotes True Positives, TN denotes True Negatives, FP denotes False Positives, and FN denotes False Negatives.

3.3.2 Confusion Matrix

A confusion matrix is a simple table used to measure how well a classification model is performing. It compares the predictions made by the model with the actual results, and indicates where the model was correct or incorrect.

— **TP** (True Positive) : the model correctly predicts a positive class.

— **TN** (True Negative) : the model correctly predicts a negative class.

— **FP** (False Positive) : the model incorrectly predicts a positive class.

— **FN** (False Negative) : the model incorrectly predicts a negative class.

Table 7 – Confusion Matrix Structure

	Predicted Positive	**Predicted Negative**
Actual Positive	*TP*	*FN*
Actual Negative	*FP*	*TN*

3.3.3 Datasets

OpenBookQA is a multiple-choice question answering dataset designed to evaluate a model's ability to answer elementary science questions. Each sample consists of a question in natural language, four answer choices labeled A, B, C, and D, and a correct answer key. The training split contains 4,957 samples, while both the validation and test splits contain 500 questions each. This dataset was used consistently across all four approaches to ensure a fair and direct comparison.

3.3.4 Results and discussion

This work investigates four distinct approaches to knowledge-augmented question answering on the OpenBookQA dataset. The evaluated approaches are summarized in Table 8.

Table 8 – Overview of the Four Evaluated Approaches

Approach	Name	Entity Extraction	KG Used	LLM Calls/Q
1	Zero-shot LLM Baseline	None	No	1
2	KG-RAG via LLM-based Extraction	LLM	Yes	3
3	KG-RAG via NLP-based Extraction	spaCy	Yes	2
4	GNN-Augmented LLM (QAGNN)	GNN/RoBERTa	Yes	–

Comparative Results and Analysis

Table 9 summarizes the performance of all four approaches on the Open-BookQA validation set (500 questions).

The results reveal several important observations.

Approach 1 achieves the highest accuracy (80.80%). The zero-shot LLM baseline outperforms all knowledge-augmented approaches, confirming the remarkable reasoning capability of large pre-trained language models on structured science questions without any external context.

Table 9 – Performance Comparison of All Four Approaches on OpenBookQA (500 Questions)

Approach	Accuracy	vs. Baseline	Runtime
Approach 1 – Zero-shot LLM Baseline	**80.80%**	–	–
Approach 2 – KG-RAG via LLM-based Entity Extraction	78.00%	-2.80%	≈ 56 min
Approach 3 – KG-RAG via NLP-based Entity Extraction	78.00%	-2.80%	≈ 30 min
Approach 4 – GNN-Augmented LLM (QAGNN + RoBERTa)	55.00%	-25.80%	Training Required

Approaches 2 and 3 have the same accuracy (78.00%) but different efficiency. Both KG-RAG approaches achieve the same accuracy, 2.80% below the baseline. The knowledge retrieved from ConceptNet did not improve performance, which can be attributed to the nature of the retrieved relations: ConceptNet provides general commonsense associations (e.g., antonyms, etymological relations) that do not always align with the specific scientific reasoning required by OpenBookQA questions. However, Approach 3 (NLP-based extraction via spaCy) is significantly more efficient than Approach 2 (LLM-based extraction), reducing inference time by approximately 46% (from 56 to 30 minutes) at no cost to accuracy. This makes Approach 3 the preferred choice for deployment.

Approach 4 (QAGNN) achieves the lowest accuracy (55.00%). While the GNN-based model integrates structured knowledge directly into its architecture through learned message passing, it is trained on a relatively small dataset (~5K examples) with approximately 125 million parameters. The performance gap relative to LLaMA-3.1-8b reflects primarily the difference in model scale and the volume of pre-training data. Nonetheless, QAGNN demonstrates consistent learning progression across 20 epochs and represents a fundamentally different — and more interpretable — reasoning paradigm.

Discussion. The counterintuitive result that knowledge augmentation (Approaches 2 and 3) slightly degrades LLM performance suggests that the ConceptNet retrieval pipeline introduces noise rather than useful signal for this task. The retrieved relations (e.g., antonyms, etymological links) are often semantically distant from the specific scientific facts required to answer OpenBookQA questions. This points to the importance of retrieval quality over retrieval quantity in KG-RAG systems.

A promising direction for future work is to combine the strengths of Approaches 3 and 4: using a large LLM as the encoder together with a GNN for structured knowledge integration, and replacing ConceptNet with a more

domain-specific knowledge source aligned with elementary science reasoning.

3.4 Conclusion

This chapter presented the experimental setup and comparative evaluation of four approaches to knowledge-augmented question answering on OpenBookQA. Approach 1 (Zero-shot LLM) achieved the highest accuracy at 80.80%, while Approaches 2 and 3 (KG-RAG) reached 78.00% with Approach 3 being signifi-cantly more efficient. Approach 4 (QAGNN) achieved 55.00%, demonstrating the learning capability of graph-augmented architectures under constrained training resources. These results highlight both the power of large-scale language models and the complementary potential of structured knowledge integration for future hybrid architectures.

4 System Implementation and Deployment

4.1 Introduction

This chapter describes the design, implementation, and deployment of the Knowledge Graph RAG system proposed in this work. The system was implemented using the Python programming language due to its flexibility and rich ecosystem for machine learning and deep learning applications, and was developed across multiple environments depending on the pipeline stage.

Specifically, Google Colab was used for rapid experimentation and testing of Approaches 1, 2, and 3 (via the Groq API to access LLaMA-3.1-8b), while Kaggle GPU environments were utilized to leverage hardware acceleration during the training of Approach 4 (QAGNN). Visual Studio Code was employed for development, debugging, and integration of the different system components.

For the graph-based question answering component, PyTorch was used to design and implement neural network architectures, while PyTorch Geometric (PyG) was used to build and train Graph Neural Networks (GNNs). These libraries enabled efficient implementation of message passing mechanisms over graph-structured data, particularly when modeling external knowledge extracted from ConceptNet. The project also relied on NumPy for numerical computations and NetworkX for constructing, manipulating, and visualizing graphs before converting them into formats compatible with GNN processing.

In the natural language processing pipeline, the spaCy library was used for syntactic parsing and entity extraction (Approach 3), while the LLaMA-3.1-8b language model was used for LLM-based entity extraction (Approach 2). The SentenceTransformer (all-MiniLM-L6-v2) model was employed to generate dense vector representations for nodes and questions in the GNN-based approach. ConceptNet was accessed via a local SQLite database built from the official CSV dump, enabling fast offline retrieval of semantic relations.

The datasets used in this work include:

— **OpenBookQA** — a multiple-choice question answering dataset with four answer choices (A, B, C, D) per question, designed to evaluate elementary science knowledge combined with commonsense reasoning.

— **ConceptNet** — a large-scale external knowledge graph used to enrich the model with semantic relationships and structured world knowledge.

To evaluate system performance, Accuracy was used as the primary evaluation metric, as the task is formulated as a multiple-choice classification problem where the model selects the correct answer from a set of four candidates.

Finally, the best-performing model was made accessible through a locally running **Multi RAG web application** and a **FastAPI-based RESTful API** using JSON format, allowing users to interact with the system in real time. The deployed system provides two main interaction modes: the **Multiple Choice Question (MCQ)** mode, offering three distinct reasoning approaches (Baseline, Graph-ConceptNet, and GNN), and the **Free Question** mode, relying on the GNN-RAG pipeline with or without an uploaded document.

4.2 General System Architecture

The deployed system follows a fully local client-server architecture. The backend, built with FastAPI, exposes a RESTful API encapsulating all processing logic: entity extraction, ConceptNet querying, GNN inference, and response generation via the Mistral language model. The React.js frontend runs locally on the user's machine and provides an interactive graphical interface accessible via a browser at localhost:3000.

Table 10 summarizes the technologies used in each layer of the system.

Table 10 – Technology stack of the deployed system

Layer	Technology	Role
Backend	FastAPI (Python)	RESTful server, routing, AI logic
Frontend	React.js	Local interactive user interface
LLM	Mistral (via Ollama)	Local response generation
GNN	GCNConv (PyTorch Geometric)	Graph-based reasoning
Knowledge Base	ConceptNet (SQLite)	Semantic relations
Embedding	SentenceTransformer (MiniLM-L6)	Vector representations
NLP	spaCy + FCoref	Entity extraction, coreference

4.3 Backend Architecture (FastAPI)

The backend is implemented in Python using the FastAPI framework, served via Uvicorn on 127.0.0.1:8000. It exposes several endpoints covering all use cases of the system.

4.3.1 Initialization and Resource Loading

At startup, the startup event() function is triggered automatically and performs the following operations in order:

1. Building or loading the ConceptNet SQLite database from the official CSV dump.
2. Loading the trained GNN model (best model.pt) and switching it to evaluation mode.
3. Loading the QA JSON file (conceptnet 1M qa emb02.jsonl) and pre-computing embedding tensors for cosine similarity search at inference time.

The SentenceTransformer embedding model, the spaCy NLP pipeline, and the FCoref coreference model are also initialized at this stage, ensuring all resources are ready before the first request is processed.

4.3.2 Exposed API Endpoints

Table 11 summarizes all endpoints of the RESTful API.

Table 11 – RESTful API endpoints of the deployed system

Method	Endpoint	Description
POST	/upload	Upload a text file and build the document knowledge graph
POST	/chat	Process a question and return the answer based on the selected approach
GET	/graph	Return a PNG image of the document graph
GET	/Q_GRAPH	Return a PNG image of the current question graph
GET	/download_graph	Download the knowledge graph in GEXF format

4.3.3 The /upload Endpoint — Document Graph Construction

The POST /upload endpoint accepts a plain text file (.txt). The backend reads the file content, applies the full NLP pipeline (syntactic parsing and coreference resolution via FCoref) to extract subject-relation-object triples, and builds a directed graph using NetworkX. This graph is then converted to PyTorch Geometric format, and node embeddings are computed using the GNN. The resulting representations are stored in memory for use during subsequent free-question queries.

4.3.4 The /chat Endpoint — Approach-Based Routing

The POST /chat endpoint receives a JSON object with four fields: query, choices_str, question type ("qcm" or "libre"), and approach. Routing is performed based on these parameters across four distinct processing paths described in Section 4.5.

4.4 Frontend Architecture (React.js)

The user interface is developed with React.js and communicates with the backend via HTTP requests. It is organized into two main areas: a **left sidebar** for document graph management, and a **central chat area** for question-answer interaction.

4.4.1 General Interface Overview

Figure **??** shows the main interface at startup, with the MCQ mode and Baseline approach selected. The left sidebar contains the file upload controls, while the central area displays the configuration selectors and the four-option input form.

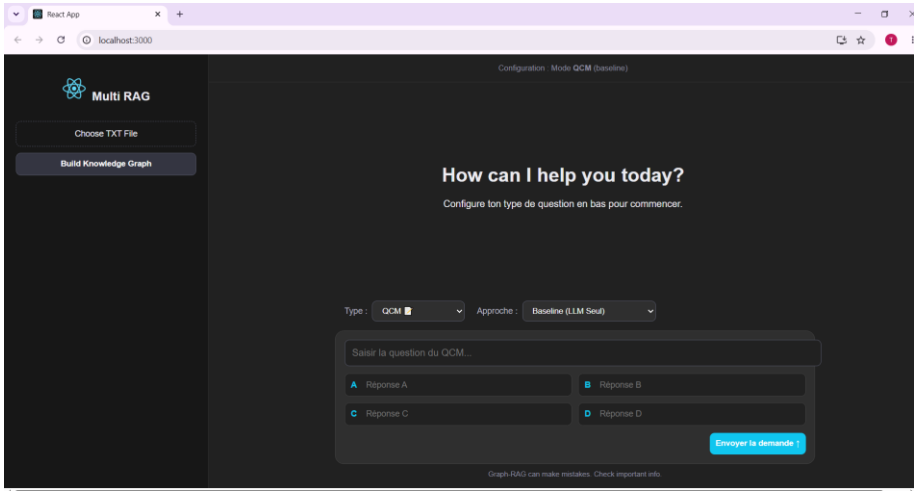

Figure 15 – Main interface of the Multi RAG application (MCQ mode, Baseline approach. Source: Author's own screenshot.

4.4.2 Left Sidebar — Document Management

The sidebar allows the user to load a text file via the *Choose TXT File* button and trigger graph construction via *Build Knowledge Graph*. Once the graph is built, its PNG visualization is displayed directly in the sidebar under the *Live Knowledge Graph* section, and a *Download PNG* button enables local export.

4.4.3 Central Chat Area — Question-Answer Interaction

The chat area displays the conversation history (user messages on the right, AI responses on the left) and a dynamic input form that adapts to the selected mode. Two selectors allow configuration:

— **Type selector**: switches between MCQ mode (showing a 2×2 grid of options A/B/C/D) and Free Question mode (single input field only).

— **Approach selector**: options available vary depending on the selected type — Baseline, Graph, or GNN for MCQ; GNN-RAG for Free Question.

The header bar at the top of the chat area displays the current configuration (e.g., *Configuration: Mode QCM (baseline)*) for constant reference.

4.5 User Interaction Scenarios

Table 12 summarizes the four interaction modes available in the deployed application.

Table 12 – Available interaction modes in the deployed system

Mode	Approach	External Context	LLM Calls/Q
MCQ	Baseline (LLM only)	None	1
MCQ	Graph (ConceptNet)	ConceptNet relations	2
MCQ	GNN (Embeddings)	GNN embeddings	—
Free Question	GNN-RAG	Doc graph / QA JSON	1

4.5.1 Scenario 1 — MCQ with Baseline Approach (LLM Only)

The user selects the MCQ type and Baseline approach, fills in the question and its four options, then clicks *Envoyer la demande*. The backend invokes the Mistral LLM directly with no external context. The model returns the letter of the correct answer (A, B, C, or D). Figure **??** shows an example where the system correctly identifies answer **C** (*art supplies*) for the question: *"If it is not used for hair, a round brush is an example of what?"*

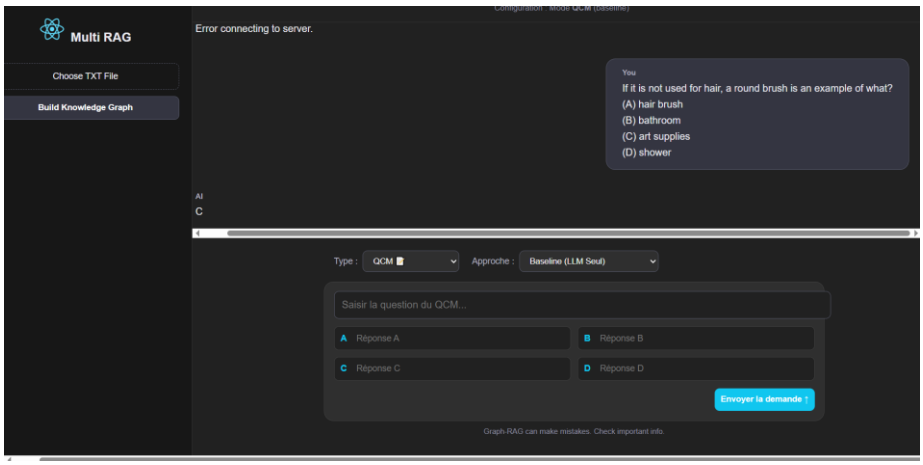

Figure 16 – MCQ Baseline result: answer C returned by the Mistral LLM Source: Author's own screenshot.

4.5.2 Scenario 2 — MCQ with Graph Approach(ConceptNet)

In this scenario, the backend extracts entities from the question using spaCy (POS tagging on NOUN/PROPN tokens, lemmatization, stop word filtering), then queries the local ConceptNet SQLite database to retrieve up to 8 semantic relations per entity. The retrieved context is injected into the LLM prompt before generating the final answer. If no ConceptNet context is found, the system automatically falls back to the Baseline approach.

Figure **??** shows the interface with the Graph (ConceptNet) approach selected, and Figure **??** presents the corresponding backend logs showing the extracted entities (['hair', 'brush']) and the ConceptNet relations retrieved.

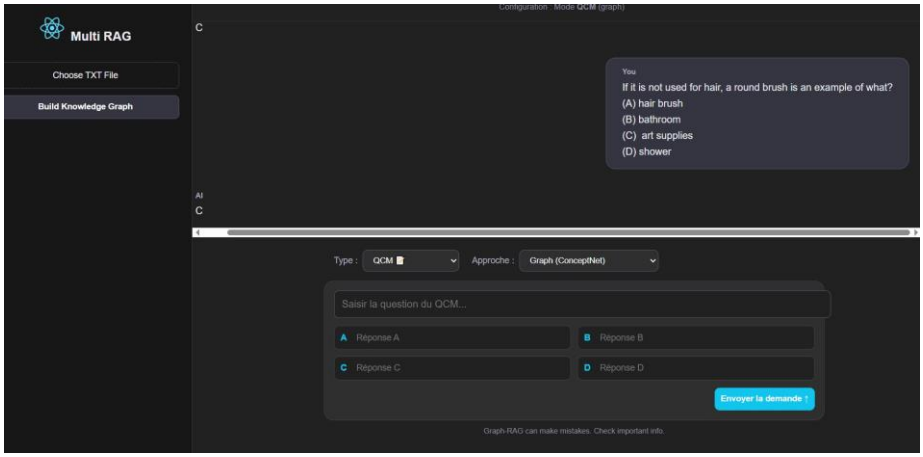

Figure 17 – MCQ interface with Graph (ConceptNet) approach selected. Source: Author's own screenshot.

```
INFO:     127.0.0.1:59642 - "POST /chat HTTP/1.1" 200 OK
06/12/2026 03:54:55 - INFO -     HTTP Request: POST http://127.0.0.1:11434/api/generate "HTTP/1.1 200 OK"
INFO:     127.0.0.1:59789 - "POST /chat HTTP/1.1" 200 OK
🔍 Entités: ['hair', 'brush']
📜 Contexte: brush is related to hair
brush RelatedTo hair
brush Synonym brush
brush is used for brush
You are likely to find a hair in a brush
06/12/2026 03:55:43 - INFO -     HTTP Request: POST http://127.0.0.1:11434/api/generate "HTTP/1.1 200 OK"
INFO:     127.0.0.1:59801 - "POST /chat HTTP/1.1" 200 OK
```

Figure 18 – Backend logs: extracted entities ['hair', 'brush'] and retrieved ConceptNet relations. Source: Author's own screenshot.

4.5.3 Scenario 3 — MCQ with GNN Reasoning Approach

The GNN approach for MCQ mode leverages the embeddings produced by the Graph Neural Network to enrich the reasoning process over the struc-

tured knowledge graph. In the current deployment, this approach serves as a placeholder, with full integration representing a future work direction. It would combine the strengths of the GNN-RAG pipeline (Scenario 4) within the structured multiple-choice framework, replacing ConceptNet retrieval with learned graph message passing.

4.5.4 Scenario 4 — Free Question with GNN-RAG (Without Document)

The user selects the Free Question type and GNN-RAG approach, then enters a question in natural language without uploading any document. The backend builds the question graph via the NLP pipeline, computes its GNN embeddings, and performs a cosine similarity search over the pre-loaded QA pairs in memory. Pairs with a similarity score above 0.45 are used as context for response generation. Figure 19 shows a natural language response obtained for the same question as in the previous scenarios.

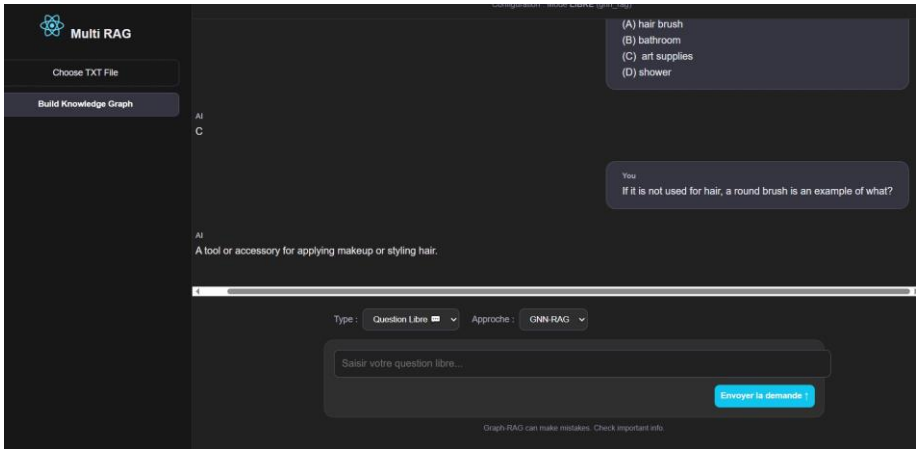

Figure 19 – Free Question mode with GNN-RAG: natural language response generated without an uploaded document. Source: Author's own screenshot.

4.5.5 Scenario 5 — Free Question with GNN-RAG (With Uploaded Document)

The user first uploads a text file via the sidebar, triggering document graph construction. The graph is displayed in real time in the *Live Knowledge Graph* section of the sidebar. When a question is submitted, the backend performs the similarity search directly within this document graph rather than the generic QA base. Figure 20 illustrates this scenario with the file `test.txt` uploaded, showing the knowledge graph in the sidebar and two successive exchanges in the chat area.

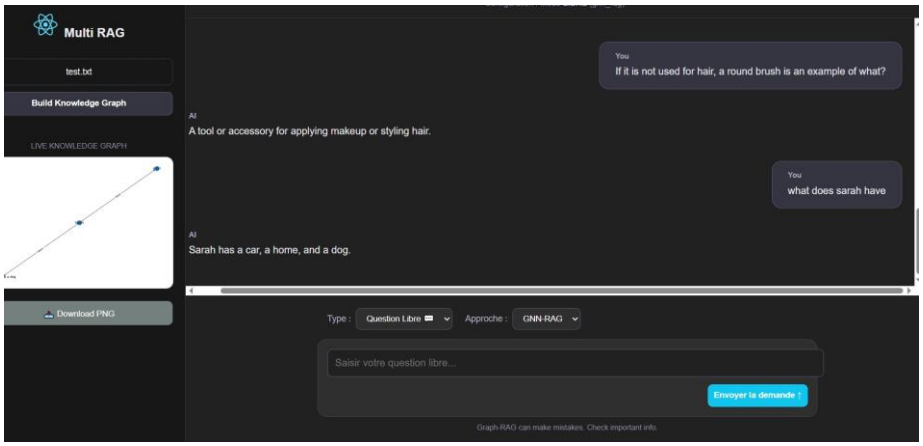

Figure 20 – Free Question mode with GNN-RAG and an uploaded document. Source: Author's own screenshot.

4.6 Justification of Deployment Technology Choices

Several technical decisions guided the deployment architecture:

— **FastAPI** was chosen over Flask or Django for its native asynchronous support, automatic OpenAPI documentation generation, and Pydantic-based input validation.

— **ConceptNet as local SQLite** avoids any dependency on external services and guarantees constant query response times regardless of network conditions.

— **Mistral via Ollama** runs entirely on the local machine, ensuring data privacy and eliminating the latency and cost associated with cloud-based LLM APIs.

— **Frontend/backend separation** via a REST API allows independent evolution of both components and simplifies unit testing of each layer.

— **CORS middleware** in FastAPI enables communication between the React frontend (port 3000) and the backend (port 8000) in the local development environment.

4.7 Conclusion

This chapter presented the deployment architecture of the Multi RAG local web application. The system is structured around a FastAPI RESTful API and a React.js interface accessible locally via a browser. Five interaction scenarios are available: the MCQ mode with three approaches (Baseline, Graph-ConceptNet, and GNN), and the Free Question mode via GNN-RAG with or without a contextual document. The screenshots presented throughout this chapter concretely

illustrate each usage scenario and confirm the correct functioning of the complete deployment pipeline.

General Conclusion

This work presented the design, evaluation, and deployment of a Multi-RAG system combining large language models with structured knowledge retrieval and graph-based reasoning for question answering. The project addressed the fundamental limitations of standard LLMs—namely hallucinations, static knowledge, and limited relational reasoning—by integrating external knowledge sources and graph neural network architectures into the generation pipeline.

The theoretical foundations established in Chapter 1 provided the necessary background on language model architectures, pretraining paradigms, and adaptation strategies, laying the groundwork for understanding both the capabilities and the limitations of modern LLMs. Chapter 2 extended this analysis to the state of the art in Retrieval-Augmented Generation, covering the progression from Naive RAG to Modular and Graph-based RAG systems, and highlighting the role of Graph Neural Networks in enhancing structured knowledge reasoning.

The experimental evaluation presented in Chapter 3 compared four distinct approaches on the OpenBookQA dataset.

Overall, this project demonstrates that the combination of large language models, structured knowledge retrieval, and graph neural networks represents a promising research direction for building more accurate, interpretable, and knowledge-grounded question answering systems. The deployed Multi RAG application constitutes a concrete and extensible foundation for future research in this area.

References

[Alammar and Grootendorst, 2024] Alammar, J. and Grootendorst, M. (2024). *Hands-On Large Language Models*. O'Reilly Media, Inc., Sebastopol, CA, 1 edition. First Edition, First Release: 2024-09-10.

[Alansari and Luqman, 2026] Alansari, A. and Luqman, H. (2026). Large language models hallucination: A comprehensive survey. *Computer Science Review*, 61:100970.

[Baek et al., 2023] Baek, J., Aji, A. F., and Saffari, A. (2023). Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. In *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, pages 78–106.

[Gao et al., 2023] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Guo, Q., Wang, M., and Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

[Han et al., 2024] Han, H., Wang, Y., Shomer, H., Guo, K., Ding, J., Lei, Y., and Tang, J. (2024). Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*.

[Hogan et al., 2021] Hogan, A., Blomqvist, E., Cochez, M., D'Amato, C., de Melo, G., Gutierrez, C., Kirrane, S., Labra Gayo, J. E., Navigli, R., Neumaier, S., Ngonga Ngomo, A.-C., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., and Zimmermann, A. (2021). Knowledge graphs. *ACM Computing Surveys*, 54(4):1–37.

[Lewis et al., 2020] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

[Speer et al., 2017] Speer, R., Chin, J., and Havasi, C. (2017). Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

[Touvron et al., 2023] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozi`ere, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł ., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

[Yasunaga et al., 2021] Yasunaga, M., Ren, H., Bosselut, A., Liang, P., and Leskovec, J. (2021). Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546.

[Zhou et al., 2020] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.