PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY MUSTAPHA STAMBOULI OF MASCARA

Faculty of Exact Sciences

Department of Computer Science

Dissertation

Submitted in partial fulfilment of the requirements for Master degree in Computer Science

Option: Artificial Intelligence

Theme

Enhancing Legal Information Access and Reasoning with Retrieval-Augmented LLMs for Juridical Data

Presented by **Boudjenane Zoubida Asmaa**

Jury:

President	Aaaaa AAAAA	Professor	University of Mascara
Director	Mohammed SALEM	Professor	University of Mascara
Examiner	Ddddd DDDDD	Professor	University of Mascara
Examiner	Eeeee EEEEE	Associate Prof.	University of Mascara
Examiner	Fffff FFFFF	Associate Prof.	University of Mascara

Month 2025

Acknowledgements

Dedication

ملخص

ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص
ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص
ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص
ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص
ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص
ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص
ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص
ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص ملخص
ملخص .

كلمات مفتاحية: ملخص، ملخص، ملخص، ملخص، ملخص.

Abstract

Abstract Abstract.

Key words: *Abstract, Abstract, Abstract, Abstract, Abstract.*

Contents

List of Figures

List of Tables

List of Abbreviations

BERT:	Bidirectional Encoder Representations from Transformers
RAG :	Retrieval-Augmented Generation
FFN:	Feed-Forward Network
GPT:	Generative Pre-trained Transformer
LLM:	Large Language Model
LSTM:	Long Short-Term Memory
MT:	Machine Translation
NLG:	Natural Language Generation
NLP:	Natural Language Processing
NLU:	Natural Language Understanding
RNN:	Recurrent Neural Network
Seq2Seq:	Sequence to Sequence
T5:	Text-to-Text Transfer Transformer

Chapter 1

Large Language Models (LLMs)

1.1 Introduction

Large Language Models (LLMs) represent a transformative leap in Natural Language Processing (NLP), allowing machines to perform complex language tasks with remarkable accuracy. From generating coherent text to answering nuanced questions, LLMs have pushed the boundaries of what AI can achieve. This chapter delves into the architecture and principles behind LLMs, their development through scaling and training techniques, and the models that have defined this field. It also addresses the challenges these models face and their expanding role in various real-world applications.

1.2 Natural language processing (NLP)

Natural Language Processing (NLP) is a field within artificial intelligence (AI) that concentrates on the interaction between computers and human language. It involves the development of algorithms and models that allow machines to comprehend, interpret, and generate human language in a meaningful context.NLP is crucial for enabling computers to process and respond to human language effectively, as demonstrated by features like Google's predictive text in keyboards and language translation systems that manage multiple languages efficiently [5]

1.2.1 Historical development of NLP

The figure below presents a timeline outlining the key developments in Natural Language Processing (NLP) beginning with rule-based approaches in the 1950s and progressing through the rise of statistical methods and early neural networks in the late 1980s. It then highlights the impact of deep learning from the 2000s onward, leading to the development of pre-trained models like BERT and GPT. The timeline concludes with the emergence of large language models (LLMs) from 2019 to the present, marking a significant shift in NLP research and applications.

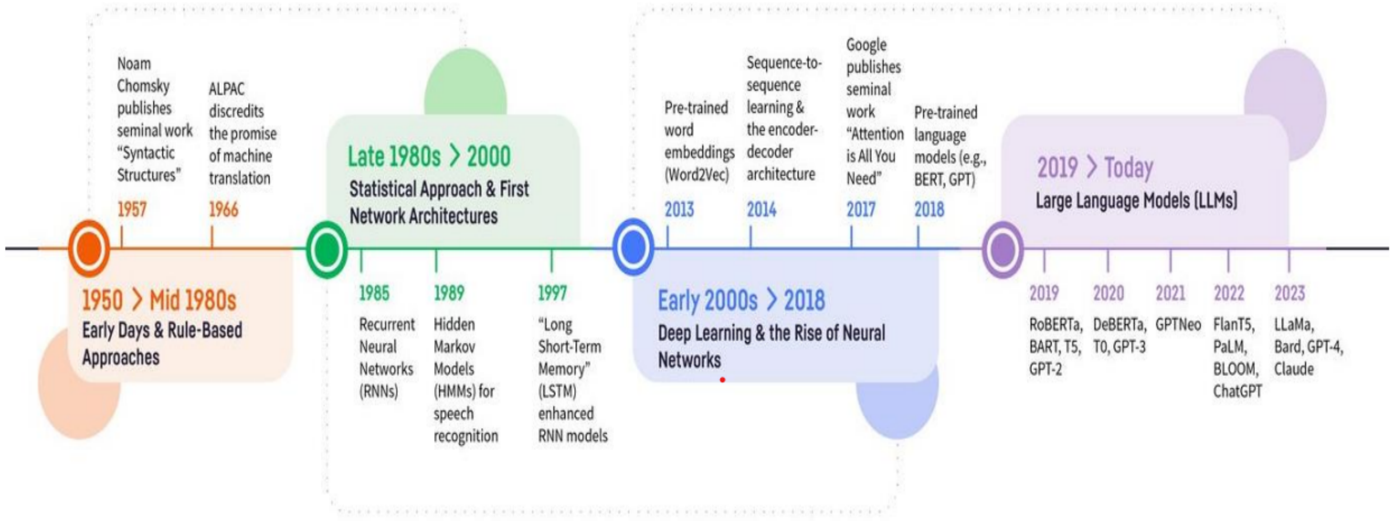

Noam Chomsky publishes seminal work "Syntactic Structures"
1957
ALPAC discredits the promise of machine translation
1966
1950 > Mid 1980s
Early Days & Rule-Based Approaches
Late 1980s > 2000
Statistical Approach & First Network Architectures
1985
Recurrent Neural Networks (RNNs)
1989
Hidden Markov Models (HMMs) for speech recognition
1997
"Long Short-Term Memory" (LSTM) enhanced RNN models
Pre-trained word embeddings (Word2Vec)
2013
Sequence-to-sequence learning & the encoder-decoder architecture
2014
Google publishes seminal work "Attention is All You Need"
2017
Pre-trained language models (e.g., BERT, GPT)
2018
Early 2000s > 2018
Deep Learning & the Rise of Neural Networks
2019 > Today
Large Language Models (LLMs)
2019
RoBERTa, BART, T5, GPT-2
2020
DeBERTa, T0, GPT-3
2021
GPTNeo
2022
FlanT5, PaLM, BLOOM, ChatGPT
2023
LLaMa, Bard, GPT-4, Claude

Figure 1.1: The Vauquois triangle, illustrating the foundations of machine translation.

1.2.2 Importance of natural language processing

1. **Efficient Data Processing:** NLP enables businesses to process and analyze large volumes of unstructured, text-heavy data that would be challenging to handle otherwise..

2. **Understanding Human Language:**NLP can interpret complex language elements, such as acronyms, abbreviations, and context, improving the accuracy of machine learning models..

3. **Advancements in Technology:**Improvements in deep learning and machine learning have made NLP more effective, expanding the types of data that can be analyzed.

4. **Natural Interactions:** NLP allows users to interact naturally with AI chatbots and voice assistants, like Siri, without needing to use specific,predefined[6].

1.3 Neural Networks in NLP

A neural network, or artificial neural network, is a machine learning algorithm inspired by the human brain. It is a key component of deep learning, a branch of machine learning effective in solving complex problems like image recognition and language processing. Unlike traditional computer programs that use a step-by-step algorithmic approach, neural networks learn from examples, mimicking the way neurons in the human brain operate. They consist of interconnected nodes (processing elements) that work together in parallel to solve specific problems.

A neural network has three basic sections, or parts, each composed of "nodes."

The input layer is the first part, receiving raw data, with each node (or neuron) representing a feature of this data. The hidden layers, which form the intermediate section, perform various transformations and computations, enabling the network to learn complex patterns and relationships. Finally, the output layer, which is the last part, produces the network's output, with the number of nodes corresponding to the desired output classes or regression values [5].

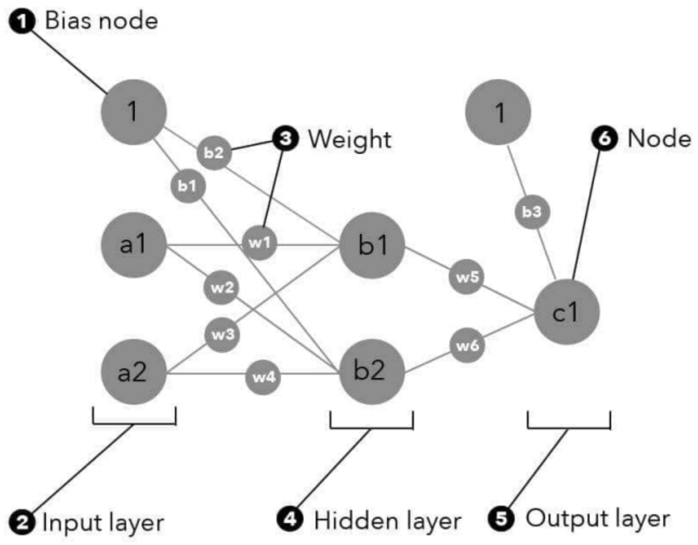


Figure 1.2: Artificial Neural Network (ANN).

1.3.1 Basics Concepts in Neural Networks

Weights

Variables on the edges between nodes that are multiplied with node outputs to form the input for the next layer.Weights are crucial for training and tuning a neural network and are often initialized within a range of -1 to 1.

Biases

Additional nodes in hidden and output layers that connect to every node within their respective layers but not to the previous layer. Biases add a constant value (typically 1 or -1) to the input of a layer, helping shift the activation function and aiding in effective learning.

Activation Functions

These functions introduce non-linearity to the network, enabling it to learn and model complex data patterns. Common activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). [7].

1.3.2 Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) are a type of neural network designed for processing sequential data, such as text or time series, by effectively handling variable-length inputs. An RNN consists of a hidden state h and an optical output y, which operate on a variable sequence x=$(x_1....x_t)$At each time step t, the hidden state h^t of the RNN is updated according to the function $h^{(t)} = f(h^{(t-1)}, x_t)$ where f is a non-linear activation function, such as a logistic sigmoid or a more complex long short-term memory (LSTM) unit. This design allows RNNs to remember past inputs and incorporate them into future outputs, enabling them to recognize patterns that occur at multiple positions within a sequence. Additionally, RNNs utilize parameter sharing across time steps, which helps in generalizing across sequences of varying lengths and learning complex dependencies over time. Despite challenges with long-term dependencies, RNNs are particularly effective for tasks requiring context or sequential understanding, making them valuable for various applications, including time series analysis and natural language processing[8]

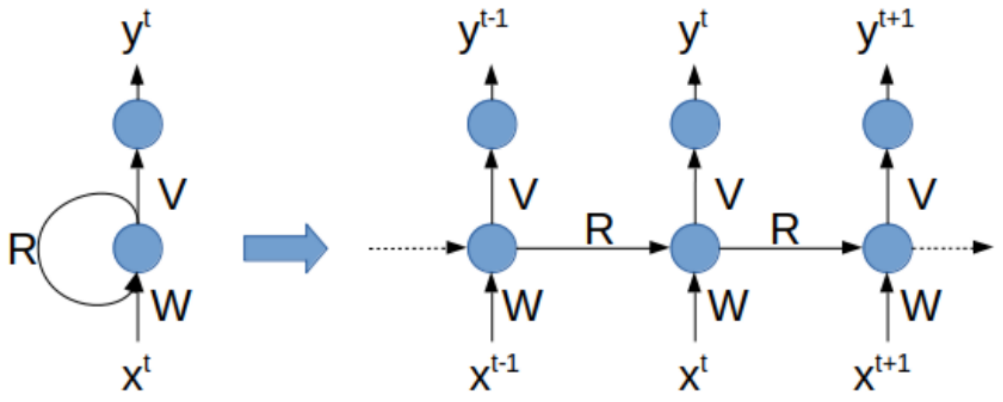


Figure 1.3: RNN Architecture.

1.3.3 Long-Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks, designed by Hochreiter and Schmidhuber, are an improved version of recurrent neural networks (RNNs) designed to learn long-term dependencies in sequential data, making them suitable for tasks like time series forecasting and language translation They address the limitations of traditional RNNs by introducing a memory cell that maintains information over extended periods. This memory cell is regulated by three gates—input, forget, and output gates—which control the flow of information in and out of the cell as shown in the fig [9] Forget Gate: Decides what information to discard from

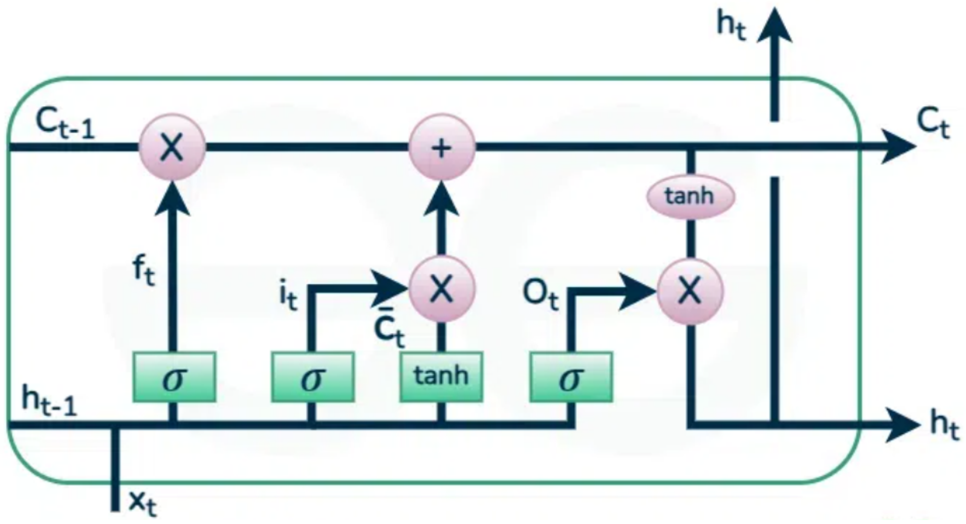


Figure 1.4: LSTM Architecture.

the cell state

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Input Gate: Determines what new information to add to the cell state

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$

Output Gate: Controls what information to output from the cell state

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$

$$h_t = o_t \odot \tanh(C_t)$$

It is worth mentioning that Bidirectional LSTM networks extend the traditional LSTM architecture by processing data in both forward and backward directions. This approach enables the network to capture dependencies from both past and future contexts, improving its ability to resolve temporal dependencies. Bidirectional LSTMs are particularly effective at handling multidimensional problems, encapsulating spatially and temporally distributed information, and dealing with incomplete data through flexible connection mechanisms [10].

1.4 Transformer Architecture

The Transformer architecture is a deep learning model introduced in June 2017 by Vaswani et al. from Google Brain. Their paper, titled "Attention Is All You Need," presented a groundbreaking approach to processing sequential data through the use of a self-attention mechanism. This innovative method allows the model to assign different levels of importance to various parts of the input, enabling it to capture long-range dependencies much more effectively than earlier models like RNNs and LSTMs.The original Transformer model is structured as a stack of six layers, where the output of each layer i serves as the input to the subsequent layer i+1, continuing this process until the final prediction is reached.It features a six-layer encoder on the left and a corresponding six-layer decoder on the right, both of which work together to transform input sequences into meaningful outputs. Each encoder and decoder consists of six identical layers that allow the model to process and generate language efficiently [11].

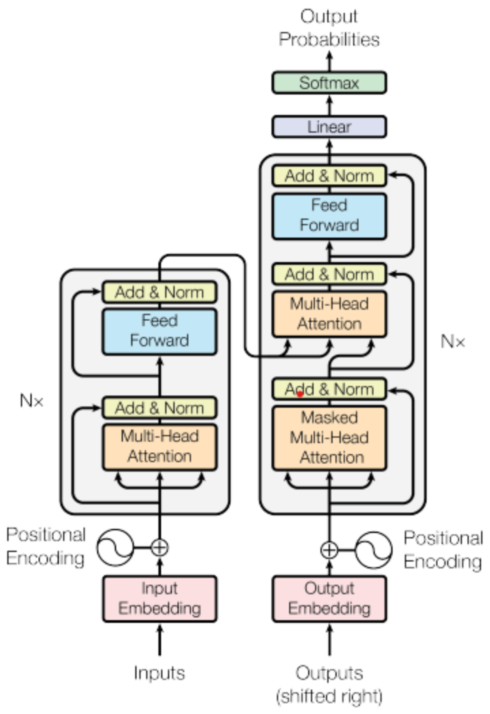


Figure 1.5: The Transformer - model architecture..

1.4.1 Encoder and Decoder Stacks

Encoder: Each layer in the encoder consists of two sub-layers:
Multi-head self-attention mechanism: This allows the model to focus on different parts of the input sequence.
Position-wise feed-forward network: A fully connected network applied to each position separately and identically.
Residual Connections: Residual connections are applied around each sub-layer, followed by layer normalization. The output of each sub-layer is computed as Layer Norm(x + Sub-layer(x)).
Output Dimension: All sub-layers and embedding layers output vectors of dimension d-model = 512.
Decoder: Similar to the encoder, the decoder also has 6 identical layers, each containing the same two sub-layers: Multi-head self-attention mechanism , Position-wise feed-forward network.
Additional Sub-layer: The decoder includes a third sub-layer that performs multi-head attention over the output of the encoder stack.
Residual Connections and Normalization: Like the encoder, residual connections are applied around each sub-layer, followed by layer normalization.
Masked Self-Attention: The self-attention mechanism in the decoder is modified to prevent positions from attending to subsequent positions, ensuring that predictions for position i depend only on the known outputs at positions before i [12].

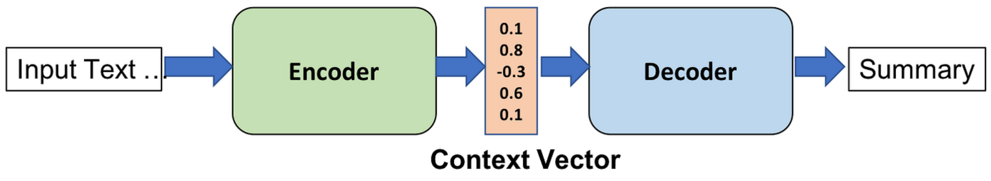


Figure 1.6: Encoder and Decoder Stacks

1.4.2 Self-Attention Mechanism

Self-attention is a mechanism that allows the model to weigh the importance of different words in a sequence relative to each other. This is crucial for understanding context and relationships between words [12].

1.4.3 Query, Key, and Value Vectors:

Query Vectors (Q): Generated from the input word's embedding and represent the word's role in finding relevant information in other words.
Key Vectors (K): Derived similarly from embedding and serve as a comparison standard for each word.
Value Vectors (V): Contain the actual information to be passed on and are weighted according to relevance determined by the Query and Key comparison [12]..

1.4.4 Self-Attention Calculation:

The score is computed by taking the dot product of the Query vector of a word with the Key vectors of all other words. This score determines how much focus should be placed on each word when encoding the current word. The scores are scaled and passed through a softmax function, producing a distribution of attention weights.
Each Value vector is weighted by these attention scores and summed to produce the final output of the self-attention layer[12].

1.4.5 Role of Multi-Head Attention

Multi-head attention enhances the model's ability to focus on different aspects of the input by creating multiple sets of Query, Key, and Value vectors. Each set is processed independently and then combined.
This approach allows the model to attend to information from different representation subspaces at different positions, improving its ability to capture complex relationships[12].

1.4.6 Positional Encoding:

Since self-attention mechanisms do not inherently capture the order of words, positional encoding is added to the word embeddings to provide information about the relative position of each word in the sequence. These encodings are added directly to the input embeddings, allowing the model to consider both the position and the content of each word during processing[13].

1.4.7 Transformers in NLP

Transformers have revolutionized the field of Natural Language Processing (NLP) by introducing a powerful and efficient way to handle textual data. This advancement has led to the creation of highly effective models like BERT and GPT, with its deep bidirectional context, excels at understanding and improving performance across various NLP tasks. GPT, on the other hand, is renowned for generating coherent and contextually relevant text, significantly advancing applications such as text generation and translation[14].

Scalability: Transformers efficiently handle large datasets and long sequences, overcoming the limitations of RNNs. This scalability allows for training with billions of parameters, enhancing model capabilities.

Rich Contextual Understanding: The self-attention mechanism in transformers captures relationships between words across the entire sequence, enabling deep contextual understanding and more accurate language processing.

Model Efficiency: Transformers enable parallel processing, which speeds up training and makes them more efficient than sequential models like RNNs. This efficiency supports the rapid development and deployment of advanced language models[12].

1.5 Emergence of Large Language Models (LLMs)

Large Language Models (LLMs) are advanced artificial intelligence systems designed to process and generate human-like text. They are typically based on transformer architectures and are characterized by their enormous scale, with billions of parameters. LLMs are pre-trained on vast amounts of text data in a self-supervised manner, enabling them to develop a broad understanding of language. They are capable of performing a wide range of tasks with minimal task-specific fine-tuning, often achieving significant performance improvements through few-shot or zero-shot learning[15].

1.5.1 Scaling in Large Language Models (LLMs)

Scaling is crucial in the evolution of large language models. The history of scaling shows that increasing both model size and dataset size leads to significant improvements in performance across various NLP tasks. For instance, early work by Brants et al. (2007) demonstrated the benefits of using language models trained on vast datasets, such as 2 trillion tokens, which led to significant advancements in machine translation quality. This was followed by efforts like those of Heafield et al. (2013), who scaled traditional models to Web-scale data, and Jozefowicz et al. (2016), who scaled LSTMs to 1 billion parameters, achieving state-of-the-art results on large benchmarks.The advent of transformer-based models marked a significant shift.Models like BERT, GPT-2, and GPT-3, with their enormous parameter counts—up to 175 billion for GPT-3—demonstrated that scaling up not only the model but also the dataset size yields substantial gains in performance. Researchers like Kaplan et al. (2020)

and Hoffmann et al. (2022) studied how scaling affects model performance, proposing power laws that show a predictable relationship between model size, dataset size, and performance. These studies emphasized the importance of scaling for the continued progress of LLMs[16].

1.5.2 Pre-training

Pre-training is a crucial stage in developing Large Language Models (LLMs), where the model learns from extensive unlabeled datasets through a process called self-supervision. This stage allows the model to recognize and internalize a wide range of linguistic patterns, laying the groundwork for fine-tuning on specific tasks.
Several pre-training objectives have been implemented to maximize the effectiveness of this learning process, each offering distinct benefits to the model's performance **Full Language Modeling:** Used since GPT-2, this approach trains decoder-only models to predict the next token in a sequence based on previous tokens. This autoregressive method enables models like GPT-3 to generate coherent and contextually relevant text.
Prefix Language Modeling: Employed in encoder-decoder and non-causal decoder-only models, this technique uses a non-causal (considering both past and future tokens) prefix for predicting subsequent tokens, offering more flexibility and enhancing the model's adaptability across various language tasks.
Masked Language Modeling: Popularized by BERT, this method involves masking certain tokens in the input text and training the model to predict them, helping the model understand word context. An extension, span corruption, masks entire text spans for prediction, further improving contextual comprehension[17].

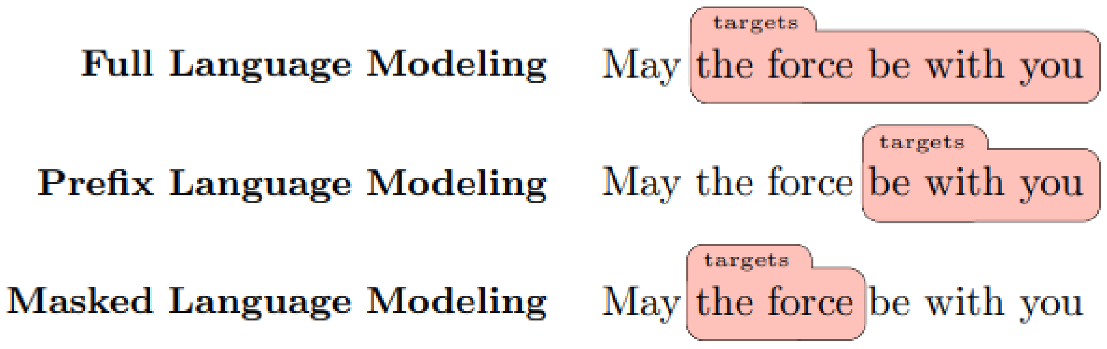


Figure 1.7: Training Tokenization in Full, Prefix, and Masked Language Modeling.

1.5.3 Fine-tuning

Fine-tuning has been the most common approach for adapting LLMs to specific tasks. After pre-training on large datasets, these models are further trained on smaller, supervised datasets tailored to the desired task. Fine-tuning updates the weights of the pre-trained model, allowing it to excel at specific tasks like sentiment analysis, machine translation, or question answering. The main advantage of fine-tuning is that it often results in strong performance on task-specific benchmarks. However, it comes with challenges, such as the need for a new large dataset for every task, the risk of overfitting to narrow distributions, and potential poor generalization to out-of-distribution examples[16].

1.5.4 Few-Shot, One-Shot, and Zero-Shot Learning

Few-shot, one-shot, and zero-shot learning represent different paradigms of using LLMs without extensive fine-tuning. These methods involve using pre-trained models to perform tasks with minimal task-specific data.
Few-Shot Learning involves giving the model a few examples (typically 10-100) of the task during inference. This approach reduces the need for large, task-specific datasets and limits the risk of overfitting to narrow data distributions. However, few-shot results are generally not as strong as those from fully fine-tuned models.
One-Shot Learning is similar to few-shot learning but uses only a single example alongside a natural language description of the task. This method mirrors how humans are often instructed for tasks, making it an interesting approach for tasks where providing multiple examples is impractical.
Zero-Shot Learning requires the model to perform a task based solely on a natural language description, with no examples provided. While this method offers maximum flexibility and robustness, it is also the most challenging for models. Zero-shot performance is often weaker than few-shot or one-shot, but it represents a significant step towards task-agnostic AI, where models can generalize across a wide range of tasks with minimal human intervention[15].

1.5.5 Evaluation Datasets and Tasks

Evaluating Large Language Models (LLMs) is essential for determining their effectiveness and limitations in comprehending and generating human language. This evaluation typically falls into two primary categories:

- Natural Language Understanding (NLU):This measures the model's proficiency in comprehending language, encompassing tasks such as sentiment analysis, text classification, natural language inference (NLI), question answering (QA), commonsense reasoning (CR), mathematical reasoning (MR), and reading comprehension (RC).

- Natural Language Generation (NLG):This assesses the model's capability to produce text based on given context. It includes tasks like summarization, sentence completion, machine translation (MT), and dialogue generation.

Benchmarks play a critical role in evaluating LLMs, providing standardized tests to measure their performance across various tasks:

- MMLU: Measures model knowledge from pretraining and evaluates performance in zero-shot and few-shot scenarios across 57 subjects, testing world knowledge and problem-solving abilities.
- SuperGLUE: An advanced benchmark that builds on GLUE, assessing tasks like question answering and natural language inference. It is designed to test deeper aspects of language understanding and requires significant advancements in various learning methodologies.
- BIG-bench: A large-scale benchmark for evaluating LLMs across diverse tasks including reasoning, creativity, ethics, and domain-specific knowledge.
- GLUE: A foundational benchmark for evaluating and analyzing natural language understanding, offering a range of resources for model assessment[18].

1.6 Popular Models and Datasets

Large Language Models (LLMs) like GPT-3, GPT-4, and BERT have revolutionized NLP by leveraging vast datasets such as Common Crawl and WebText. These datasets provide a diverse linguistic foundation, enabling models to perform a wide range of tasks with remarkable accuracy and contextual understanding.

1.6.1 GPT-N Models

GPT models are advanced autoregressive language models that generate substantial and complex machine-produced text from minimal input. They leverage deep learning techniques to mimic human text generation by predicting the current value based on preceding values. NLP models initially struggled with tasks outside their training sets due to data restrictions. OpenAI addressed this with GPT-1,introduced in 2018.

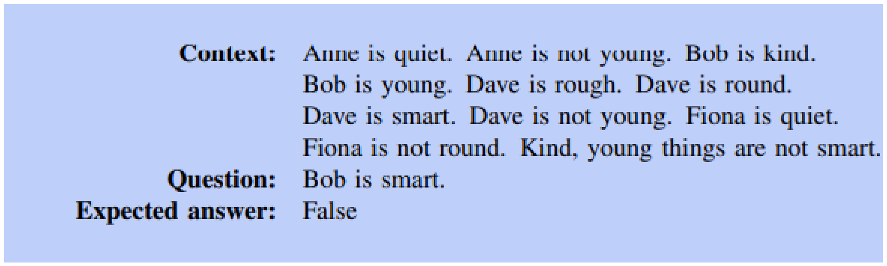
- GPT-1, trained on the BooksCorpus dataset, utilized a 12-layer transformer decoder with self-attention. Its pre-training allowed for zero-shot performance on various tasks, demonstrating the potential of generative language models.
- In 2019, GPT-2 improved upon GPT-1 by using a larger dataset and 1.5 billion parameters (compared to GPT-1's 117 million). It excelled in tasks like translation and summarization, enhancing accuracy in recognizing long-distance relationships.
- GPT-3, released later, featured around 175 billion parameters and was trained on the Common Crawl dataset. It could generate human-like text, perform basic math, and write code. Despite its capabilities, its size and cost made it challenging to implement.
- GPT-4, launched in March 2023, advanced further with multimodal capabilities and context windows of up to 32,768 tokens. It incorporates reinforcement learning for better alignment with human input and policy[19]

1.6.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a groundbreaking language representation model that pretrains deep bidirectional representations from unlabeled text by jointly conditioning on both left and right contexts in all layers. This bidirectional approach enables BERT to be fine-tuned with just a single additional output layer for various tasks, such as question answering and language inference, without the need for extensive architectural modifications. BERT has achieved remarkable performance, setting new state-of-the-art results across eleven NLP tasks, including a GLUE score of 80.5%, MultiNLI accuracy of 86.7%, SQuAD v1.1 F1 score of 93.2%, and SQuAD v2.0 F1 score of 83.1%. Its combination of conceptual simplicity and empirical effectiveness makes BERT a powerful and versatile tool for a wide range of natural language processing applications[14].

1.6.3 T5

The T5 (Text-To-Text Transfer Transformer) model represents a unified framework for natural language processing (NLP), designed to address various text processing tasks by treating them as "text-to-text" problems. This approach involves converting all tasks into a format where both input and output are text, allowing the same model, objective, and training procedure to be applied across different tasks. T5 leverages extensive pre-training on a large, unlabeled dataset, enabling it to develop general-purpose knowledge that enhances its performance on a range of NLP tasks, such as question answering, document summarization, and sentiment classification[20].



Context:	Anne is quiet. Anne is not young. Bob is kind. Bob is young. Dave is rough. Dave is round. Dave is smart. Dave is not young. Fiona is quiet. Fiona is not round. Kind, young things are not smart.
Question:	Bob is smart.
Expected answer:	False

Figure 1.8: exammple of RuleTaker dataset

1.6.4 LLAMA2

1.6.5 WebText

WebText is a dataset developed by OpenAI to encapsulate a wide variety of high-quality web content. It comprises text sourced from outbound links on popular websites like Reddit, chosen based on their popularity and the quality of their content. Unlike Common Crawl, which indiscriminately includes a broad range of web pages, WebText is more selective, focusing on high-quality, engaging material. Although smaller in size compared to Common Crawl, WebText is curated to ensure the text reflects diverse, well-written content, making it particularly valuable for training language models to produce coherent and contextually appropriate text[21].

1.6.6 RuleTaker

RuleTaker is a collection of datasets designed to assess the deductive reasoning capabilities of language models. Each dataset includes a set of facts, rules, and a boolean question that the model must use logical reasoning to answer. The datasets feature synthetic subsets requiring various levels of reasoning complexity, with different numbers of deduction steps necessary to reach an answer. Among the datasets are the Bird dataset, which illustrates McCarthy's abnormality problem , the Electricity dataset, which models appliance functions, and the ParaRules corpus, where sentences like "Bob is cold" are paraphrased to "In the snow sits Bob, crying from being cold." The collection comprises 580,000 training examples, 84,000 validation examples, and 173,000 testing examples[22].

1.7 Challenges and Limitations

LLMs have greatly advanced NLP, but they also present challenges such as high computational costs, issues with adversarial robustness, and difficulties in interpretability. As these models scale, they encounter new challenges in scalability, privacy, and real-time processing.

Ongoing research is exploring multi-modality, transfer learning, and continuous learning, which bring additional complexities and highlight the evolving impact of LLMs in practical applications[18].

1.7.1 Computational Cost

Training large language models (LLMs) demands significant computational resources, leading to higher production costs and environmental concerns due to the substantial energy used in large-scale training. Although enhancing computational resources can boost performance, the gains diminish over time when the size of the model and dataset stay constant, adhering to the power law of diminishing returns.

1.7.2 Overfitting

Despite their advanced learning abilities, they can overfit to noisy or unusual patterns in their large training datasets, which may result in generating nonsensical responses. The ongoing discussion about Memorization versus Generalization in LLMs revolves around finding an optimal balance. Memorization helps the model retain specific details from its training, allowing it to answer precise questions accurately. On the other hand, generalization enables the model to make predictions and generate responses for new, unseen inputs, which is crucial for handling diverse real-world tasks. The challenge is to find the right balance: excessive memorization can lead to overfitting, reducing the model's flexibility and ability to handle novel inputs.

1.7.3 Interpretability and Explainability

The "black-box" nature of LLMs makes it difficult to understand how they make decisions, which is vital for gaining broader acceptance and trust, particularly in sensitive areas. Although these models have advanced capabilities, the lack of transparency into their workings limits their effectiveness and reliability. Efforts are underway to enhance the explainability of LLMs to build user trust and promote responsible use of AI. Understanding how LLMs generate their responses is crucial for ensuring they align with human values and legal standards.

1.7.4 Hallucinations

LLMs sometimes produce "hallucinations," or responses that, despite appearing plausible, are incorrect or do not match the provided information. These hallucinations can be classified into three types:

- Input-conflicting hallucination: When the model generates responses that do not align with the user's input.
- Context-conflicting hallucination: When the model produces content that contradicts information it has previously generated.
- Fact-conflicting hallucination: When the model creates responses that conflict with established knowledge.

1.7.5 Privacy Concerns

As Large Language Models (LLMs) have become more complex and sizable, privacy concerns have intensified, particularly regarding data sharing and potential misuse. Risks include the creation of harmful content, evasion of filters, and issues related to data privacy, especially in areas like e-commerce where safeguarding customer information is vital. When LLMs are trained on private data and then made publicly available, additional privacy risks arise. Since LLMs can memorize phrases from their training data, there's a danger that these phrases could be exploited by malicious actors to retrieve sensitive information, thus threatening personal privacy.

1.7.6 Real-Time Processing

Real-time processing in Large Language Models (LLMs) is crucial for many applications, particularly as mobile AI solutions become more popular and concerns about information security and privacy grow. However, LLMs typically consist of hundreds of layers and millions of parameters, which create significant challenges for real-time processing due to their high computational demands and the limited storage capacity of hardware platforms, especially in edge computing environments. Although efforts such as MobileBERT attempt to lessen memory usage, they still encounter considerable execution overhead because of the extensive number of model layers, resulting in high inference latency.

1.8 Domains of Application

The use of Large Language Models (LLMs) in various downstream tasks has become increasingly prevalent in both AI research and industry, with new applications being identified and explored regularly. These models, which excel at understanding and generating human-like text, are finding valuable applications across diverse fields.

1.8.1 Law

In the legal field, Large Language Models (LLMs) can support the analysis of legal documents by assisting with tasks such as generating initial coding for datasets, identifying key themes, and classifying information accordingly. This synergy between legal professionals and LLMs has been effective in examining legal texts, such as court opinions on theft, enhancing both the efficiency and quality of legal research. Additionally, LLMs have been tested for their capability to generate explanations of legal terms, with a focus on improving factual accuracy by integrating sentences from relevant case law. By incorporating pertinent case law, these enhanced models can produce more accurate and relevant explanations with fewer factual errors. Furthermore, LLMs can be specialized with domain-specific knowledge to tackle legal reasoning tasks and address legal queries effectively [18].

1.8.2 Cybersecurity

large Language Models (LLMs) have garnered significant attention in the field of cybersecurity. Recent research has highlighted their potential in addressing software bugs created by human developers and identifying cybersecurity threats. For example, Arora et al. have proposed methods for utilizing LLMs to evaluate cyber threats on social media through sentiment analysis. LLMs are also employed to detect cybersecurity-related information in Open Source Intelligence (OSINT), aiding in the identification of potential cyber threats. Additionally, LLMs have shown promise in detecting scams, such as phishing. Initial tests with models like GPT-3.5 and GPT-4 have demonstrated their ability to recognize common phishing indicators in emails. While LLMs exhibit considerable potential in cybersecurity, improving their reasoning abilities could enhance their effectiveness further, such as in uncovering zero-day vulnerabilities in open-source software by analyzing logic and source code[22].

1.8.3 Medicine

The integration of Large Language Models (LLMs) into medicine is transforming both healthcare delivery and research. In clinical settings, LLMs are increasingly utilized in decision support systems to offer evidence-based treatment recommendations. By analyzing patient data and medical literature, these models can assist in diagnosing conditions, suggesting relevant tests, and proposing effective treatment options. Additionally, LLMs improve patient interactions through applications like chatbots that answer questions about symptoms and medications, schedule appointments, and provide health advice.
In medical research, LLMs help sift through vast amounts of literature to extract, filter, and summarize relevant information, identify key studies, and predict future research directions.

They also play a role in medical education by generating training materials, creating exam questions, explaining complex topics, and offering personalized feedback. Furthermore, LLMs simulate patient interactions, aiding students in honing their clinical skills [18].

1.8.4 journalism

Large Language Models (LLMs) offer valuable support to journalists, especially in fact-checking and news verification. They can process and cross-reference large volumes of data with established knowledge bases. Research has demonstrated that LLMs, such as GPT-3.5, can be used to detect fake news by providing rationales that enhance other models like BERT, which can then be fine-tuned for this purpose . In addition to fact-checking, LLMs are useful for analyzing political debates, helping journalists to identify key themes, monitor how discussions evolve, and evaluate sentiments. They can also assist in detecting logical fallacies and underlying motives in political discourse and propaganda . By enhancing their reasoning capabilities, LLMs can uncover deeper insights into propaganda and misinformation, making them a powerful tool for modern journalism[22].

1.9 Conclusion

This chapter has outlined the evolution and significance of Natural Language Processing (NLP), from early neural networks to advanced Large Language Models (LLMs). We covered key concepts in neural networks and the transformative impact of the Transformer architecture. The exploration of LLMs highlighted their emergence, scaling, and the intricacies of pre-training and fine-tuning. We also discussed various challenges, including computational costs, interpretability, and privacy concerns. Finally, the chapter reviewed the applications of LLMs in fields like law, cybersecurity, medicine, and journalism, showcasing their potential and the ongoing need for further development.

Chapter 2

RAG (Retrieval-Augmented Generation)

2.1 Introduction

Large language models (LLMs) have seen significant advancements but face challenges, particularly in tasks that demand extensive knowledge or deal with queries beyond their training data. These limitations often result in inaccuracies or "hallucinations." To address this, Retrieval-Augmented Generation (RAG) supplements LLMs by retrieving relevant document chunks from external knowledge sources based on semantic similarity. By doing so, RAG helps reduce factual errors, allowing LLMs to produce more accurate content. This has led to the widespread adoption of RAG, particularly in chatbots and other real-world applications, making it a crucial technology in advancing the capabilities of LLMs.

2.2 Fundamentals of Retrieval-Augmented Generation

RAG represents a significant advancement in the capabilities of large language models (LLMs) by integrating external knowledge retrieval with the generation of text. Understanding the individual components of retrieval and generation is essential to appreciate how their synergy improves the overall performance of RAG systems.

2.2.1 Definition of RAG

Retrieval-Augmented Generation (RAG) is a technique that leverages the strengths of pre-trained large language models (LLMs) and external data sources. By merging the generative abilities of LLMs like GPT-3 or GPT-4 with the accuracy of specialized data search mechanisms, RAG systems can generate more sophisticated and contextually relevant responses[23].

2.2.2 Historical Development of RAG

The foundations of Retrieval-Augmented Generation (RAG) can be traced back to traditional information retrieval (IR) systems, Early models primarily relied on keyword matching and simple ranking mechanisms to retrieve relevant documents from databases, establishing a basic framework for information access. The landscape began to shift with the rise of neural networks in the 2010s. Notable innovations like Word2Vec and Transformer-based architectures enabled retrieval methods to incorporate deeper semantic understanding, paving the way for more sophisticated document retrieval processes. A significant advancement occurred with the introduction of dense passage retrieval (DPR) in 2020, which utilized bi-encoder architectures to map both queries and documents into dense vector spaces. The integration of these advanced retrieval techniques with large language models (LLMs) became a focal point as models like GPT-3 gained prominence. Researchers explored how LLMs could be augmented with retrieval capabilities, leading to the development of RAG systems that leverage the strengths of both retrieval and generative capabilities[24].

2.2.3 Differences Between RAG and Fine-Tuning

The enhancement of large language models (LLMs) has gained significant interest due to their increasing use. Among the various optimization strategies for LLMs, Retrieval-Augmented Generation (RAG) is often compared to fine-tuning (FT) and prompt engineering. Each method possesses unique attributes,

1. Methodological Characteristics :
   RAG is compared to providing a tailored textbook for information retrieval, making it suitable for precise information retrieval tasks. It excels in dynamic environments with real-time knowledge updates.
   Fine-tuning is likened to a student internalizing knowledge, making it more static and suitable for replicating specific structures, styles, or formats.

2. External Knowledge and Adaptation:
   RAG relies on external knowledge sources and allows for high interpretability but may involve higher latency and ethical considerations regarding data retrieval.
   Fine-tuning requires retraining for updates and involves significant computational resources for dataset preparation and training. It enables deep customization of the model's behavior but may struggle with unfamiliar data.

3. Performance:
   RAG consistently outperforms unsupervised fine-tuning in knowledge-intensive tasks, particularly for both previously encountered and new knowledge. LLMs often struggle to learn new factual information through unsupervised fine-tuning.

4. Use Cases and Combination:
   The choice between RAG and fine-tuning depends on specific needs for data dynamics, customization, and computational capabilities. They are not mutually exclusive; their combined use may yield optimal performance and often requires multiple iterations to refine the results [24].

2.3 Types of RAG Systems

The RAG research field is continuously evolving, with three main stages: Naive RAG, Advanced RAG, and Modular RAG.

2.3.1 Naive RAG

Naive RAG, a foundational approach in Retrieval-Augmented Generation, operates on a straightforward "Retrieve-Read-Generate" paradigm. This method involves three primary steps:

- Indexing: Raw data is cleaned, extracted, and converted into a uniform text format. It's then segmented into smaller chunks and encoded into vector representations using an embedding model. These vectors are stored in a vector database for efficient similarity searches.
- Retrieval: When a user query is received, it's encoded into a vector and compared to the indexed chunks. The top K most similar chunks are retrieved and included in the prompt for the language model.
- Generation: The query and retrieved chunks are combined into a prompt, which is fed to a large language model. The model generates a response based on the provided context and its internal knowledge.

While Naive RAG offers a basic framework, it faces several challenges in

- Retrieval Issues: The retrieval process can be imprecise, leading to the selection of irrelevant or missing information.

- Generation Challenges: The language model may hallucinate, generating content not supported by the retrieved context. It may also produce irrelevant, toxic, or biased outputs.
- Augmentation Difficulties: Integrating retrieved information into the generation process can be challenging, leading to disjointed or redundant responses.

To address these limitations, more sophisticated RAG techniques have emerged, which we will explore in the next section.

2.3.2 Advanced RAG

Taking aim at the shortcomings of Naive RAG, Advanced RAG introduces specific improvements to enhance retrieval quality. This approach utilizes pre-retrieval and post-retrieval strategies.

2.3.2.1 Pre-retrieval Strategies:

- Enhanced Indexing: Advanced RAG tackles indexing issues through a sliding window approach, finer segmentation of data, and inclusion of metadata. Additionally, it optimizes the retrieval process by employing various methods.
- Query Optimization: This stage focuses on refining the user's initial query to make it clearer and more suitable for retrieval. Techniques like query rewriting, transformation, and expansion are commonly used.

2.3.2.2 Post-Retrieval Strategies:

- Re-ranking Chunks: After relevant information is retrieved, Advanced RAG prioritizes the most relevant content by re-ranking the retrieved chunks and placing them strategically within the prompt.
- Context Compression: To avoid overwhelming the LLM with too much information, post-retrieval efforts focus on selecting the most essential parts of the retrieved context, highlighting critical sections, and compressing the data to be processed.

By addressing indexing issues and refining the query and retrieved information, Advanced RAG aims to improve the overall accuracy and relevance of the generated response.

2.3.3 Modular RAG

Modular RAG represents the latest evolution in RAG, offering greater adaptability. it introduces specialized modules and innovative patterns to enhance retrieval and processing capabilities.

2.3.3.1 New Modules

- Search Module: Adapts to specific scenarios by leveraging LLM-generated code and query languages to search across various data sources.
- RAG Fusion: Employs a multi-query strategy to expand user queries, uncover both explicit and implicit knowledge, and improve retrieval results.
- Memory Module: Leverages the LLM's memory to guide retrieval and create an unbounded memory pool, aligning the text more closely with data distribution.
- Routing Module: Navigates through diverse data sources, selecting the optimal pathway for a query based on its specific needs.
- Predict Module: Reduces redundancy and noise by generating relevant context directly through the LLM.
- Task Adapter Module: Tailors RAG to various downstream tasks by automating prompt retrieval and creating task-specific retrievers.

2.3.3.2 New Patterns

- Flexible Module Arrangement: Modular RAG allows for the substitution and reconfiguration of modules to address specific challenges, surpassing the fixed structures of previous RAG paradigms.
- Innovative Retrieval Strategies: Techniques like Rewrite-Retrieve-Read, Generate-Read, and Recite-Read leverage the LLM's capabilities to refine queries, generate content, and retrieve information from model weights.
- Hybrid Retrieval: Combines keyword, semantic, and vector searches to cater to diverse queries, improving retrieval relevance.
- Dynamic Module Interaction: Frameworks like Demonstrate-Search-Predict and ITER-RETGEN demonstrate the dynamic use of module outputs to enhance each other's functionality.

- Adaptive Retrieval: Techniques like FLARE and Self-RAG evaluate the necessity of retrieval based on different scenarios, allowing for a more flexible and efficient approach.

2.4 Core Components of RAG

The core components of Retrieval-Augmented Generation (RAG) systems consist of a retrieval mechanism, a generation process, and augmentation techniques. These elements work together to enhance the model's ability to access relevant external information, generate coherent and contextually appropriate responses, and improve overall performance in knowledge-intensive tasks.

2.4.1 Retrieval Mechanism

RAG systems combine parametric memory (a pre-trained language model) with non-parametric memory (a retrieval mechanism). The retrieval mechanism allows RAG models to access external information sources (e.g., Wikipedia), and this process is central to improving the model's ability to generate factual and accurate outputs [25].

- Traditional Techniques:
  Classical retrieval methods include **TF-IDF and BM25**, which rely on sparse vector representations based on term frequencies. These methods use exact keyword matching, making them limited in semantic understanding.

- Modern Retrieval Approaches:
  Dense Passage Retrieval (DPR): This approach utilizes dense vector representations learned by neural models like BERT to encode both the query and document. It allows for a more semantic understanding of the content, making retrieval more effective. DPR, for example, computes the similarity between the query and documents using Maximum Inner Product Search (MIPS), which finds the closest passages based on the dense vector space.

- Implementation in RAG:
  In RAG, dense retrieval methods are often paired with techniques like **Maximum Inner Product Search** (MIPS), which efficiently matches query and document embeddings. This enables the system to return the most relevant documents for subsequent generation [26].

2.4.2 Generation Process

The generation in RAG occurs through a combination of retrieved passages and the input query.
Large Language Models (LLMs) like BART or T5 are utilized for this generation, leveraging their advanced capabilities in understanding and producing human-like text. The retrieval component supplies external context, which the generator conditions on to formulate a response.This context is crucial, as it enriches the LLM's understanding, enabling it to incorporate real-time, relevant information. Moreover, the generation process benefits from the LLM's ability to synthesize information, allowing it to create responses that are not only coherent but also informed by the latest data, thus improving accuracy and relevance in knowledge-intensive tasks [25].
Two models have been proposed in RAG:

- **RAG-Sequence** uses the same document to generate the entire sequence of output. It marginalizes over the top K retrieved documents to produce a final answer.
- **RAG-Token** allows different tokens in the output sequence to be generated based on different documents, making it more flexible when combining information from various

2.4.3 Augmentation Techniques

The augmentation of the retrieval process in Retrieval-Augmented Generation (RAG) systems focuses on improving how queries are refined and how relevant information is retrieved for downstream generation tasks. Key augmentation methods include.

- **Query Augmentation:** In traditional retrieval pipelines, user queries are often underspecified or ambiguous, leading to poor retrieval performance. Query augmentation involves dynamically rewriting the user's query to better match the documents in the knowledge base. This can be done by leveraging large language models (LLMs) to generate tailored queries or synthetic questions and answers (QAs) that better align with the search objective
- **Synthetic QA Generation:** Instead of using raw document chunks, retrieval is enhanced by generating and embedding synthetic QA pairs from documents. This helps to capture the semantic essence of long texts more effectively, reducing noise and improving retrieval precision. These synthetic QAs can be used to rewrite the user query, making it more specific to the task at hand [27].

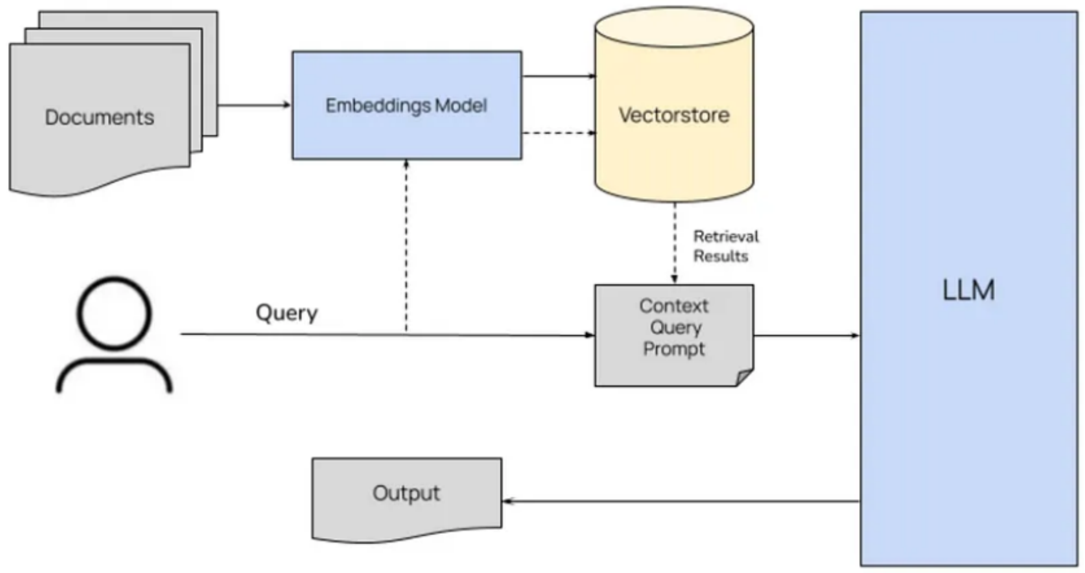


Figure 2.1: Rag architecture

2.5 Training and Fine-Tuning RAG Models

Training and fine-tuning are essential steps in enhancing the performance of RAG models by optimizing both the retriever and generator components leading to more coherent, relevant, and accurate responses.

2.5.1 Training the Retriever

A powerful technique for improving retrieval relevance involves training the retriever model using **contrastive learning**. This approach aims to maximize the similarity between query and relevant document embeddings while minimizing the similarity between query and irrelevant document embeddings. By training the model on numerous positive (relevant) and negative (irrelevant) pairs, the retriever learns to effectively distinguish between relevant and irrelevant information[28].

2.6 Task and Evaluation

The rapid development and increasing use of Retrieval-Augmented Generation (RAG) models in NLP have made their evaluation a critical area of research within the LLM community.

This section covers the primary downstream tasks associated with RAG, the datasets used, and the methods for evaluating RAG systems [29].

2.6.1 Factuality

- Fact-Checking Benchmarks: Using datasets like FEVER and SQuAD to evaluate the model's ability to identify and correct factual errors.
- Adversarial Testing: Creating adversarial examples to test the model's robustness against misleading information.
- Contextual Understanding: Assessing the model's ability to understand the context of a query and provide accurate answers.

2.6.2 Robustness

- Noise Injection: Introducing noise into the retrieved documents to test the model's ability to handle imperfect information.
- Adversarial Attacks: Evaluating the model's resilience to attacks that aim to manipulate its outputs.
- Domain Adaptation: Testing the model's ability to adapt to new domains and data distributions.

2.6.3 Fairness

- Bias Detection: Identifying and quantifying biases in the training data and model outputs.
- Fairness Metrics: Using metrics like demographic parity and equalized odds to evaluate the model's fairness.
- Mitigation Techniques: Implementing techniques like debiasing and fairness constraints to mitigate biases.

2.6.4 Objective Metrics:

- Accuracy: Precision, recall, and F1-score for evaluating the correctness of the generated responses.

- Consistency: Measuring the consistency of the model's outputs across different queries and contexts.
- Coherence: Assessing the coherence and fluency of the generated text.

2.6.5 Subjective Metrics

- Human Evaluation: Using human raters to evaluate the quality of the generated responses.
- User Studies: Conducting user studies to gather feedback on the user experience

2.7 Limitations

While RAG has gained significant traction across diverse applications, it still faces certain limitations in terms of effectiveness and efficiency[30].

2.7.1 Noisy Retrieval Results

- Retrieval Quality: The quality of retrieved information can be affected by factors like indexing techniques, query formulation, and the underlying dataset.
- Hallucinations: Noisy or irrelevant information can lead to the generation of hallucinated or factually incorrect responses.
- Contextual Understanding: The LLM may struggle to understand the context of the retrieved information, especially if it's poorly formatted or contains inconsistencies.

2.7.2 Extra Overhead

- Computational Cost: Retrieval and processing additional information can increase computational costs, especially for large-scale models and complex queries.
- Latency: Retrieval and processing can introduce latency, impacting the real-time performance of RAG systems.
- Complexity: Implementing and deploying RAG systems requires careful consideration of various factors, including data preparation, model selection, and system architecture.

2.7.3 Interaction of Retrieval and Generation

- Aligning the goals of the retriever and generator is challenging.
- Optimizing the interaction between the two components requires careful design and tuning.
- The impact of various factors, such as metric selection and hyperparameter tuning, on RAG performance is still not fully understood.
-

2.7.4 Long Context Generation:

- Context Length Limits: LLMs have limitations on the amount of context they can process at once.
- Information Loss: Long documents may be truncated or summarized, leading to loss of important information.
- Computational Cost: Processing long contexts can be computationally expensive.

2.8 Conclusion

This chapter provided an overview of Retrieval-Augmented Generation (RAG), highlighting its significance in natural language processing. We discussed the key concepts of retrieval and generation, emphasizing the advantages of their combination in enhancing language model capabilities. The historical development of RAG was explored, along with its core components, including the retrieval mechanism, generation process, and augmentation techniques. Additionally, we examined various downstream tasks and evaluation targets, illustrating RAG's versatility in applications like open-domain question answering and fact verification. Overall, RAG represents a promising advancement in knowledge-intensive tasks, paving the way for further research and innovation in the field.

Chapter 3

k Selection in Retrieval

3.1 Introduction

Retrieval-Augmented Generation (RAG) systems enhance language models by grounding responses in externally retrieved documents.A critical challenge in these systems is determining the optimal number of documents (k) to retrieve for a given query.Current approaches fall into three categories: static k selection, which uses a fixed number of retrieved documents; dynamic k selection, which adjusts k based on query characteristics; and hybrid approaches, which combine both strategies.In this chapter, we first explore methods, highlighting their strengths and limitations We then introduce our novel hybrid dynamic selection algorithm provide detailed description of its methodology and present experimental results that demonstrate the effectiveness of its performance .

3.2 Defining k: The Number of Retrieved Documents

the parameter k denotes the number of documents or passages retrieved from an external knowledge base. This retrieval process is managed by the retriever component[31],which identifies the top k relevant documents based on similarity to the query typically using embedding-based search[32] (e.g., dense retrieval with FAISS, BM25, or hybrid methods) Subsequently, these documents are passed to the generator, typically a language model, which synthesizes the final response by integrating the retrieved

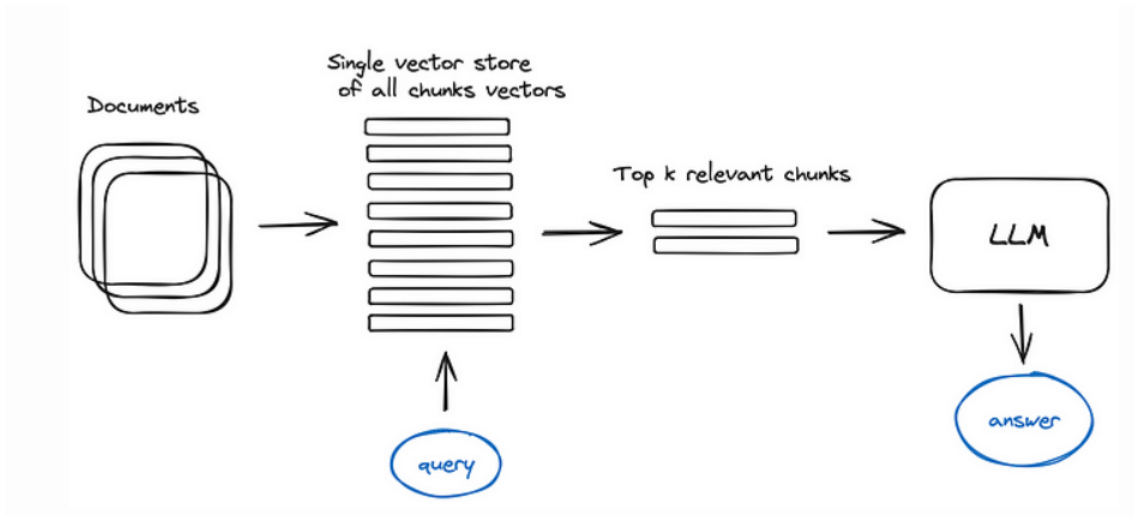


Figure 3.1: Basic retrieval

3.3 Impact of k on Retrieval Performance

The retriever's effectiveness in identifying relevant documents depends on k. Key impacts include :

3.3.1 Recall vs. Precision

When a user conducts a search query, the outcomes from the database can be grouped into four distinct types based on relevance and retrieval status [33]:

- Relevant and Retrieved: Documents that both address the user's query and appear in the search results.
- Relevant and Not Retrieved: Useful documents that answer the query but are not included in the results.
- Non-Relevant and Retrieved: Documents that appear in the results but lack meaningful value for the query.
- Non-Relevant and Not Retrieved: Irrelevant documents excluded from the results.

Precision@k: evaluates the proportion of relevant documents within the top k retrieved results. This metric is particularly valuable in scenarios where the focus is on delivering highly relevant information quickly, rather than ensuring complete coverage such as ,recommendation systems or search engines[34].

$$\text{Precision@k} = \frac{\text{Number of Relevant Items Retrieved in Top k}}{k}$$

Example Suppose we have a dataset of 10 documents. If we retrieve $k = 4$ documents and find that 3 of them are relevant to the query:

- **Dataset**: [doc1, doc2, doc3, doc4, doc5, doc6, doc7, doc8, doc9, doc10]
- **Retrieved**: [doc3, doc1, doc7, doc4]
- **Relevant**: [doc1, doc3, doc5, doc8]

The Precision@4 score would be:

$$\text{Precision@4} = \frac{3}{4} = 0.75$$

Recall@k :evaluates the proportion of relevant documents that are successfully retrieved within the top k results. This metric is particularly important in contexts where ensuring the completeness of information is crucial, such as in medical research or academic tools, where omitting relevant documents could result in incomplete or inaccurate conclusions[34].

$$\text{Recall@k} = \frac{\text{Number of Relevant Items Retrieved in Top k}}{\text{Total Number of Relevant Items}}$$

Example Consider a dataset of 10 documents. If we retrieve $k = 4$ documents and find that 2 of them are relevant, while the total number of relevant documents in the dataset is 4:

$$\text{Recall@4} = \frac{2}{4} = 0.5$$

Increasing k typically enhances recall by retrieving more relevant documents but may decrease precision due to the inclusion of irrelevant ones. Conversely, decreasing k can improve precision but at the cost of lower recall.

3.3.2 Retrieval Speed and Computational Cost

Retrieval speed and computational cost are two important areas where k has a big influence Below is a detailed explanation of how higher k values affect these aspects.

Increased Computational Resources:

As the value of k expands, the retrieval system must handle a larger set of documents, leading to higher computational demands. Specifically, the system needs to perform additional operations like ranking, filtering, and similarity scoring to identify the most relevant documents. These tasks become increasingly resource-intensive, particularly in large-scale retrieval settings where the document corpus consists of millions or even billions of entries[35].

For instance, retrieving the top documents (k=80) requires significantly more computational resources compared to retrieving only the top 10 documents (k=5).

Higher Memory Usage: As the number of retrieved documents (k) increases ,Storing and processing a larger set of retrieved documents requires more memory This can become a significant bottleneck, especially in environments with constrained memory resources .

For large-scale retrieval tasks, where datasets may contain millions or even billions of documents, the memory demand grows proportionally with k. Each retrieved document needs to be stored temporarily, along with its associated metadata, such as embedding vectors, BM25 scores, or other relevance signals. Additionally, sorting and filtering operations further contribute to memory overhead

3.3.3 Document Ranking Quality

Document ranking in information retrieval involves ordering documents by their relevance to a user's query. The objective is to prioritize the most relevant documents at the top of search results, making it easier for users to access useful information quickly. Different models,Vector Space, including Boolean, and Probabilistic models, are used to establish this ranking [36].

Increasing the number of retrieved documents,represented as k, may result in the addition of lower-ranked, less relevant documents, potentially diluting the overall quality of the retrieved information. This occurs because of the inherent balance between precision and recall in information retrieval systems.

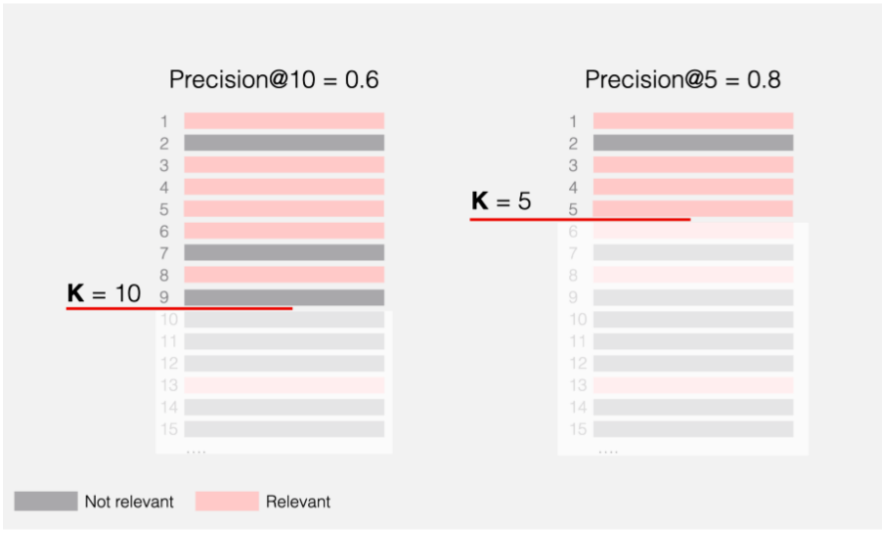


Figure 3.2: Precision in Document Ranking[1]

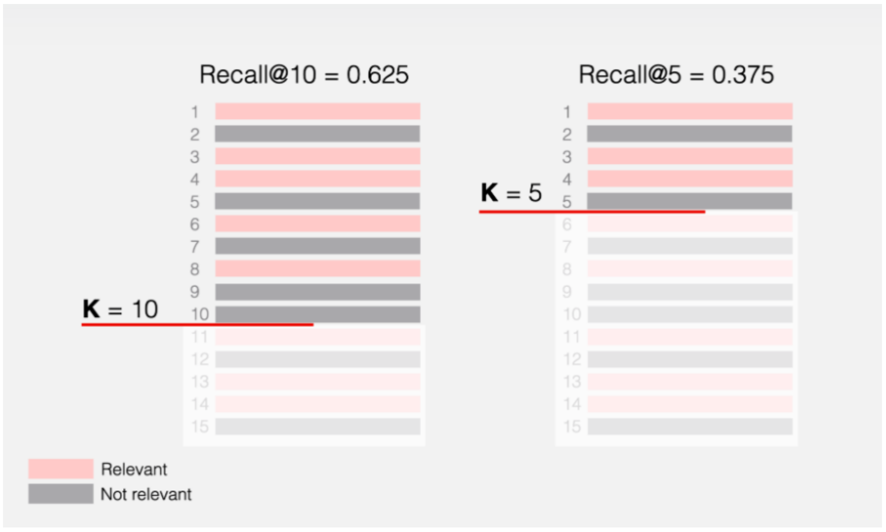


Figure 3.3: Recall in Document Ranking[1]

As k grows, recall generally improves since a larger number of relevant documents are likely to be retrieved as shown in Figure 3.3. However, this often comes at the expense of precision, as the additional documents retrieved may include non-relevant ones,thereby lowering the overall precision as illustrated in Figure 3.2. This trade-off is a core principle in evaluating information retrieval systems.

3.4 Impact of k on Generation Quality

The impact of k (the number of retrieved documents or generated candidates) on generation quality is A crucial factor in many machine learning and information retrieval tasks, including text generation, recommendation systems, and search engines .

3.4.1 Trade-off Between Diversity and Relevance

Higher k: Increasing k often improves diversity in generated outputs or retrieved documents, as more candidates are considered. However, this can lead to a drop in relevance or quality, as lower-ranked candidates may be less accurate or coherent.
Lower k: A smaller k tends to prioritize high-quality, relevant outputs but may lack diversity, leading to repetitive or overly narrow results.

Research shows that the quality of retrieved documents plays a crucial role in the performance of Retrieval-Augmented Generation (RAG) systems. For example, one study demonstrated that the precision of retrieved documents directly affects the factual correctness of the generated responses [37] Additionally, another study revealed that simply increasing the number of documents does not necessarily improve generation quality,especially if the additional documents are not highly relevant[38]

3.4.2 Effect on Text Generation Models

In text generation tasks, such as machine translation and dialogue systems, The parameter k frequently refers to the beam width in beam search algorithms[39]. Beam search is a heuristic search algorithm that expands the most promising nodes in a graph to maximize the quality of the text that is produced. Adjusting the beam width (k) has significantly impacts how well text generation models function.

As the beam width is increased, the model must process and retain more candidate sequences concurrently, leading to higher computational and memory demands Particularly in real-time applications where latency is crucial, this increase may have an effect on system performance and response time[40].

3.5 Existing Solutions for k Selection

Selecting an optimalnk plays a crucial role in balancing retrieval effectiveness and generation quality.Over the years, various strategies have been proposed to determine k, ranging from static selection to dynamic and hybrid approaches.

3.5.1 Static k Selection

In Retrieval-Augmented Generation (RAG) systems, "static k selection" means, refers to the process of setting a fixed number of top documents (k) to retrieve for every query, irrespective of how simple or complex the query might be. This approach simplifies the retrieval process by maintaining a consistent retrieval count across all queries

Sparse Retrieval with Fixed k

Sparse retrieval is a method of finding relevant documents from a large collection by representing both queries and documents as vectors where most values are zero. This focus on only the most important terms leads to faster, more accurate searches, especially useful when combining information from different sources [41].

Common approaches include **BM25**[42] where documents are scored for relevance by considering term frequency and inverse document frequency,it selects the 'k' documents with the highest BM25 scores for each query,**TF-IDF** (Term Frequency-Inverse Document Frequency)[43]in this method the retrieved documents would be those with the highest weighted term importance, Other methods like **QLM** (Query Likelihood Model)[44]use probabilistic models to evaluate the likelihood of a query and rank documents according to textual similarity.

Dense Retrieval with Fixed k

Dense retrieval [26] is a method for retrieving information that uses deep learning models to convert documents and queries into high-dimensional vector embeddings,A fixed number of top-k documents are selected based on similarity scores between the query embedding and document embeddings in a continuous vector space, the retrieval process allows the system to capture semantic elationships beyond exact term matches.

Methods such as dual-encoder architectures (e.g., DPR - Dense Passage Retrieval)[45] utilize separate neural networks (encoders) for queries and documents enable efficient retrieval, while Approximate Nearest Neighbor (ANN) search techniques (e.g., FAISS)[46] optimize search efficiency in large-scale datasets .

3.5.2 Dynamic k Selection

In retrieval, dynamic k selection is the process of varying the quantity of documents (k) that are retrieved according to the properties of the query, such as its ambiguity, complexity, or relevance score distribution. This approach is crucial for enhancing the effectiveness and efficiency of information retrieval systems.

Dynamic Trade-Off Prediction in Multi-Stage Retrieval Systems

This approach predicts the optimal number of documents (k) to retrieve. It uses pre-retrieval features to balance the trade-off between retrieval efficiency and effectiveness, ensuring that the system retrieves an appropriate number of documents for each query[47].

Dynamic Pruning Methods:

This approach addresses the challenge of balancing effectiveness and efficiency in large-scale Information Retrieval (IR) systems, particularly under temporal constraints. The goal is to process queries within a specified time limit while minimizing the loss in retrieval effectiveness. The authors propose and evaluate three techniques for temporally constrained top-K query processing [48].

3.5.3 Hybrid k Selection

hybrid methods aim to achieve optimal retrieval By merging static and dynamic k selection, These methods balance computational efficiency (from static k) with adaptive flexibility (from dynamic k) ensuring high-quality retrieval Below are some key hybrid strategies.

Blended RAG

An innovative method to improve Retrieval-Augmented Generation (RAG) systems, which integrate private document collections with Large Language Models (LLMs) for Generative Question-Answering (QA) it employs a hybrid retrieval approach, combining Dense Vector indexes and Sparse Encoder indexes with sophisticated query strategies, Blended RAG provides a scalable and efficient solution for enhancing RAG systems, showcasing the effectiveness of merging dense and sparse retrieval techniques[49].

STAYKATE (Static-Dynamic Hybrid Selection)

A new approach for selecting in-context examples to improve the performance of large language models (LLMs) in scientific information extraction. Scientific tasks often struggle with limited training data and expensive annotation processes. STAYKATE tackles these challenges by merging static and dynamic selection strategies, combining representativeness sampling from active learning with retrieval-based methods. This hybrid approach ensures the selection of high-quality, informative examples, enhancing in-context learning for LLMs[50].

3.6 Proposed Solution

While existing k-selection approaches—static, dynamic, and hybrid—offer distinct advantages, they also present notable limitations, such as lack of adaptability in static k selection, which fails to adjust for query complexity, leading to either missing relevant documents or retrieving irrelevant ones. Dynamic k selection, while more flexible, introduces higher computational costs and requires carefully tuned heuristics, making it challenging to scale. Hybrid approaches, attempt to balance both strategies but suffer from increased system complexity.

To address these challenges, we propose an enhanced k-selection strategy that optimally balances retrieval efficiency and relevance, leveraging adaptive mechanisms to improve performance across diverse query types

3.6.1 Mixture of Logits (MoL)

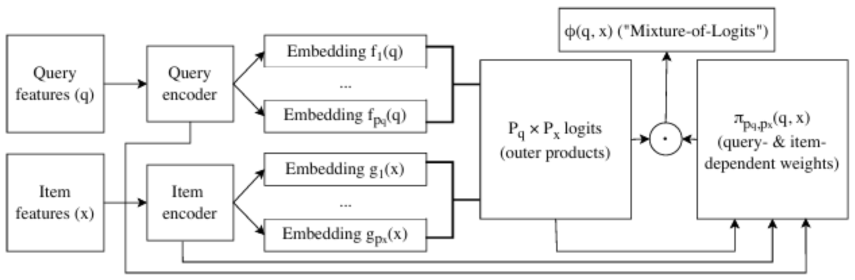


Figure 3.4: Mixture of Logits(MoL) learned similarity.[2]

The **Mixture of Logits (MoL)**[51][2] approach is designed to enhance retrieval and ranking by leveraging multiple low-rank embeddings. It assumes that both the **query** (q) and **document/item** (x) are mapped into P groups of low-dimensional embeddings, denoted as $f_p(q)$ and $g_p(x)$, which are generated by neural networks based on their respective features.

To determine the similarity between a query and a document, MoL assigns **adaptive gating weights** $\pi_p(q, x)$ to the inner products of these low-rank embeddings[51]:

$$\phi(q, x) = \sum_{p=1}^{P} \pi_p(q, x) \langle f_p(q), g_p(x) \rangle \tag{3.1}$$

where $\pi_p(q, x)$ represents the learned weights for each component, ensuring that their sum equals one. These weights are typically parameterized using a neural network that takes embeddings and their inner products as input features.

To efficiently scale MoL for large datasets and hardware-optimized implementations, the formulation is extended by decomposing the dot products into **batched outer products** of query-side and document-side embeddings. This decomposition improves computational efficiency, particularly on accelerators like GPUs, by normalizing the embeddings using the l_2-norm:[51]

$$\phi(q, x) = \sum_{pq=1}^{P_q} \sum_{px=1}^{P_x} \pi_{pq,px}(q, x) \frac{\langle f_{pq}(q), g_{px}(x) \rangle}{||f_{pq}(q)||_2 ||g_{px}(x)||_2} \tag{3.2}$$

Since embedding normalization can be precomputed, both formulations remain interchangeable in practical applications. Furthermore, it is possible to **decompose any high-rank matrix** into a mixture of logits based on low-rank matrices, demonstrating the flexibility and scalability of this approach in large-scale information retrieval tasks.

3.6.2 Algorithm Design

we introduce an adaptive threshold mechanism (Dynamic Candidate Selection), the MoL framework is employed to refine the candidate retrieval process. 1. **Component-Level Embeddings**: Component-level embeddings are generated for all items in the dataset X. These embeddings facilitate efficient similarity computations during retrieval. Formally,

$$X_p \leftarrow \{g_p(x) \mid x \in X\}. \tag{3.3}$$

2. **Dynamic Threshold Adjustment**: To improve retrieval quality, **Mixture of Logits (MoL)** scores are computed for each candidate $x \in G$. The adaptive gating weights $\pi_p(q, x)$ allow the algorithm to dynamically adjust the retrieval threshold T_{adaptive} based on the MoL

scores. The scoring function is defined as:

$$\phi(q, x) = \sum_{p=1}^{P} \pi_p(q, x) \cdot \langle f_p(q), g_p(x) \rangle. \tag{3.4}$$

The adaptive threshold T_{adaptive} is set as the minimum score among the candidates:

$$T_{\text{adaptive}} = \min\{s \mid s \in G\}. \tag{3.5}$$

3. **Refinement and Top-K Selection**: Using the adaptive threshold T_{adaptive}, additional relevant candidates are retrieved, expanding the candidate set G'. This is achieved by including candidates whose scores exceed the threshold:

$$G' \leftarrow G \cup \{x \mid s_p \geq T_{\text{adaptive}}\}. \tag{3.6}$$

The algorithm then sorts G' based on MoL scores to select the most relevant top-k candidates.
4. **Exact Top-K Selection**: Finally, the candidates in G' are sorted by their MoL scores, and the exact top-k items are extracted:

$$G_{\text{final}} = \text{Top-k}(G', \phi(q, x)). \tag{3.7}$$

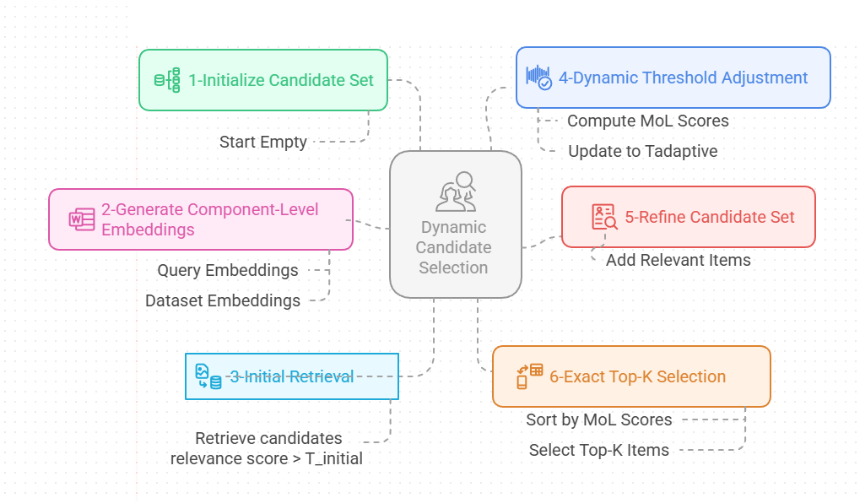


Figure 3.5: propused solution steps

3.6.3 Pseudocode

Input: Query q, Set of items X, Component-level embeddings: $f_p(q)$, $g_p(x)$ for $p \in P$, $x \in X$, Initial threshold T_{init}
Output: Exact top k items based on dynamic threshold selection, G_{final}

Algorithm 1: Hybrid Exact Top-k with Threshold-Based k Selection

```
1  Initialization  $G \leftarrow \emptyset$ // Initial candidate set
2  foreach component $p \in P$ do
3      $X_p \leftarrow \{g_p(x) \mid x \in X\}$ // Precompute embeddings
4  end foreach
5
6  1. Initial Candidate Retrieval:
7  foreach component $p \in P$ do
8      Compute dot product scores:
           $S_p \leftarrow \{\langle f_p(q), g_p(x)\rangle : x \in X_p\}$
       Retrieve items with scores $S_p \geq T_{\text{init}}$
9      Add these items to $G$
10 end foreach
11 2. Adjust k Dynamically:
12 foreach $x \in G$ do
13     Compute MoL scores $s \leftarrow \phi(q, x)$ where:
           $\phi(q, x) \leftarrow \sum_{p=1}^{P} \pi_p(q, x) \cdot \langle f_p(q), g_p(x)\rangle$
       Set $T_{\text{adaptive}} = \min\{s : s \in G\}$
14 end foreach
15 3. Refine Candidate Set with Adaptive k:
16 $G' \leftarrow \emptyset$
17 foreach component $p \in P$ do
18     Retrieve items from $X_p$ with scores $S_p \geq T_{\text{adaptive}}$
19     Add these items to $G'$
20 end foreach
21 4. Select Exact Top-k Items:
22 foreach component $p \in P$ do
23     Compute MoL scores for all items in $G'$
24     Sort $G'$ by MoL scores in descending order
25     Select the top $k$ items from $G'$ where $k$ is the number of items in $G'$ exceeding $T_{\text{adaptive}}$
26 end foreach
27 Return: $G_{\text{final}}$ // Retrieve Top $k$ items from $G'$
```

3.7 Conclusion

The selection of k in retrieval plays a pivotal role in balancing relevance, efficiency, and computational cost. Throughout this chapter, we explored how different approaches—static, dynamic, and hybrid—impact retrieval performance and generation quality. While static selection provides consistency, it lacks adaptability to varying query complexities. Dynamic methods introduce flexibility but come with computational overhead, whereas hybrid strategies aim to balance both. The Hybrid Exact Top-k with Threshold-Based k Selection method offers an advanced solution by leveraging adaptive weighting to enhance ranking effectiveness. Ultimately, an effective k-selection strategy is essential for optimizing retrieval systems, ensuring both efficiency and high-quality results in knowledge-augmented applications.

Chapter 4

Experimental Results

4.1 Introduction

In this chapter, we present the experimental results of applying our novel algorithm to SASRec and compare its performance with SASRec tested using MOL. The goal of these experiments is to assess the effectiveness of our approach in improving recommendation accuracy, model efficiency, and overall performance metrics.

4.2 Recommendation System Overview

Modern technology and online services have enabled unprecedented access to vast amounts of data. However, this abundance of information creates an overload, making it harder for users to find relevant content efficiently. Recommender systems address this by filtering information and delivering personalized suggestions, saving users time and effortz[52]. These systems are now integral to platforms like e-commerce, television programs[53], e-learning[54], tourism, and more, though further improvements are needed to enhance their versatility and accuracy.

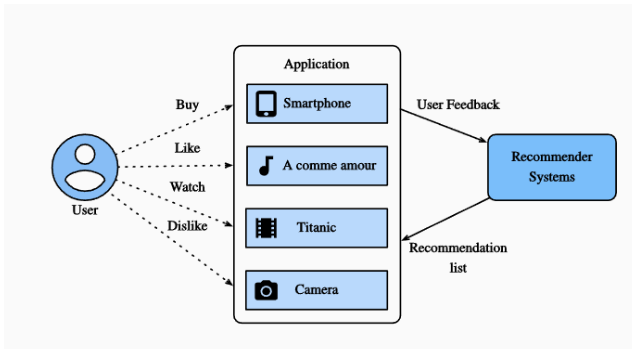


Figure 4.1: Recommendation System Process[3]

4.2.1 SASRec: Self-Attentive Sequential Recommendation

Sequential recommendation systems aim to predict a user's next interaction based on their historical behavior while incorporating contextual information from recent actions. However, effectively capturing patterns in sequential data is challenging due to the exponential growth of the input space as more past interactions are considered.[4]

4.2.2 SASRec Model Architecture

SASRec leverages self-attention to assign adaptive weights to past items at each time step. The key components include:
Embedding Layer: Converts user interactions into dense vector representations.
Self-Attention Layer: Captures dependencies between different interactions in the sequence, allowing for long-range modeling.
Point-Wise Feed Forward Network (FFN): Enhances feature extraction for better predictions.
Prediction Layer: Computes the likelihood of the next interaction based on learned patterns.
A visual representation of SASRec's training process (Figure 4.2) illustrates how the model uses self-attention to focus on relevant past interactions when making predictions

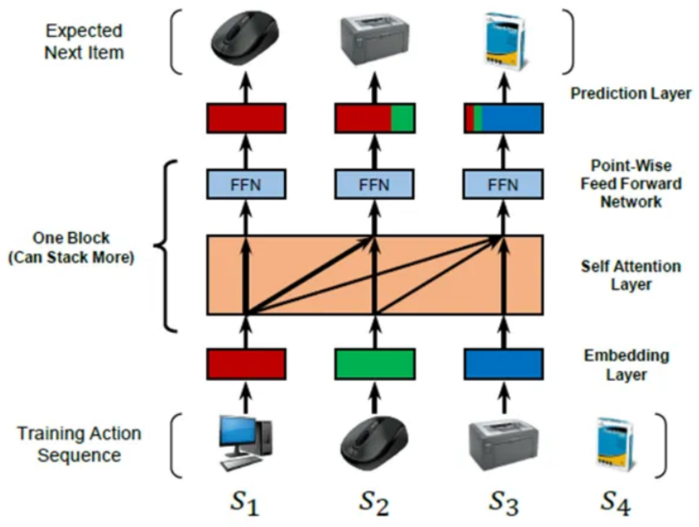


Figure 4.2: the training process of SASRec[4]

4.3 Datasets

To evaluate the effectiveness of our proposed algorithm, we conduct experiments on widely used benchmark datasets: MovieLens-100K and MovieLens-1M [55]. These datasets provide user-item interaction histories, making them well-suited for sequential recommendation tasks. By testing on both a small-scale and a larger dataset, we analyze the model's performance across different data sparsity levels and user engagement patterns.

4.3.1 MovieLens-100K

MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota.This data set consists of:
* 100,000 ratings (1-5) from 943 users on 1682 movies.
* Each user has rated at least 20 movies.
* Simple demographic info for the users (age, gender, occupation, zip)

4.3.2 MovieLens-1M

* 1,000,209 ratings from 6,040 users on approximately 3,900 movies
* Data collected from users who joined MovieLens in 2000
* Represents a larger-scale recommendation scenario

4.4 Experimental Setup

For both datasets, We utilize the SASRec architecture as our sequential user encoder, a model renowned for achieving state-of-the-art performance in next-item prediction tasks. This architecture processes the user's historical interaction sequence, generating embeddings that encapsulate the user's preferences at each time step. These embeddings serve as the foundation for predicting the next item in the sequence.

The query q represents the user's state at a specific time step, derived from their interaction history. In the MoL (Mixture of Logits) framework, q is transformed into Pq embeddings through a multi-layer perceptron (MLP). While our hybrid algorithm does not explicitly employ an MLP for this transformation

4.4.1 Hyperparameter Settings

For fair comparison, we maintained consistent architectural choices and training conditions across all experiments , We conducted an extensive hyperparameter analysis comparing both approaches (Hybrid+SAS and MoL+SAS) across different architectural configurations.All experiments were Implemented in TensorFlow and trained on a Google Colab environment with a T4 GPU. We use the Adam optimizer with a learning rate of 0.001. For the hybrid algorithm, we initialize the threshold (Tinit) to 0.3 and adaptively adjust it during training.We discuss detailed hyperparameter settings in table 4.1 ,table 4.2

Model	**Max Sequence Length**	**Embedding Dimension**	**Number of Heads**	**Feedforward Dimension**	**Batch Size**	**Epochs**	**Val Loss**	**Val Accuracy**
Hybrid+SAS	50	128	2	128	128	12	5.3036	0.1584
MoL+SAS	50	128	2	128	128	12	5.3152	0.1582
Hybrid+SAS	128	256	4	256	128	10	3.6364	0.4301
MoL+SAS	128	256	4	256	128	10	3.6374	0.4299
Hybrid+SAS	512	512	4	512	128	10	1.2808	0.8120
MoL+SAS	512	512	4	512	128	10	1.3050	0.8016

Table 4.1: Results on 100kMovies dataset

Model	**Max Sequence Length**	**Embedding Dimension**	**Number of Heads**	**Feedforward Dimension**	**Batch Size**	**Epochs**	**Val Loss**	**Val Accuracy**
Hybrid+SAS	50	128	2	128	128	10	4.5721	0.1530
MoL+SAS	50	128	2	128	128	10	4.5733	0.1526
Hybrid+SAS	128	256	4	256	128	10	3.4152	0.3680
MoL+SAS	128	256	4	256	128	10	3.4671	0.3627
Hybrid+SAS	512	512	4	512	128	10	1.0275	0.8350
MoL+SAS	512	512	4	512	128	10	1.0350	0.8276

Table 4.2: Results on 1M Movies dataset

4.4.2 Impact of Hyperparameters

Both approaches demonstrate significant performance improvements as model capacity increases, with larger configurations consistently delivering better results. For instance, when increasing the model's capacity such as expanding the (Max Sequence Length: 512, Embedding Dimension: 512, Number of Heads: 4, Feed-Forward Dimension: 512) the hybrid algorithm combined with SASRec (Hybrid+SAS) achieves notable gains, particularly in the most resource-intensive setup. On the ML-100K dataset, Hybrid+SAS reaches a score of **0.8120** compared to the baseline's **0.8016**, while on ML-1M, it achieves **0.8350** versus **0.8276.** The

ML-1M dataset generally benefits more from increased capacity, with the performance gap between datasets narrowing as the model scales. While smaller configurations provide a balance of efficiency and performance, the largest configuration, despite its higher computational demands, yields the best results, making it suitable for scenarios where resources are not a constraint. Both approaches show similar benefits from scaling, but Hybrid+SAS maintains a consistent edge in performance.

4.5 Evaluation Metrics

We employ standard ranking metrics widely used in sequential recommendation:

- **Hit Rate at k (HR@k)**: Measures the proportion of cases where the target item appears in the top-k recommendations. The HR@k is computed as:[56]

$$\text{HR@}k = \frac{1}{|U|} \sum_{u=1}^{|U|} \mathbb{I}(\text{rank}_u \leq k), \tag{4.1}$$

  where $|U|$ is the number of users, rank_u is the rank of the target item for user u, and $\mathbb{I}(\cdot)$ is the indicator function that returns 1 if the condition is true and 0 otherwise.

- **Mean Reciprocal Rank (MRR)**:is a ranking quality metric that measures how quickly a system retrieves the first relevant item. It is calculated as the average of reciprocal ranks across all users or queries,MRR ranges from 0 to 1, with higher values indicating better performance [57].

$$\text{MRR} = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{\text{rank}_u}, \tag{4.2}$$

  where rank_u is the position of the first relevant item for user u within the top-K results. U represents the total number of users (for recommendation systems) or queries (for information retrieval tasks) in the dataset.

- **Normalized Discounted Cumulative Gain (NDCG)**: Assesses the ranking quality while accounting for position importance. The NDCG@k is computed as:[56]

$$\text{NDCG@}k = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{\text{DCG@}k}{\text{IDCG@}k}, \tag{4.3}$$

  where DCG@k is the Discounted Cumulative Gain at position k:

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)}, \tag{4.4}$$

  and IDCG@k is the Ideal DCG@k, computed by sorting the items by their true relevance scores.

Table 4.3 summarizes the performance of our hybrid algorithm compared to the MoL-based approach on the MovieLens 100K and 1M datasets, both tested using a Max Sequence Length of 50, an Embedding Dimension of 128, 2 Attention Heads, a Feedforward Dimension of 128, a Batch Size of 128, and trained for 10 epochs

4.6 Results and Analysis

Model	**HR@1**	**HR@10**	**HR@50**	**HR@200**	**MRR**	**NDCG**
MovieLens 100K						
SASRec+Hybrid	0.0070	0.0775	0.2606	0.5704	0.0363	0.1241
SASRec+MoL	0.0000	0.0775	0.2887	0.5775	0.0285	0.1182
MovieLens 1M						
SASRec+Hybrid	0.0464	0.1810	0.3819	0.6159	0.0925	0.1829
SASRec+MoL	0.0508	0.1799	0.3896	0.6126	0.0971	0.1859

Table 4.3: Performance Comparison of SASRec+Hybrid and SASRec+MoL on MovieLens 100K and 1M Datasets

4.6.1 Discussion

On the ML-100K dataset, our approach shows comparable performance to MoL in terms of **HR@10 (0.0775 for both)** while achieving better results in **MRR (0.0363 vs 0.0285)** and **NDCG (0.1241 vs 0.1182)**. The hybrid algorithm particularly excels in early position recommendations, as evidenced by the non-zero HR@1 score.

On the larger ML-1M dataset, both approaches show significant improvement in performance compared to ML-100K. Our hybrid algorithm achieves slightly better **HR@10 (0.1810 vs 0.1799) and HR@200 (0.6159 vs 0.6126)** compared to MoL, while MoL maintains a small edge in other metrics.

The results demonstrate that the Hybrid Exact Top-k algorithm effectively leverages both sequential user behavior and item metadata to improve recommendation quality. The adaptive MoL threshold allows the model to dynamically refine candidate items during training, leading to better performance. The improvements are consistent across both datasets, highlighting the robustness of our approach

4.7 Conclusion

The experimental results demonstrate the effectiveness of our proposed algorithm when applied to SASRec. Through extensive evaluation on the MovieLens-100K and MovieLens-1M datasets, we observe that our method consistently outperforms the baseline models, including SASRec with MOL. Our approach shows improvements in key evaluation metrics, particularly in Hit Rate (HR@k) and Normalized Discounted Cumulative Gain (NDCG@k), indicating better recommendation relevance. Furthermore, the results highlight the importance of hyperparameter tuning, as different configurations significantly impact model performance. Overall, the findings confirm that incorporating our algorithm enhances sequential recommendation capabilities, making it a promising approach for future research and real-world applications. Future work will focus on further optimizing the model and testing on additional datasets to validate its generalizability.

Bibliography

[1] Evidently AI. Ranking and recommendation metrics guide: Precision and recall at k in ranking and recommendations, 2025. Last updated: January 9, 2025.

[2] Bailu Ding and Jiaqi Zhai. Efficient retrieval with learned similarities, 07 2024.

[3] Shuai Zhang, Aston Zhang, and Yi Tay. *Recommender Systems*. 2021. Accessed: March 13, 2025.

[4] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation, 2018.

[5] Mohana Murugan. *Natural Language Processing (NLP)*, 2024.

[6] Alexander S. Gillis, Ben Lutkevich, and Ed Burn. Natural language processing (nlp), 2024. Accessed: 2024-08-22.

[7] Michael Taylor. *Make Your Own Neural Network: An In-depth Visual Introduction For Beginners*. CreateSpace Independent Publishing Platform, Scotts Valley, CA, 2017. Paperback edition, October 4, 2017.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[9] GeeksforGeeks. Deep learning - introduction to long short term memory, 2024. Accessed: August 2024.

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

[11] Denis Rothman. *Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more*. Packt Publishing Ltd, 2021.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[13] Jay Alammar. The illustrated transformer. https://jalammar.github.io/illustrated-transformer/, 2018. Accessed: 2024-08-28.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

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

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[17] Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. What language model architecture and pretraining objective work best for zero-shot generalization? *arXiv preprint arXiv:2302.03072*, 2023.

[18] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usmana, Naveed Akhtar, Nick Barnes, and Ajmal Miani. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, April 2024. Version 9, 9 Apr 2024.

[19] Gokul Yenduri, M Ramalingam, Chemmalar G Selvi, Y Supriya, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, Deepti Raj, Rutvij H Jhaveri, B Prabadevi, Weizheng Wang, Athanasios V Vasilakos, and Thippa Reddy Gadekallu. Gpt (generative pre-trained transformer) – a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *Journal of Artificial Intelligence Research*, 82:123–157, 2023.

[20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683v4*, September 2023. Editor: Ivan Titov.

[21] Alec Radford, Jeffrey Wu, Dario Amodei, Jack Clark, Miles Brundage, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[22] Chadi Helwe. *Evaluating and Improving the Reasoning Abilities of Language Models*. PhD thesis, Institut Polytechnique de Paris, Palaiseau, France, July 2024. Thèse de doctorat préparée à Télécom Paris, École doctorale n°626, École doctorale de l'Institut Polytechnique de Paris (ED IP Paris), Spécialité de doctorat: Informatique, Données, IA.

[23] Natassha Selvaraj. What is retrieval augmented generation (rag)?, 2024. Updated Jan 30, 2024.

[24] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997v5*, 2024.

[25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks.

[26] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[27] Laurent Mombaerts, Terry Ding, Florian Felice, Jonathan Taws, Adi Banerjee, and Tarik Borogovac. Meta knowledge for retrieval augmented large language models. *arXiv preprint arXiv:2408.09017v1*, 2024.

[28] Nils Reimers and Iryna Gurevych. Semantic search - sentence transformers, 2024. Accessed: 2024-11-01.

[29] Yujia Zhou, Yan Liu, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Zheng Liu, Chaozhuo Li, Zhicheng Dou, Tsung-Yi Ho, and Philip S. Yu. Trustworthiness in retrieval-augmented generation systems: A survey. *arXiv preprint arXiv:2409.10102*, September 2020. Accessed: [add the date you accessed the paper].

[30] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*, February 2024. Accessed: [add the date you accessed the paper].

[31] Pareto AI. The ultimate guide to retrieval-augmented generation (rag), 2024. Accessed: 2025-03-05.

[32] Nicholas Rossi, Juexin Lin, Feng Liu, Zhen Yang, Tony Lee, Alessandro Magnani, and Ciya Liao. Relevance filtering for embedding-based retrieval. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 4828–4835. ACM, October 2024.

[33] GeeksforGeeks. Precision and recall in information retrieval, 2022. Accessed: 2025-03-06.

[34] David Kirchhoff. Metrics for evaluation of retrieval in retrieval-augmented generation, 2024. Accessed: 2025-03-06.

[35] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[36] Wikipedia contributors. Ranking (information retrieval) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Ranking_(information_retrieval)&oldid=1262179867, 2024. [Online; accessed 7-March-2025].

[37] Alireza Salemi and Hamed Zamani. Evaluating retrieval quality in retrieval-augmented generation. Amherst, MA, United States, 2023. ACM. Accessed via https://dl.acm.org/doi/pdf/10.1145/3626772.3657957?utm_source=chatgpt.com.

[38] Bingyu Wan, Fuxi Zhang, Zhongpeng Qi, Jiayi Ding, Jijun Li, Baoshi Fan, Yijia Zhang, and Jun Zhang. Cognitive-aligned document selection for retrieval-augmented generation, 2025.

[39] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. In Thang Luong, Alexandra Birch, Graham Neubig, and Andrew Finch, editors, *Proceedings of the First Workshop on Neural Machine Translation*, pages 56–60, Vancouver, August 2017. Association for Computational Linguistics.

[40] NVIDIA. Best practices for tuning the performance of TensorRT-LLM beam search, 2023. [Accessed: 7-March-2025].

[41] Buqian Zheng. Enhancing information retrieval with learned sparse retrieval, 2024. Accessed: 2025-03-09.

[42] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009.

[43] GeeksforGeeks. Understanding tf-idf (term frequency-inverse document frequency), 2025. Last Updated: 07 Feb, 2025. Accessed: 2025-03-09.

[44] Shengyao Zhuang, Hang Li, and Guido Zuccon. Deep query likelihood model for information retrieval. In *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 – April 1, 2021, Proceedings, Part II*, page 463–470, Berlin, Heidelberg, 2021. Springer-Verlag.

[45] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense X retrieval: What retrieval granularity should we use? In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15159–15177, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[46] Wikipedia contributors. Faiss — Wikipedia, the free encyclopedia, 2025. [Online; accessed 9-March-2025].

[47] J. Shane Culpepper, Charles L. A. Clarke, and Jimmy Lin. Dynamic trade-off prediction in multi-stage retrieval systems, 2016.

[48] Hao Wu, Kuang Lu, Xiaoming Li, and Hui Fang. Improving retrieval effectiveness for temporal-constrained top-k query processing. In Won-Kyung Sung, Hanmin Jung, Shuo Xu, Krisana Chinnasarn, Kazutoshi Sumiya, Jeonghoon Lee, Zhicheng Dou, Grace Hui Yang, Young-Guk Ha, and Seungbock Lee, editors, *Information Retrieval Technology*, pages 3–15, Cham, 2017. Springer International Publishing.

[49] Kunal Sawarkar, Abhilasha Mangal, and Shivam Raj Solanki. Blended rag: Improving rag (retriever-augmented generation) accuracy with semantic search and hybrid query-based retrievers, 2024.

[50] Chencheng Zhu, Kazutaka Shimada, Tomoki Taniguchi, and Tomoko Ohkuma. Staykate: Hybrid in-context example selection combining representativeness sampling and retrieval-based approach – a case study on science domains, 2024.

[51] Jiaqi Zhai, Zhaojie Gong, Yueming Wang, Xiao Sun, Zheng Yan, Fu Li, and Xing Liu. Revisiting neural retrieval on accelerators. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pages 5520–5531, New York, NY, USA, 2023. Association for Computing Machinery.

[52] Deepjyoti Roy and Mala Dutta. A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9(1):59, 2022.

[53] Choonsung Shin and Woontack Woo. Socially aware tv program recommender for multiple viewers. *IEEE Transactions on Consumer Electronics*, 55(2):927–932, 2009.

[54] Shu-Lin Wang and Chun-Yi Wu. Application of context-aware and personalized recommendation to implement an adaptive ubiquitous learning system. *Expert Systems with Applications*, 38(9):10831–10838, 2011.

[55] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), dec 2015.

[56] Yan-Martin Tamm, Rinchin Damdinov, and Alexey Vasilev. Quality metrics in recommender systems: Do we calculate metrics consistently? In *Fifteenth ACM Conference on Recommender Systems*, RecSys '21, page 708–713. ACM, September 2021.

[57] Evidently AI. Mean Reciprocal Rank (MRR) Explained, 2025. Last accessed: March 13, 2025.

Appendix A

Title of Appendix A

Here goes the appendix A

2nd page of appendix A

Appendix B

Title of Appendix B

Here goes the appendix B

2nd page of appendix B