
UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

# Evaluating Modern Retrieval-Augmented Generation (RAG) Systems

Diploma Thesis

**Apostolos Falaras**

**Supervisor:** Manolis Vavalis

February 2025


UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

# Evaluating Modern Retrieval-Augmented Generation (RAG) Systems

Diploma Thesis

**Apostolos Falaras**

**Supervisor:** Manolis Vavalis

February 2025


ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

# Αξιολόγηση Σύγχρονων Ενισχυμένων με Ανάκτηση (RAG) Συστημάτων

Διπλωματική Εργασία

**Απόστολος Φαλάρας**

**Επιβλέπων/πουσα:** Μανόλης Βάβαλης

Φεβρουάριος 2025

Approved by the Examination Committee:

Supervisor **Manolis Vavalis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Dimitrios Katsaros**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Elias Houstis**
Professor Emeritus, Department of Electrical and Computer Engineering, University of Thessaly

# Acknowledgements

First and foremost, I would like to express my gratitude to my family for always supporting and believing in me throughout the course of the difficult and demanding years of my studies, and for their efforts in ensuring that I always had the best possible conditions in order to accomplish my goals. In addition, I would like to thank my supervisor, Professor Manolis Vavalis, for his valuable contributions and advice that guided and helped me fulfill this thesis. Also, I would like to thank Professor Dimitrios Katsaros and Professor Elias Houstis for kindly agreeing to review and evaluate my thesis.

# DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Apostolos Falaras

Diploma Thesis

**Evaluating Modern Retrieval-Augmented Generation (RAG) Systems**

**Apostolos Falaras**

# Abstract

Artificial Intelligence (AI), and specifically Generative AI systems, has found applications in diverse domains and scientific fields, such as engineering and law. The most prominent applications are Large Language Models (LLM), which are trained on vast amounts of data in order to learn the human language and gain domain-specific knowledge that will help them solve complex knowledge-intensive tasks. However, a limitation of LLMs is the validity of their training data, which can be outdated or not covering certain knowledge domains. Retrieval-Augmented Generation (RAG) is a technique that addresses those issues, by providing knowledge to LLMs, through external databases. A RAG pipeline consists of a retriever, which fetches relevant context passages based on the input query, and a generator, which is a LLM that answers the query with the help of the retrieved context and its internal parametric knowledge. Nonetheless, it's crucial to evaluate the performance of RAG systems, assessing the retriever's and generator's individual capabilities, as well as the end-to-end performance. The evaluation process allows us to inspect RAG systems, and refine their configuration to achieve optimal efficiency. Multiple RAG evaluation frameworks are available on GitHub, with some important examples being RAGChecker, RAGAs, ARES, and AutoRAG. The purpose of this thesis is to utilize those frameworks in the evaluation of a simple RAG application that uses both generic and legal datasets as the external database, and report their results that demonstrate the systems ability to deal with generic and legal data, and understand legal jargon.

## Keywords:

Generative AI, Large Language Models, Retrieval-Augmented Generation, evaluation frameworks, RAGChecker, RAGAs, ARES, AutoRAG

Διπλωματική Εργασία

**Αξιολόγηση Σύγχρονων Ενισχυμένων με Ανάκτηση (RAG) Συστημάτων**

**Απόστολος Φαλάρας**

# Περίληψη

Η Τεχνητή Νοημοσύνη (AI), και ειδικότερα τα συστήματα Παραγωγικής Τεχνητής Νοημοσύνης (Generative AI), έχουν βρει εφαρμογή σε ποικίλους τομείς και επιστημονικά πεδία, όπως η μηχανική και ο νόμος. Οι πιο κυρίαρχες εφαρμογές είναι τα Μεγάλα Γλωσσικά Μοντέλα (LLMs), που προπονούνται σε τεράστιες ποσότητες δεδομένων για να μάθουν την ανθρώπινη γλώσσα και να αποκτήσουν εξειδικευμένες γνώσεις που θα τα βοηθήσουν να επιλύσουν σύνθετες, και απαιτητικές σε γνώση εργασίες. Ωστόσο, ένας περιορισμός των LLMs είναι η εγκυρότητα των δεδομένων προπόνησης, τα οποία μπορεί να παρωχημένα ή να μην καλύπτουν εξειδικευμένους τομείς γνώσης. Η Παραγωγή Ενισχυμένης Ανάκτησης (RAG) είναι μια τεχνική που απευθύνεται σε αυτά τα ζητήματα, παρέχοντας γνώση στα LLMs, μέσω εξωτερικών βάσεων δεδομένων. Ένας αγωγός RAG περιλαμβάνει ένα στοιχείο ανάκτησης, που αναζητά σχετικά με την ερώτηση, αποσπάσματα κειμένου, και ένα στοιχείο παραγωγής, που είναι το LLM που απαντά στην ερώτηση χρησιμοποιώντας τα ανακτημένο περιεχόμενο και την εσωτερική, παραμετρική γνώση του. Παρ'όλα αυτά, είναι κρίσιμο να αξιολογήσουμε την απόδοση των RAG συστημάτων, εξετάζοντας τις ατομικές ικανότητες των στοιχείων ανάκτησης και παραγωγής, καθώς και την συνολική απόδοση. Η αξιολόγηση μας επιτρέπει να διερευνούμε τα RAG συστήματα, και να τροποποιούμε την διαμόρφωση τους για να πετύχουμε την βέλτιστη επίδοση. Πολλαπλά συστήματα αξιολόγησης RAG είναι διαθέσιμα στο GitHub, με σημαντικά παραδείγματα τα RAGChecker, RAGAs, ARES, και AutoRAG. Ο σκοπός αυτής της διπλωματικής είναι η χρήση αυτών των συστημάτων για την αξιολόγηση μιας απλής εφαρμογής RAG που χρησιμοποιεί γενικού σκοπού και νομικά δεδομένα ως εξωτερική βάση, και καταγράφουμε τα αποτελέσματα που επιδεικνύουν την ικανότητα του συστήματος να διαχειριστεί αυτά τα δεδομένα, καθώς και να καταλάβει την νομική γλώσσα.

**Λέξεις-κλειδιά:**

Παραγωγική Τεχνητή Νοημοσύνη, Μεγάλα Γλωσσικά Μοντέλα, Παραγωγή Ενισχυμένης Ανάκτησης, συστήματα αξιολόγησης, RAGChecker, RAGAs, ARES, AutoRAG.

# Table of contents

**Acknowledgements** **ix**

**Abstract** **xii**

**Περίληψη** **xiii**

**Table of contents** **xv**

**List of figures** **xix**

**Abbreviations** **xxi**

**1 Introduction** **1**

1.1 Artificial Intelligence and Law . . . . . 1

1.2 Presentation of the subject . . . . . 2

1.3 Thesis Organization . . . . . 3

**2 Background** **5**

2.1 Large Language Models . . . . . 5

2.1.1 Definition & Brief History . . . . . 5

2.1.2 The Transformer . . . . . 6

2.1.3 The Encoder . . . . . 7

2.1.4 The Decoder . . . . . 8

2.2 Large Language Model Architectures . . . . . 9

2.2.1 Encoder-Decoder Models . . . . . 9

2.2.2 Encoder-Only Models . . . . . 10

2.2.3 Decoder-Only Models . . . . . 10

2.3 Large Language Model Training . . . . . 11
2.3.1 Pre-Training . . . . . 11
2.3.2 Fine-Tuning . . . . . 13
2.4 Retrieval-Augmented Generation . . . . . 15
2.4.1 The Retriever . . . . . 15
2.4.2 Retrieval Models . . . . . 17
2.4.3 The Generator . . . . . 18
2.4.4 Fundamental RAG Approaches . . . . . 19

**3 Literature Review** **21**
3.1 RAGChecker . . . . . 21
3.2 RAGAs . . . . . 24
3.3 ARES . . . . . 26
3.4 AutoRAG . . . . . 27
3.5 BERGEN . . . . . 30
3.6 Tonic Validate . . . . . 31
3.7 Other Frameworks . . . . . 31
3.7.1 RAGLAB . . . . . 31
3.7.2 Trulens . . . . . 32
3.7.3 RagaAI . . . . . 32
3.7.4 Phoenix . . . . . 32
3.7.5 Giskard/RAGET . . . . . 32
3.8 Summary of Research . . . . . 33

**4 Experimental Results** **35**
4.1 Datasets . . . . . 35
4.2 RAG Pipelines . . . . . 37
4.3 Implementation Strategy . . . . . 37
4.3.1 Dataset Preprocessing . . . . . 37
4.3.2 Database Initialization . . . . . 38
4.3.3 Pipeline Testing . . . . . 38
4.4 Evaluation Results . . . . . 40
4.4.1 RAGChecker . . . . . 40

4.4.2 RAGAs . . . . . 42
4.5 Comparison of Evaluation Results . . . . . 43
4.6 Limitations . . . . . 44
4.6.1 Vector Stores & Retrievers . . . . . 45
4.6.2 Generator & Embedding Models . . . . . 45
4.6.3 Evaluation Frameworks . . . . . 45
4.7 Project Directory Structure . . . . . 47

**5 Synopsis and Prospects** **49**
5.1 Future Work . . . . . 50

**Bibliography** **53**

# List of figures

1.1 Simple RAG Pipeline, from the Medium's article "How I built a Simple Retrieval-Augmented Generation (RAG) Pipeline", Dr Julija, 19 Feb 2024 . 2

2.1 Transformer architecture (Source: "Attention is all you need", Vaswani et al., 2017). . . . . 7

4.1 The template for the RAG system's input prompt . . . . . 39
4.2 JSON format of the RAG results of the MS MARCO dataset . . . . . 39
4.3 JSON format of the RAG results of the LegalBench-RAG dataset . . . . . 40
4.4 Input JSON format for the RAG results required by RAGChecker . . . . . 40
4.5 Prompt Template for the creation of the few In-domain examples for ARES 46

# Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| ML | Machine Learning |
| DL | Deep Learning |
| NLP | Natural Language Processing |
| LLM | Large Language Model |
| IR | Information Retrieval |
| HMM | Hidden Markov Model |
| RNN | Recurrent Neural Network |
| CNN | Convolutional Neural Network |
| LSTM | Long Short-Term Memory |
| PII | Personally-Identifiable Information |
| BPE | Byte-Pair Encoding |
| T5 | Text-to-Text Transfer Transformer |
| BART | Bidirectional and Auto-Regressive Transformer |
| BERT | Bidirectional Encoder Representations from Transformer |
| GPT | Generative Pre-trained Transformer |
| LLaMA | Large Language Model Meta AI |
| RoPE | Rotary Position Embeddings |
| PaLM | Pathways Language Model |
| MLM | Masked Language Modeling |
| NSP | Next Sentence Prediction |
| SOP | Sentence Order Prediction |
| DAE | Denoising AutoEncoding |
| RLHF | Reinforcement Learning from Human Feedback |
| RM | Reward Model |

| | |
|---|---|
| RL | Reinforcement Learning |
| PPO | Proximal Policy Optimization |
| RAG | Retrieval-Augmented Generation |
| PPI | Prediction-Powered Inference |
| BLEU | Bilingual Evaluation Understudy |
| ROUGE | Recall-Oriented Understudy for Gisting Evaluation |
| METEOR | Metric for Evaluation of Translation with Explicit ORdering |
| MRR | Mean Reciprocal Rank |
| MAP | Mean Average Precision |
| AP | Average Precision |
| NDCG | Normalized Distributed Cumulative Gain |
| CG | Cumulative Gain |
| DCG | Discounted Cumulative Gain |
| IDCG | Ideal Discounted Cumulative Gain |
| STS | Semantic Text Similarity |
| BERGEN | BEnchmarking Retrieval-augmented GENeration |
| BoW | Bag-of-Words |
| TF-IDF | Term Frequency-Inverse Document Frequency |
| ANN | Approximate Nearest Neighbors |
| DPR | Dense Passage Retriever |
| MMR | Maximal Marginal Relevance |

# Chapter 1

# Introduction

## 1.1 Artificial Intelligence and Law

Artificial Intelligence (AI) [1,2] is a scientific field that aims to develop systems which are capable of replicating human intelligent, leveraging knowledge from mathematics, machine learning (ML), deep learning (DL), natural language processing (NLP), etc. AI has impacted multiple aspects of our lives, extending from basic electronic devices to large-scale industries, like finance, medicine, and law, thereby enhancing productivity and reducing costs [3]. Generative AI, a subcategory of AI, uses ML and DL methods that generate multi-modal content in the form of text, images, audio, and videos [4]. It has been the center of the AI revolution, advancing the workflows across multiple industries, including the legal field.

From one perspective, Generative AI tools [5–7] are capable of assisting legal professionals in legal research, contract review and analysis, and legal document drafting, while making legal services more accessible to clients. Tools, such as CoCounsel [8] and HarveyAI [9], not only have automated iterative and labor-intensive tasks, but also have executed them with higher precision, allowing legal experts to concentrate on more critical issues, such as strategic decision-making for cases.

However, from another perspective [10,11], it has been pointed out that lawyers shouldn't replace their expertise skills with the AI's capabilities, because these tools might generate "hallucinations", that is, fabricated and incorrect responses to legal questions. Moreover, the datasets used to train these tools might be outdated and biased, meaning they don't represent the current legal system, and may prolong social stereotypes that discriminate against certain social groups. In addition, although AI systems can coherently create text using the legal jar-

gon, they are still deprived of the moral and emotional judgement of attorneys and judges, qualities that influence the outcome of many cases, and thus making legal experts irreplacable. Last but not least, we can't forget the data privacy concerns, as most clients would be reluctant to give their personal information to an AI application.

## 1.2 Presentation of the subject

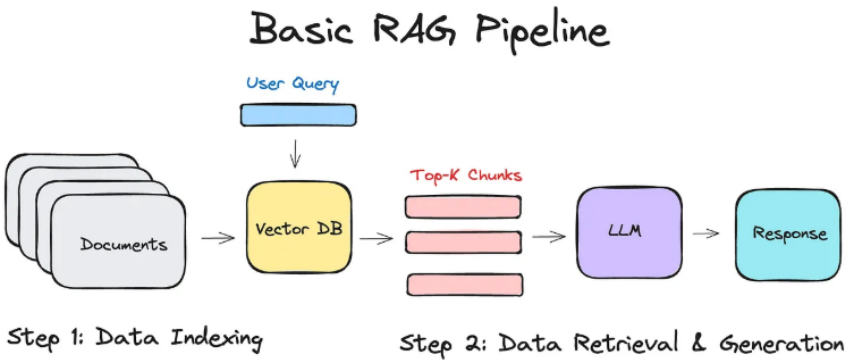


Figure 1.1: Simple RAG Pipeline, from the Medium's article "How I built a Simple Retrieval-Augmented Generation (RAG) Pipeline", Dr Julija, 19 Feb 2024

The purpose of this thesis is to test a collection of Evaluation Frameworks in a Retrieval Augmented Generation (RAG) system that can handle both generic open-domain and legal data. According to [12,13], RAG is an AI framework that combines the generative capabilities of pre-trained Large Language Models (LLMs) with Information Retrieval (IR) models that can search external data repositories, outside of the LLM's training datasets. The aim of RAG is to enhance the general knowledge of the LLMs with the retrieved domain-specific data that are related to a user's input prompt, so that LLMs provide answers that are highly-correlated to the user's request. RAG is an attractive technique that doesn't require any model training, unlike the time-consuming and costly pre-training and fine-tuning processes of LLMs, and can easily be applied to a variety of Generative AI applications [14]. Specifically, the legal domain has incorporated RAG to the Generative AI tools it already uses, to further improve the performance of those models in many legal tasks [15, 16].

## 1.3 Thesis Organization

The rest of this thesis is organized as follows: In Chapter 2, I present the necessary theoretical background of LLMs and RAG, mainly focusing on the fundamental concepts that are related to the motivation of my study. In Chapter 3, I explore a collection of RAG evaluation frameworks, analyzing their retrieval and generation metrics. In Chapter 4, I test some of these tools in a RAG pipeline, that will be tested both on generic and legal datasets. Finally, in Chapter 5 I summarize the contents of this thesis, and make some concluding remarks and draw some conclusions.

# Chapter 2

# Background

## 2.1 Large Language Models

### 2.1.1 Definition & Brief History

Large Language Models (LLMs) are fundamental Generative AI applications that have revolutionized NLP. They're capable of understanding and generating human language in a coherent and context-aware manner, with many corporations already employing them in their operations, mainly through chatbots and virtual assistants [17]. In order for LLMs to grasp the essence of the human language, they have to be pre-trained on massive datasets that consist of a variety of language use cases [18]. Following that, LLMs can be further trained, or fine-tuned, with domain-specific datasets, so that they learn to perform a set of specialized downstream tasks. A notable feature of LLMs is transfer learning [19], where their pre-trained knowledge is expanded, through fine-tuning, to cover a specialized field, such as engineering or law. Moreover, the scale of LLMs [20] is what separates them from other NLP models and is the reason they achieve state-of-the-art performance in many NLP tasks, showcasing a set of "emergent" abilities, such as in-context learning, step-by-step reasoning, etc.

Before exploring LLMs, it is important to review the history of NLP models that leads up to the contemporary LLMs. Early NLP systems were rule-based systems and solely relied on grammatical and syntactical rules, and word definitions, that were configured as input commands. Systems such as ELIZA (1966) [21] and SHRDLU (1972) [22], struggled to adapt to various language contexts. In the 1990s, NLP systems turned to statistical methods, with the development of n-grams and Hidden Markov Models (HMMs), which were used in

tasks, like language generation and Named Entity Recognition (NER). N-gram models [23] learn to predict words based on a preceding sequence of words. HMMs [24] utilize a set of hidden states and a set of observations that are correlated to the occurrence of hidden states.

In the early 2010s, NLP shifted to the neural network, as recurring neural networks (RNNs) and convolutional neural networks (CNNs) were applied to a variety of tasks, such as sentiment classification and textual entailment [25]. Additionally, word embeddings [26], which are continuous vector representations of words, provided a similarity measure among words and phrases, and have significantly reduced the need for feature extraction as part of the data pre-processing step [27], as neural networks can be trained to learn the embeddings of the input text data. Finally, the advent of the Transformer architecture in 2017, introduced a new class of NLP models, including LLMs, that embraced the ”pre-training and fine-tuning” paradigm. In this setting, models are pre-trained on vast corpora of text, in order to learn the grammar and semantics of language, which are captured by the context-sensitive embeddings that form the internal knowledge base of the model. The embeddings can then be adjusted according to the needs of a downstream task.

### 2.1.2 The Transformer

Sequence-to-Sequence modeling transforms an input sequence to an output sequence, and consists of tasks such as machine translation, speech recognition, and text summarization. These models typically follow an encoder-decoder architecture. Initially, RNNs, and especially LSTMs [28], were trained in a supervised setting because they could handle sequences of arbitrary predefined length. An encoder LSTM compresses the information of a sequence into a vector, which a decoder LSTM leverages to produce the output sequence. Additionally, the attention mechanism [29] increased the context window of RNNs, allowing them to calculate vector representations for each sequence element. Meanwhile, although less popular for the task, CNNs [30] enabled the parallel processing of the input sequence, and they could model token relationships in fewer steps depending on the number of convolution layers and the kernel’s size.

However, it was the Transformer [31], a deep neural network encoder-decoder architecture, that surpassed all previous models in terms of performance, as its self-attention mechanism significantly extended its context window, enabling it to capture semantic relationships between sequence elements across substantially greater distances. Below, there’s the image

of the Transformer architecture. The Transformer is constructed as a stack of encoders alongside a stack of decoders. The following presentation of the Transformer's components is based on [31–33], and the equations mentioned belong in [31].

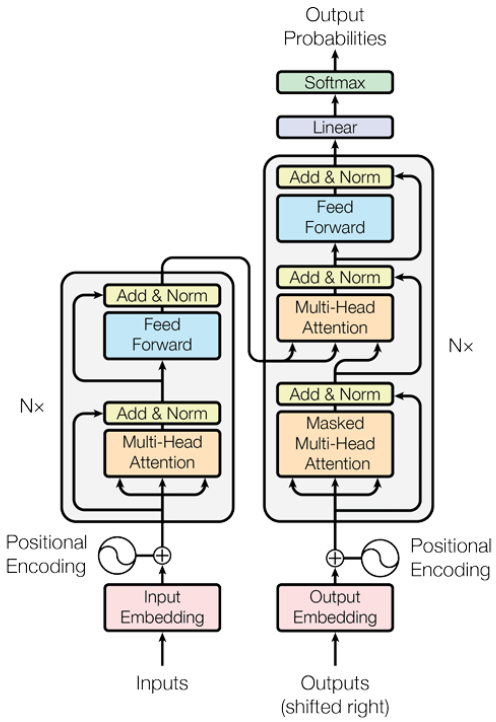


Figure 2.1: Transformer architecture (Source: "Attention is all you need", Vaswani et al., 2017).

### 2.1.3 The Encoder

The encoder consists of a multi-headed self-attention layer followed by a fully-connected layer. Initially, the input needs to be tokenized and the tokens must be encoded into a numerical representation, the embeddings, using an embedding layer. The Transformer ignores the positions of its inputs, unlike RNNs and CNNs, and that's why we must add the following sine and cosine functions as positional embeddings to the token embeddings:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{2.1}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{2.2}$$

where $pos$ is the position of the input token and $i$ is the index of the dimension within the embedding vector.

Next, the input embeddings are fed into the multi-headed self-attention component, which allows each token in the input sequence to attend to all other tokens and quantify the strength (attention) of their relationships. Firstly, for each token, the query ($Q$), key ($K$), and value ($V$) vectors are computed using 3 fully connected layers. The attention score for a token is calculated as the dot product of it's query with the keys of the rest of tokens, scaled by the square root of the dimension of the query/key vectors ($\sqrt{d_k}$) for training stability, and fed into a softmax function that produces the attention weights, which determine which tokens are important (weights close to 1) and which are not (weights close to 0).

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.3}$$

The self-attention mechanism is multi-headed, which means that the query-key-value vector triples will be divided in multiple partitions ("heads"), and self-attention will operate on each partition independently and simultaneously, aiming at capturing complex dependencies in the input data. Finally, the outputs will be concatenated and fed to the pointwise feed-forward network component for additional processing.

$$FFN = ReLU(xW_1 + b_1)W_2 + b_2 \tag{2.4}$$

Both of the aforementioned components use residual connections and apply layer normalization to the sum of the component's input and output. These features enhance training stability and confront the vanishing gradient problem. Layer normalization [34] is formulated as,

$$LayerNorm(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta \tag{2.5}$$

where $\mu$ and $\sigma$ are the mean and variance of $x$, respectively, $\gamma$ is the gain parameter, and $\beta$ is the bias. The final output of the encoder stack is a set of vectors that has thoroughly captured the essence of the input data. This knowledge base will be later used by the decoder in the generation process.

### 2.1.4 The Decoder

The decoder consists of a masked, followed by an encoder-decoder, multi-headed self-attention layer, and a fully-connected layer. Similar to the encoder, the decoder's input gets

embedded, and then augmented with the positional embeddings. The decoder generates the output sequence in an autoregressive manner, using the encoder's knowledge base and the previously generated output tokens. That's why the first self-attention layer employs a "look-ahead" mask that eliminates the attention weights of future tokens with respect to the current token, so that a token can only attend to all previous (and itself) tokens. Moving on, the second self-attention layer uses the queries of the output of the previous layer, and the keys and values from the internal vector representations of the encoder, so that the decoder can attend to the most relevant parts of the latent encoded knowledge. After, the pointwise feed-forward network further processes the output of the previous layer. All three layers utilize residual connections and layer normalization. Finally, the decoder stack concludes with a linear layer followed by a softmax activation that produces the final probabilities for the next token in the sequence over all tokens in the encoded vocabulary.

## 2.2 Large Language Model Architectures

The Transformer has revolutionized the fields of NLP and Generative AI, with almost all publicly available LLMs following the original or some variations of the Transformer architecture.

### 2.2.1 Encoder-Decoder Models

The encoder-decoder models follow the standard Transformer architecture, and some popular models are T5, and BART. Specifically, T5 [35] (Text-to-Text Transfer Transformer) slightly adjusts the original Transformer, by placing the Layer Normalization before the residual connection, and eliminating its bias term. Also, the model uses a different position embedding technique. and implements general "text-to-text" NLP tasks, such as text summarization, sentiment analysis, and machine translation. BART [36] (Bidirectional and Auto-Regressive Transformers) combines a bidirectional encoder (like BERT) and an autoregressive or left-to-right decoder (similar to GPT). BART has substituted the ReLU activations by GeLU activation functions. There are two variations of BART, one using 6 and the other 12 encoder-decoder stacks.

### 2.2.2 Encoder-Only Models

The most well-known encoder-only LLM is BERT [37] (Bidirectional Encoder Representations from Transformer), which is trained to encode the bidirectional latent representations of the input text. BERT's "Base" version (110M parameters) consists of 12 stacked encoders that compute 768-dimensional embeddings and each self-attention layer has 12 heads, while the "Large" version (340M parameters) consists of 24 stacked encoders that compute 1024-dimensional embeddings and each self-attention layer has 24 heads. BERT accepts as input either a single or a pair of sentences, which are represented by their tokens' input embeddings, which are calculated as the sum of the token, segment (which denote the sentence each token belongs to), and positional embeddings (which encode the position of the token in the sequence). Some distinguished BERT variations are RoBERTa [38], DistilBERT [39], and ALBERT [40].

### 2.2.3 Decoder-Only Models

Decoder-only models are distinguished for their generation capabilities, and specialize in tasks such as question-answering, and conversation chatbots. The most prominent LLMs are the series of GPT (Generative Pre-trained Transformer) models. GPT-1 [41] (2018), introduced transfer learning, a technique that pre-trains the LLM's parameters on a large unlabeled corpus, and then further adjusts them using a labeled dataset, so that LLMs can attain a certain level of expertise in a field. GPT-2 (2019) [42], with a size of 1.5B parameters, displayed the ability of LLMs to perform tasks in a zero-shot setting, without additional fine-tuning, by conditioning their outputs on the input text segment and the defined task. GPT-3 [43] (2020), with a size of 175B parameters, demonstrated that significantly increasing the scale of LLMs enhances their text generation capabilities, particularly due to the use of In-Context Learning (ICL). ICL is an emergent ability that allows LLMs to efficiently understand and complete previously unseen tasks (beyond the scope of the pre-training corpus), by conditioning on a set of a few task examples, that are accompanied by the task description in the input prompt. Finally, GPT-4 (2023) featured multimodality, as it's able to accept both text and image inputs. However, the authors of [44] didn't publicize any technical, architectural or training details.

Besides the GPT model series, it's worth mentioning LLaMA (Large Language Model Meta AI) [45], a family of LLMs whose parameter size extends from 7B to 65B parameter.

LLaMA modified the original Transformer, by placing the normalization layer before each transformer sub-layer, and instead of using the LayerNorm function, it uses the RMSNorm function, presented in [46],

$$RMS_{Norm}(x) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{2.6}$$

Also, it substitutes the ReLU activation functions with SwiGLU, and the absolute position embeddings with the Rotary Position Embeddings (RoPE). Similarly, PaLM (Pathways Language Model) [47] employs the SwiGLU activation functions in the feed-forward layers, and utilizes RoPE embeddings. In addition, PaLM has parallelized the execution of each multi-headed self-attention layer with the feed-forward layer of the network. As a result, the output of each transformer block is calculated as,

$$y = x + FFN(LayerNorm(x)) + SelfAttention(LayerNorm(x)) \tag{2.7}$$

## 2.3 Large Language Model Training

The training scheme employed by LLMs, which consists of unsupervised pre-training and supervised fine-tuning [41], was developed to tackle the lack of domain-specific human-annotated data for supervised training, the significant time and cost required to create such datasets, as well as alleviate the need for task-specific model architectures.

### 2.3.1 Pre-Training

Pre-training [18, 20] is the foundational training phase of LLMs, during which they acquire the necessary language skills, which include learning the grammatical and syntactical rules, as well as modeling the usage of language across multiple contexts. The final result of pre-training is the LLM's knowledge base that is encoded and stored in the model's adjusted neural network parameters [41].

#### Data Preprocessing

This process requires massive amounts of text data, and some contextually rich and popular data sources are: Wikipedia pages, CommonCrawl [48] (a repository of web crawled data),

PushShift.io [49] (Reddit conversation collection), the Pile [50] (a large-scale high-quality dataset), and WebText [42] (dataset containing millions of webpages). After the collection process is finished, we need to discard low-quality text, either using a trained LLMs or by a set of predefined rules that filter out text segments based on statistical and keyword features [18, 20]. Also, we must deduplicate text, as data copies [51] have been proven to hurt the training of LLMs, due to the reduced variety of contexts in the pre-training corpus. In addition, the work in [52] stresses the importance of data privacy, which can be achieved by careful anonymization of the Personally-Identifiable Information (PII), which include names, addresses, phone numbers, locations, dates etc.

**Tokenization**

The preprocessing stage concludes with the tokenization of the preprocessing corpus, a procedure that splits the input text data into ”tokens” [18], which can be defined as indivisible units of texts. Some commonly used tokenization techniques are:

- Byte-Pair Encoding (BPE) [53]: It iteratively combines the most frequent sequences of text or bytes, across the corpus, until it reaches the predefined vocabulary size.
- WordPiece [54]: It resembles BPE in the sense that it combines the most frequent tokens, which will maximize the model's likelihood on the training data.
- Unigram [55]: It employs a unigram language model and iteratively optimizes token probabilities, by discarding the least impactful tokens from the vocabulary.

**Pre-Training Objectives**

The main unsupervised pre-training objective that almost all LLMs are trained on is Language Modeling (LM) [20, 41], in which the model attempts to predict the next token based on the previous tokens in a sequence, by calculating the probability distribution across all tokens in the vocabulary.

Formally, given an example sequence of tokens $X = \{x_1, x_2, \ldots, x_n\}$, the LM objective is to maximize the likelihood:

$$L(x) = \sum_i log\ P(x_i | x_{i-k}, \ldots, x_{i-1}; \Theta) \tag{2.8}$$

where $k$ is the LLM's context window, and $\Theta$ represents the model's weight parameters, which are trained using stochastic gradient descent. Language Modeling is the main pre-training objective for decoder only models, such us the GPT series models [41–44], LLaMA [45] and PaLM [47].

On the contrary, encoder-only models, like the BERT model and its variations [37–40] mainly employ Masked Language Modeling (MLM), which masks a randomly selected proportion (15%) of the input tokens with the special $[MASK]$ token, and tries to predict those missing tokens based on the preceding and subsequent context, by minimizing the cross-entropy loss of those tokens, and Next Sentence Prediction (NSP), which examines pairs of sentences and determines whether one of them semantically follows the other. ALBERT substitutes NSP with an improved objective, Sentence Order Prediction (SOP), which evaluates if the order of a pair of sentences is logically correct.

Last but not least, encoder-decoder models, such as T5 [35] and BART [36], utilize Denoising AutoEncoding (DAE) pre-training objective, where the input text gets corrupted and the LLMs attempts to recover the original text. However, the denoising techniques used by T5 and BART are different. T5 simply removes continuous spans of text, while BART performs token masking and removal, text infilling (replacing text spans with the $[MASK]$ token), sentence shuffling and document rotation. The DAE can be mathematically formulated in [20] as the following loss function:

$$L_{DAE}(x) = logP(\tilde{x}|x_{\backslash\tilde{x}}) \tag{2.9}$$

where $x$ is the original input text, $\tilde{x}$ is the set of replaced text spans, and $x_{\backslash\tilde{x}}$ is the corrupt text after performing DAE.

### 2.3.2 Fine-Tuning

Fine-Tuning [18, 20, 41] adapts LLMs to a single or a set of domain-specific tasks, and involves adjusting the pre-trained parameters of the model using a much smaller labeled dataset. This training setting is commonly referred to as transfer learning, as the model builds upon its pre-training knowledge to attain expertise in a particular field. However, we also need to explore two other types of fine-tuning, namely instruction and alignment tuning.

### Instruction Tuning

Instruction Tuning [18, 20] trains LLMs on datasets of instruction instances that teach LLMs how to perform certain tasks. Each example contains the task description, the required inputs and outputs, and a small number of task demonstrations. When it comes to gathering the necessary training instances, researchers have tried collecting training examples from NLP task datasets [56]. In some cases, such as [57], human labelers were assigned to formulate the corresponding prompts for each task. Additionally, given the fact that human annotation is expensive and time-consuming, LLM were assigned to generate a synthetic dataset of instruction-tuning instances, such as SELF-INSTRUCT [58], which is prompted to create new task instructions as well as a few demonstrations for each new task. To sum up, instruction tuning not only equips LLM with complex problem-solving abilities, but also allows them to successfully generalize to previously unseen tasks.

### Alignment Tuning

Alignment Tuning [18, 20] prevents LLMs from generating false information (or hallucinations), biased and toxic content. The primary goal of this process is to produce a LLM that aligns with human ethical and societal values, making an effort to assist users with their queries. In particular, according to [59], LLMs should possess the attributes of helpfulness (willingly trying to resolve the user queries, by providing contextual and effective answers), honesty (transparency about the level of confidence of the generated answers), and harmlessness (not producing harmful content and respond to malicious queries).

Reinforcement Learning from Human Feedback (RLHF) [20,60] is the primary alignment tuning method that initially performs supervised fine-tuning on LLMs, using datasets whose instances are the appropriate LLM responses on various input prompts. These datasets are labeled by human annotators. After that, the annotators rank the LLM responses from best to worst, and a Reward Model (RM) is trained on those response rankings. Finally, RLHF performs Reinforcement Learning (RL) to the pre-trained LLM (that takes the role of the policy in this RL system), against the RM, using an algorithm, such as the popular Proximal Policy Optimization (PPO) [61].

## 2.4 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) [62, 63] is a technique that enhances the knowledge of LLMs by providing information from external data repositories, aiming to confront the lack of up-to-date or domain-specific data that might result in LLMs generating ”hallucinations”, which are fabricated statements that might appear logically supported in the LLM’s response, thus deceiving simple users and professionals. The main components of a RAG pipeline is the retriever, which is responsible for retrieving relevant, to an input query, text passages, and the generator, which is assigned to exploit both the retrieved context and its parametric knowledge in order to resolve the user question.

### 2.4.1 The Retriever

#### External Data Sources

In order to set up and use the retriever module effectively, a series of preprocessing steps must be completed to ensure the efficient storage of the retrieval corpus. The first step in the development process of a RAG pipeline is selecting the appropriate data sources that satisfy an application’s requirements. The most common type of retrieval data is text, which can be datasets for various downstream tasks, like question answering (Natural Questions [64], HotpotQA [65], MS Marco [66]) and language modeling (MMLU [67], WikiText [68]), domain-specific datasets, or internal corporate dataset. Besides text [63], the retrieval corpus can be semi-structured like PDFs, which combine text and image data, or structured such as Knowledge Graphs (KGs), which are compatible with special retrievers (G-Retriever [69]) suitable for querying a database of KGs. Some approaches suggested replacing the retriever module with another LLM that can provide the context for the user questions.

#### Chunking Process

After collecting the retrieval corpus we need to partition it in smaller segments, called chunks [70], a technique aiming to capture a variety of semantic meanings and ideas, in different isolated parts of the text data. However, we should account for the chunking granularity as smaller chunks are faster to process but capture less context, while the encoding for larger chunks is a resource-intensive process. Nonetheless, the selection of the chunks’ size depends on the application area of the retrieval corpus, the subsequent embedding model that

will encode the chunks, and the degree of complexity of user queries.

Some of the available chunking methodologies [71] operate on various text precisions. More specifically, character-based chunking splits the corpus depending on a predefined chunk size, and its recursive version uses a set of delimiter characters (newlines, spaces, etc) to divide text in more contextual partitions. Also, a coarse chunking approach divides the corpus based on the documents' type or structure. Additionally, semantic chunking unifies text passages that carry the same meaning, while agentic chunking employs a LLM that ensures chunks are semantically isolated and independent. To conclude, sub-document chunking produces the summaries each document in the corpus, and use them as a search metadata during the retrieval of chunks.

**Embeddings Creation**

The next step is to create the embeddings [63] for the chunked corpus, which are the vector representations that capture and distinguish each chunk's semantic information. Embeddings can be sparse or dense [70], and have enabled the similarity comparison between their corresponding chunks, which is implemented during the retrieval process. The most common sparse embedding methods are [72]:

- One-hot Encoding: It creates embedding vectors whose dimension is the same as the size of the dataset's vocabulary, and only one of their positions is 1, while the rest are 0.

- Bag-of-Words (BoW): It creates embedding vectors, by replacing words with their occurrence frequency in the corpus, ignoring their order and the text's syntax.

- Term Frequency-Inverse Document Frequency (TF-IDF): It improves upon BoW by calculating the occurrence frequency of words across all the documents of the retrieval corpus.

Dense embeddings, unlike sparse embeddings, consist of mostly non-zero elements and each dimension of the vector can potentially capture a separate feature in the input corpus. Some popular methods [70] for computing them are, the family of BERT models [37–40], which have the capacity to generate contextually rich representations. Also, LLM-based encoders, such as OpenAI's "text-embedding-ada-002", "text-embedding-3-small" and "text-

embedding-3-large" models [73], display powerful representational abilities, by encoding tokens using vectors of dimension 1536, 1536, and 3072, respectively.

#### Indexing

Indexing [70] is a necessary process for large-scale vector databases, as it optimizes the Approximate Nearest Neighbor (ANN) similarity search [74]. Analytically, indexing involves the efficient organization of the chunk embeddings in the vector database that speeds up the comparison between the query and the chunk embeddings, without which similarity search would have been time-consuming and resource intensive, as the query embedding would have to be compared to every chunk embedding in the database. Indexing allows the retrieval of the top-$k$ ($k$ is an arbitrary hyperparameter) pertinent, to the original query, chunks.

### 2.4.2 Retrieval Models

This section delves into various popular retrieval models.

#### BM25

BM25 (Best Match 25) [75, 76] is a foundational IR model that ranks the documents in the retrieval corpus based on their relevance to the input user query. BM25 is suitable for RAG applications, as it, not only analyzes the occurrence frequency of the query's terms in the document corpus, but also accounts for the size of the documents. Although BM25 is an improvement of the traditional TF-IDF method and is excellent for matching words by the query and the corpus, it still struggles with capturing more complex semantic features in the input data.

#### Dense Passage Retriever

Dense Passage Retriever (DPR) [75,77] is a contemporary IR model that, unlike the sparse TF-IDF and BM25 models, creates deep vector embeddings for the documents in the retrieval corpus. The key feature of embeddings is that 2 text passages that use significantly different words, but convey the same meaning, will still be mapped be closely positioned in the multi-dimensional embedding space. This task is accomplished by training 2 encoder networks on a dataset of query-passages pairs. During inference, the semantic similarity between the

query and context passages' embeddings is calculated by the dot product similarity, and DPR retrieves the top-$k$ most similar to query, context passages.

#### ColBERT

ColBERT (Contextualized Late Interaction Over BERT) [78] is another popular retriever model that exploits the thorough representation capabilities of LLMs and employs a ”late interaction” mechanism that allows it to accelerate the retrieval process by pre-computing the corpus embeddings. Specifically, ColBERT utilizes 2 BERT models that independently embed the query and the corpus documents, in the token-level, allowing for deeper and more nuanced comparisons between them. The ”late interaction” functionality is implemented as the sum of the maximum similarity terms of a query and a document. The maximum similarity terms refer to the similarity score between a query token and its most relevant token in the documents.

#### REALM

REALM (Retrieval-Augmented Language Model Pre-Training) [75, 79] is a retrieval-augmented model that adopts the approach of integrating its retrieval functionality in a LLM's pre-training and fine-tuning processes. Particularly, REALM contains 2 key components: a neural knowledge retriever that is trained to predict the probability of identifying and retrieving a document from the corpus, and a knowledge-augmented encoder that is trained to predict the LLM's output based on the query and the retrieved context. The neural retriever is pre-trained on the MLM objective, ensuring that retrieval improves the LLM's capacity to effectively incorporate external information.

### 2.4.3 The Generator

The generator refers to the LLMs, analyzed in Section 2.3, that are prompted to resolve the user question, leveraging it's relevant retrieved context passages. More specifically, LLMs must carefully inspect the context segments and extract the necessary information, if there are any, that address the question, and combine them with any supporting stored internal knowledge that provides more context to the response.

### 2.4.4 Fundamental RAG Approaches

The various RAG approaches can generally be classified into 3 categories: Naive, Advanced, and Modular RAG [63].

#### Naive RAG

Naive RAG [63,80] encompasses the simplest RAG pipeline architectures, which consists of an indexing, a retrieval and a generation (LLM) step, and also referred as the ”retrieve-and-read” RAG framework. Indexing is the process of diving the retrieval corpus into smaller chunks, which are embedded and stored in a vector database. Then, when the RAG system receives an input query, it embeds it and performs similarity search, in order to retrieve the top-$k$ most similar context chunks. Finally, the generation component, receives the input query and the pertinent context chunks and resolves the query combining the retrieved context and its internal parametric knowledge.

However, this RAG implementation may face several difficulties. Specifically, the retriever might struggle detecting the relevant document chunks, due to a number of reasons, such as the employed chunking strategy and the chosen embedding model, or even application fields of the RAG system that contain domain-specific jargon, such as law and medicine. Additionally, the generator might not effectively utilize the retrieved context, leading to the production of fabricated or hallucinated responses.

#### Advanced RAG

Advanced RAG [63,80] is the successor of the Naive RAG paradigm, aiming to deal with its limitations and improve the overall performance of RAG applications, as these type of systems utilize certain pre-retrieval query rewriting techniques and post-retrieval document re-ranking strategies.

In detail, a ”rewrite-retrieve-read” framework [81] was suggested that trains a LLM, in a RL setting, to modify the original queries so that they can adapt to the retriever and enhance the retrieval quality. Also, in [82], the ”least-to-most” prompting method teaches LLMs and RAG systems to divide intricate problems into a sequence of smaller sub-problems. The prompt for each sub-problem contains some demonstration instances for the specific task, the previous questions in the sequence, as well as their answers, which are leveraged by the next query pairs. Furthermore, the ”step-back” prompting strategy [83] generates more generic

queries from the original query, in order to derive foundational knowledge that will assist in the solution of the initial query.

One the other side, post-retrieval re-ranking is a method that rearranges the order in which context passages were retrieved, so that the most critical passages are ranked accordingly. Re-ranking [63] confronts the noise in the retrieved context and ensures the most important chunks are fed to the subsequent LLM, following the model's specified context window limitations. Also, it addresses the ”Lost-in-the-middle” problem [84], a situation where a model's performance drastically decreases when the most significant information is placed in the middle section of the retrieved chunks' sequence. Re-ranking can be performed using the Maximal Marginal Relevance (MMR) [85] metric that attempts to balance the similarity of the retrieved chunks to the query, with the diversity between the retrieved chunks. In addition, a LLM can be employed to re-rank the retrieved context passages [63].

**Modular RAG**

Modular RAG [63, 80] represents an evolution beyond the 2 previous paradigms. In greater detail, Modular RAG allows the integration of new advanced modules that improve the retrieval and generation functionalities. Examples of such modules constitute a search module, like KnowledGPT [86], which enables LLMs and RAG systems to access multiple external data sources in order to effectively tackle complex queries. Also a memory module, like Selfmem [87], employs a retrieval-augmented generator that produces candidate outputs depending on the input and current memory, and a memory selector that is trained to evaluate those outputs, in order to iteratively expand the retrieval memory with high-quality generated outputs.

In addition, Modular RAG enables developers to create and test novel RAG pipelines, by combining different modules, and allows them to trace and isolate errors within individual modules, thus simplifying the debugging and optimization processes. More specifically, the ”Generate-then-Read” [88] RAG pipeline utilizes a context generator-LLM that transforms the input query into retrieval context, which will be later fed, along with the query, to the generator of the RAG pipeline. Additionally, the ”Demonstrate-Search-Predict” [89] framework executes 3 operations: i) Demonstration, which prompts the pipeline's LLM to generate demonstration examples for the query, ii) Search, the typical retrieval step, and iii) Predict, the typical generation step that also leverages the task demonstrations.

# Chapter 3

# Literature Review

The purpose of this section is to present the results of the extensive research that was conducted on the evaluation frameworks available for RAG systems. In particular, I will highlight the key features and evaluation metrics of each framework, and also present practical usage details, such as, availability, licensing, and open-source status. This section also provides the necessary tables that summarize the comparative analysis of the evaluation frameworks.

## 3.1 RAGChecker

RAGChecker [90] is a framework that performs a thorough evaluation process, including a variety of metrics that assess the retriever's and generator's functionality, as well as overall metrics that evaluate the end-to-end performance of RAG systems. RAGChecker operates at the granularity of individual claims, which are derived from the system's generated response and the ground-truth answer, thus treating a singular claim as the evaluation unit to compute the metrics. In order to obtain the claims, a "text-to-claim" module is employed to extract atomic claims from text, and a "claim-entailment" module verifies if a generated claim also belongs in the ground-truth. The equations and mathematical notation used in the following paragraphs belong in the original paper and the corresponding Github repository [90, 91], with the exception of the F1 score, which I provided.

The overall metrics are precision, recall and F1 score. Precision is defined as the fraction of correct generated claims $c_i^{(m)}$ in the generated response $m$. A claim is correct if it can inferred from the ground-truth answer $gt$. Recall is defined as the fraction of ground-truth claims $c_i^{(gt)}$ that can be found in the model response, and F1 score is the harmonic mean of

precision and recall.

Table 3.1: RAGChecker - Overall Metrics

| **Metric** | **Formula** |
|---|---|
| Precision | $\frac{\lvert\{c_i^{(m)} \lvert c_i^{(m)} \in gt\}\rvert}{\lvert\{c_i^{(m)}\}\rvert}$ |
| Recall | $\frac{\lvert\{c_i^{(gt)} \lvert c_i^{(gt)} \in m\}\rvert}{\lvert\{c_i^{(gt)}\}\rvert}$ |
| F1 Score | $2\frac{Precision \cdot Recall}{Precision + Recall}$ |

The retriever metrics are claim recall and context precision. Firstly, Claim Recall is the fraction of ground-truth claims $c_i^{(gt)}$ that can be found in the set of retrieved chunks $\{chunk_j\}$. Secondly, Context Precision is the fraction of relevant chunks $\{r\text{-}chunk_j\}$ from the $k$ retrieved chunks. A chunk is considered relevant when at least one ground-truth claim can be inferred from it. At this point it's worth noting that context precision is calculated using the number of relevant chunks, and not claims, because even a relevant chunk will probably contain additional irrelevant information.

Table 3.2: RAGChecker - Retriever Metrics

| **Metric** | **Formula** |
|---|---|
| Claim Recall | $\frac{\lvert\{c_i^{(gt)} \lvert c_i^{(gt)} \in \{chunk_j\}\}\rvert}{\lvert\{c_i^{(gt)}\}\rvert}$ |
| Context Precision | $\frac{\lvert\{r\text{-}chunk_j\}\rvert}{k}$ |

The generator metrics are faithfulness, relevant and irrelevant noise sensitivity, hallucination, self-knowledge and context utilization. Faithfulness is the fraction of the model-generated claims $c_i^{(m)}$ that can be attributed to retrieved chunks, and it indicates the level of accuracy with which the model depends on the retrieved context for generation. Relevant and

Irrelevant Noise Sensitivity refer to the fraction of generated claims $c_i^{(m)}$ that are incorrect, but can be extracted from relevant and irrelevant retrieved chunks, respectively. The set of irrelevant retrieved chunks is denoted as $\{irr\text{-}chunk_j\}$. These two metrics quantify the extent to which the model is affected by noise (out-of-context data), when that noise, in the first case, is accompanied by useful context and, in the second case, is accompanied by useless context. Moreover, Hallucination is the fraction of generated claims $c_i^{(m)}$ that belong neither in the ground-truth answer $gt$ nor in any retrieved chunk. It's simply the set of false statements that the model generated, depending on its knowledge base. Furthermore, Self-Knowledge describes the fraction of generated responses $c_i^{(m)}$ that can be traced in the ground-truth $gt$ but not in any retrieved chunk, meaning that the model used its internal knowledge to produce such claims. This metric expresses the dependence of the model on its own knowledge, and maybe the ignorance towards the retrieved data. Lastly, Context Utilization is defined as the fraction of ground-truth claims $c_i^{(gt)}$ that can be found in the set of retrieved chunks, that can also be extracted from the generated response $m$. This metric quantifies the usage of the retrieved relevant context in the generated response $m$.

Table 3.3: RAGChecker - Generator Metrics

| **Metric** | **Formula** |
|---|---|
| Faithfulness | $\frac{\lvert\{c_i^{(m)} \lvert c_i^{(m)} \in \{chunk_j\}\}\rvert}{\lvert\{c_i^{(m)}\}\rvert}$ |
| Relevant Noise Sensitivity | $\frac{\lvert\{c_i^{(m)} \lvert c_i^{(m)} \notin gt \text{ and } c_i^{(m)} \in \{r\text{-}chunk_j\}\}\rvert}{\lvert\{c_i^{(m)}\}\rvert}$ |
| Irrelevant Noise Sensitivity | $\frac{\lvert\{c_i^{(m)} \lvert c_i^{(m)} \notin gt \text{ and } c_i^{(m)} \in \{irr\text{-}chunk_j\}\}\rvert}{\lvert\{c_i^{(m)}\}\rvert}$ |
| Hallucination | $\frac{\lvert\{c_i^{(m)} \lvert c_i^{(m)} \notin gt \text{ and } c_i^{(m)} \notin \{chunk_j\}\}\rvert}{\lvert\{c_i^{(m)}\}\rvert}$ |
| Self-Knowledge | $\frac{\lvert\{c_i^{(m)} \lvert c_i^{(m)} \in gt \text{ and } c_i^{(m)} \notin \{chunk_j\}\}\rvert}{\lvert\{c_i^{(m)}\}\rvert}$ |
| Context Utilization | $\frac{\lvert\{c_i^{(gt)} \lvert c_i^{(gt)} \in \{chunk_j\} \text{ and } c_i^{(gt)} \in m\}\rvert}{\lvert\{c_i^{(gt)} \lvert c_i^{(gt)} \in \{chunk_j\}\}\rvert}$ |

## 3.2 RAGAs

RAGAs [92] is another RAG evaluation framework that focuses on the three foundational metrics for RAG evaluation - faithfulness, answer relevance, and context relevance - but adopts a different method for their computation, that is applying a set of prompts to a LLM. The mathematical notations and formulas of the most significant metrics provided in this subsection, belong in [92].

To elaborate, for the calculation of Faithfulness, which describes the degree to which the generated answer is based on the retrieved context, a LLM is prompted to derive a set of claims $S(\alpha_s(q))$ from a generated answer $\alpha_s(q)$, where $q$ is the initial question. Then, the LLM is prompted to examine if every claim $s_i$ can be inferred from the retrieved context $c(q)$. As a result, faithfulness is the fraction of inferred claims $|V|$ from the total claims $|S|$.

Furthermore, for the computation of Answer Relevance, which tests if the generated response sufficiently handles the input question, the same LLM is prompted to produce a sequence of $n$ questions, $\{q_i\}$, that could be addressed by the generated answer $\alpha_s(q)$. Thereafter, the set of candidate questions are embedded and the cosine similarity of each to the original question embedding is computed, $sim(q, q_i)$. Finally, answer relevance can be defined as the average of the cosine similarities.

Moreover, for the estimation of Context Relevance/Recall, which examines the degree to which the retrieved passages include the necessary context to handle the input question, the LLM is prompted to produce a set of claims $S_{ext}$, from the context $c(q)$, that are required to answer the question. Consequently, context relevance is the fraction of the sentences in the context the successfully confront the question.

Table 3.4: RAGAs Evaluation Metrics - (1/2)

| **Metric** | **Formula** |
|---|---|
| Faithfulness | $\frac{\|V\|}{\|S\|}$ |
| Answer Relevance | $\frac{1}{n}\sum_{i=1}^{n} sim(q, q_i)$ |

| | |
|---|---|
| Context Relevance/Recall | $\frac{\lvert\text{Number of extracted sentences in } S_{ext}\rvert}{\lvert\text{Number of sentences in } c(q)\rvert}$ |

Besides those metrics, RAGAs' Github repository [93] as well as its documentation website [94] introduce additional RAG evaluation metrics as well as general language comparison metrics. Firstly, Context Precision@K is introduced as the fraction of contextually relevant retrieved chunks, and is computed as the average of Precision@k for all $k$ values (which represent the number of retrieved chunks) up to $K$. Secondly, a variation of context recall this is Context Entity Recall that examines how many ground-truth entities (names, locations, dates, and more) were present in the generated answer. Moreover, Noise Sensitivity is implemented as the equivalent metric in RAGChecker, accounting for both cases where the model responds with false claims that were derived from relevant and irrelevant retrieved chunks.

Table 3.5: RAGAs Evaluation Metrics - (2/2)

| **Metric** | **Formula** |
|---|---|
| Context Precision@$K$ | $\frac{\sum_{k=1}^{K}(Precision@k \times v_k)}{\lvert\text{Relevant Items in the top } K \text{ results}\rvert}$ |
| Context Entity Recall | $\frac{\lvert(\text{Retrieved-Context Entities}) \cap (\text{Ground-Truth Entities})\rvert}{\lvert(\text{Ground-Truth Entities})\rvert}$ |
| (Relevant/Irrelevant) Noise Sensitivity | $\frac{\lvert\text{Incorrect Claims in generated answer}\rvert}{\lvert\text{Total number of Claims in the generated answer}\rvert}$ |

On the other side, the general language measurements include the mainstream precision, recall, and F1 score metrics, which are evaluated on the generated answer's statements in comparison to the ground-truth answer. These 3 metrics are grouped together in the Factual Correctness metric of the GitHub page. Additionally, the Semantic Similarity score utilizes the embeddings of the response and the reference answer, and a set of metrics which don't need the contribution of LLMs are presented below. These are:

- Non-LLM Semantic Similarity: It computes the similarity (on a scale of 0 to 1) between the model response and the ground-truth answer using distance measurements such as the Hamming, Levenshtein, and Jaro distances.

- BLEU (Bilingual Evaluation Understudy) Score: It computes the similarity (on a scale of 0 to 1) between the model response and the ground-truth answer using the n-gram precision.
- ROUGE (Recall-Oriented Understudy for Gisting Evaluation) Score: It computes the fraction of overlap between the generated and reference response based on the n-gram precision, recall and F1 score.
- Exact March: A binary metric that tests whether the generated response is exactly the same (1), or not (0), as the ground-truth answer.
- String Presence: A binary metric that checks whether the generated response contains (1), or not (0), several parts or keywords of the reference answer.

## 3.3 ARES

ARES [95, 96] is an automated framework for evaluation of RAG applications that employs compact language models to perform the synthetic dataset generation and RAG assessment processes. In particular, the first step in the ARES pipeline is the creation of a synthetic dataset, using a LLM that receives a set of input text segments and a set of few-shot instances, and it learns to generate question-answer pairs. Each questions appears in multiple pairs, one containing the correct answer and several other containing incorrect answers, resulting in dataset that includes both positive and negative examples. Also, ARES discards queries that don't have their corresponding segment returned as the top choice by the retriever. Afterwards, the synthetic dataset instances of query-passage-answer are used to train three LLMs, also called judges, so that they learn to estimate the framework's evaluation metrics on previously unknown examples. The evaluation metrics are:

Table 3.6: ARES Evaluation Metrics

| **Metric** | **Description** |
|---|---|
| Context Relevance | Tests if the returned context is relevant to the input query. |

| | |
|---|---|
| Answer Faithfulness | Tests if the generated answer is grounded on the retrieved context or if it contains hallucinations. |
| Answer Relevance | Tests if the generated answer is contextually relevant to the query and retrieved context. |

At this point, it's significant to mention that the feature that distinguishes ARES from other frameworks is its Prediction-Powered Inference (PPI) mechanism, which uses another input dataset, the human preference validation set. This method calculates the metrics on the validation set in order to obtain a "rectifier" function, which is capable of mitigating the errors and biases of the LLM judges, and calculates confidence intervals for the estimated metric values on the synthetic dataset examples, thereby providing more robust evaluations.

## 3.4 AutoRAG

AutoRAG is another open source RAG evaluation framework [97] that not only offers a variety of evaluation metrics for the retriever and generator, but also provides users with tools for all components of the RAG pipeline, such as different retrievers (BM25, VectorDB, HubridRFF, HybridCC), rerankers (URP, Tart, MonoT5 and more), and generators (llama index llm, openai llm, vllm). The metrics are defined in both the GitHub repository [98] and the documentation website [99]. In more detail, AutoRAG's retrievers are also assessed using the precision, recall, and F1 score in terms of the generated and ground-truth claims, and these metrics are also provided in versions that check the generated and ground-truth tokens. These metric variations are used by the passage compressor component.

Moving on, the framework introduces three retrieval metrics that are not defined by the previously analyzed frameworks. Firstly, AutoRAG uses Mean Reciprocal Rank (MRR), which is the average of the reciprocal ranks - the inverse of ranks (index positions) of the first relevant retrieved chunk in the top-$k$ list of chunks of all user queries, $U$. In addition, it uses Mean Average Precision (MAP), which is calculated as the average of the Average Precision (AP) terms across all user queries, $U$, each of which is the average of the precision for all the positions up to which a new relevant chunk was retrieved in the top-$k$ list of chunks.

The formulas for MRR and MAP were derived from [100]. Finally, the framework employs Normalized Distributed Cumulative Gain (NDCG). However, to define this metric we need to compute three other quantities first:

- Cumulative Gain (CG): CG is the sum of the relevance of the top-$k$ retrieved chunks.

$$CG = \sum_{i=1}^{k} rel_i \tag{3.1}$$

- Discounted Cumulative Gain (DCG): The downside of CG is that multiple sequences can have the same CG value, if the retrieved chunks are retrieved in a different order. That's why DCG divides every relevance term, $rel_i$, by their logarithmic rank, so as to mitigate the effect of the relevance of chunks retrieved later in the top-$k$ sequence.

$$DCG = \sum_{i=1}^{k} \frac{rel_i}{log_2(i+1)} \tag{3.2}$$

- Ideal Discounted Cumulative Gain (IDCG): IDCG is the most desirable value of DCG.

Table 3.7: AutoRAG Retrieval Metrics

| **Metric** | **Formula** |
|---|---|
| Mean Reciprocal Rank | $\frac{1}{U}\sum_{u=1}^{U}\frac{1}{rank_u}$ |
| Mean Average Precision | $\frac{1}{U}\sum_{u=1}^{U} AP@K_u$ |
| Normalized Distributed Cumulative Gain | $\frac{DCG_k}{IDCG_k}$ |

AutoRAG's generator metrics comprise both traditional evaluation metrics as well as semantic-based metrics. In greater depth, the framework utilizes BLEU, to test the overlap precision of n-grams in the generated response that are also present in the ground-truth answer, and ROUGE, to test the overlap recall of n-grams on the ground-truth answer that are also present in the generated response.

In a similar context, METEOR (Metric for Evaluation of Translation with Explicit ORdering) [101] operates at the unigram level, and requires the execution of several steps in

order to be computed. In particular, a series of ”alignments” (sets of one-to-one mappings) is computed, linking unigrams in the generated and ground-truth answers. There are three categories of alignments, determined by the matching criteria applied on unigrams: exact word matches, word stem matches, and synonym words matches. Next, the biggest subset of each alignment is selected, and after several optimization steps a final alignment is derived, from which the unigram precision $P$, recall $R$, and F1 score are calculated. F1 score significantly weights the recall term by a factor of 9:

$$F_{mean} = \frac{10PR}{P + 9R} \tag{3.3}$$

METEOR applies a penalty term that takes into account cases where the matching unigrams in the generated response are to a certain degree scattered, while they form a sequence in the ground truth answer. The higher the degree of fragmentation in the generated unigrams, the higher the penalty applied to the METEOR score. The penalty is formulated as:

$$Penalty = 0.5 \cdot \left( \frac{\#chunks}{\#unigram\_chunks} \right) \tag{3.4}$$

In conclusion, the METEOR Score is declared as:

$$METEOR_{Score} = F_{mean} \cdot (1 - Penalty) \tag{3.5}$$

However, unlike (n-gram)-based metrics whose evaluations might disagree with the labels given by human annotators, SemScore calculates the Semantic Text Similarity (STS) between the generated response and the ground-truth reference. In this process, the two answers are embedded, using a pre-trained model, and their cosine similarity is calculated, which ranges from -1 (completely inverse semantic meaning) to 1 (identical semantic meaning). In similar context, BERTScore [102] tokenizes and embeds the generated response, $\mathbf{x}$, and ground-truth answer, $\hat{\mathbf{x}}$. The contextual embeddings vectors are then normalized and their dot product is utilized to calculate precision, recall, and F1 score as follows:

Table 3.8: BERTScore Metrics

| Metric | Formula |
|---|---|
| Precision | $P_{BERT} = \frac{1}{\lvert x \rvert} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x_i^T \hat{x}_j}$ |
| Recall | $R_{BERT} = \frac{1}{\lvert \hat{x} \rvert} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^T \hat{x}_j$ |
| F1 Score | $F1_{BERT} = 2\frac{P_{BERT} \cdot R_{BERT}}{P_{BERT} + R_{BERT}}$ |

Last but not least, AutoRAG employs the G-Eval with GPT-4 [103] evaluation framework, which uses specialized prompts to address different NLP tasks. The prompts provide the task description, a list of the instruction steps that will guide the LLM towards the solution of the problem (also defined as a Chain-of-Thoughts (CoT)), and the evaluation metrics it needs to produce as a result. GPT-4 can assess the coherence and fluency of the generated response, as well as its consistency and relevance in terms of the question and the retrieved context.

## 3.5 BERGEN

BERGEN (BEnchmarking Retrieval-augmented GENeration) [104,105] is a RAG library that resembles AutoRAG, in the sense that it equips users with multiple retrievers (BM25, RetroMAE, v2, RepLlama, etc), rerankers (MiniLM, DeBERTa-v3, BGE(-M3)), generator LLMs (Llama2, Llama3, SOLAR, Mixtral, etc), and evaluation datasets (Natural Questions, TriviaQA, HotpotQA, etc).

BERGEN utilizes two categories of metrics that assess both the matching of words and phrases, and the semantic meaning of the generated response. On one hand, Match, Exact Match, Precision, Recall, F1 score, ROUGE-1, ROUGE-2, ROUGE-L analyze the mapping of words and sentences in the generated and ground-truth answer. On the other hand, BEM (Bert Matching), GPT-4 and LLMEval (employing the SOLAR-10.7B-Instruct-v1.0 model) address the lack of semantic analysis of the first category of metrics. Moreover, it employs

augmentation precision and recall, which verify the existence of the retrieved context in the LLM-generated response.

## 3.6 Tonic Validate

Tonic Validate [106] is a RAG evaluation tool, offering a series of metrics that address multiple aspects of the system's generated responses. In greater detail, it evaluates answer consistency, which calculates the proportion of generated claims that can be attributed to the retrieved context, or checks their existence as binary metric. Also, it parses the generated answer, searching for PII elements, checks if it matches the ground-truth answer and computes their semantic similarity.

Additionally, Tonic Validate includes metrics that addresses non-semantic features of the generated response. Specifically, it checks the retrieved context's and response's length, verifies that the retrieved context doesn't contain PII. Furthermore, it scans the generated text for hateful content, duplicate data, as well as the generation latency of the response.

## 3.7 Other Frameworks

Some other notable frameworks that I came across during my research were RAGLAB [107, 108], Trulens [109], RagaAI [110], Phoenix [111], and Giskard [112]. Generally, all of these frameworks ensure that a RAG system will be evaluated in terms of its output's answer faithfulness and relevance, as well as the context relevance of the retrieved text passages, in relation to the user question. Nevertheless, it's worth mentioning that some of these toolkits extend their assessment functionalities beyond the three aforementioned and foundational evaluation metrics.

### 3.7.1 RAGLAB

RAGLAB is a general RAG research framework that enables developers to easily experiment with various RAG pipeline configurations, as it provides a series of retrievers and generators, as well as various benchmarks and utility functions that assist users in their research. Regarding the evaluation process, RAGLAB employs 3 classic metrics, which are F1 Score, Exact Match, and Accuracy, and 2 advanced metrics, which are Factscore, and ALCE.

Factscore [113] evaluates the factual accuracy of the generated response, and ALCE [114] is a benchmark that evaluates the citation accuracy and precision of RAG systems.

### 3.7.2 Trulens

Besides the 3 foundational RAG metrics (also known as RAG Triad), Trulens provides a suite of metrics [115] that evaluate general LLM applications in terms of their capacity to be honest (Answer and Context Relevance, Groundedness, Embedding distance etc), helpful (Prompt Sentiment, Language Mismatch, Conciseness, Coherence etc), and harmless (PII Detection, Toxicity etc).

### 3.7.3 RagaAI

RagaAI extends the fundamental RAG evaluation capabilities to checking for cases of False Refusal, instances where the LLM refuses to answer the query, even though the retrieved context provides the solution to it, and for cases of Toxicity, instances of harmful generated responses from the LLM of the pipeline. Additionally, it parses the generated response, searching for PII data.

### 3.7.4 Phoenix

Phoenix assesses the retrieval and generation functions, using basic metrics, such as precision, NDCG, and hit (binary metric that checks whether at least 1 relevant context passage was retrieved for a given query) for retrieval, and answer correctness and hallucination for generation.

### 3.7.5 Giskard/RAGET

Giskard provides the RAGET evaluation toolkit that evaluates the performance of the different components of the RAG pipeline, the retriever, the generator, the external knowledge base, and if present in the pipeline, the rewriter and router components. The output is a percentage score reflecting the performance of each component, as well as a recommendation section, suggesting possible improvements to the pipeline.

## 3.8 Summary of Research

In conclusion, in order to enrich the quality of my research on the available RAG evaluation frameworks, the following table summarizes the key capabilities and features of the aforementioned frameworks:

- **Year**, **License**, main **Programming Language**, and the **Latest Update date (with respect to the time this thesis was written)**, as specified in each framework's GitHub page.
- **Faithfulness Evaluation** and **Hallucination Detection** - Features that capture the set of different generation evaluation metrics, which mainly focus on the evaluation of the response's faithfulness to the generated context, and the detection of hallucinated claims produced by the LLM of the RAG pipeline.
- **Retrieval Assessment** - Feature that captures the set of different retrieval evaluation metrics.
- **LLM-based Evaluation** - Feature that checks whether or not the framework employs LLM-based evaluation metrics.
- **Traditional Evaluation** - Feature that checks whether or not the framework uses traditional metrics, such as precision, recall, F1 Score, BLEU, ROUGE, etc
- **Explainability** - Feature that checks whether or not the framework provides insights into the reasoning behind the evaluation results, by pinpointing the source of performance degradation in the RAG pipeline.
- **Dataset Scalability** - Feature that checks whether the framework can process large datasets.
- **Open-source** - Feature that checks if the framework is open-source.
- **Custom Metrics** - Feature that checks if the framework allows users to define application-specific metrics.
- **Dataset Agnostic** - Feature that checks whether the framework accepts datasets of various domains or requires domain-specific datasets.

- **UI/Dashboard Support** - Feature that checks whether or not the framework provides a UI application to users.
- **RAG Tools** - Feature that checks if the framework provides general RAG tools, such a set of retrievers, rerankers, and generators, and even test datasets, that can be combined in multiple ways that allow users to experiment and optimize RAG pipelines.

Table 3.9: Overview features of the examined RAG frameworks

| Feature | RAGChecker | RAGAs | ARES | AutoRAG | BERGEN | Tonic Validate | RAGLAB | Trulens | RagaAI | Phoenix | Giskard/RAGET |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Year** | 2024 | 2023 | 2024 | 2024 | 2024 | 2023 | 2024 | 2023 | 2024 | 2023 | 2023 |
| **License** | Apache-2.0 | Apache-2.0 | Apache-2.0 | Apache-2.0 | CC BY-NC-SA 4.0 | MIT | MIT | MIT | Apache-2.0 | Elastic License 2.0 | Apache-2.0 |
| **Programming Language** | Python | Python | Python | Python | Python | Python | Python | Python | Python | Python, Typescript | Python |
| **Latest Update** | Sep. 25, 2024 | Jan. 22, 2025 | - | Jan 25, 2025 | Oct 10, 2024 | Nov 14, 2024 | - | Jan 27, 2025 | Jan 23, 2025 | Jan 31, 2025 | Nov 22, 2024 |
| **Faithfulness Evaluation** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Hallucination Detection** | ✓ | ✓ | ✓ | ✓ | - | ✓ | - | ✓ | ✓ | ✓ | ✓ |
| **Retrieval Assessment** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **LLM-based Evaluation** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ |
| **Traditional Evaluation** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Explainability** | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Dataset Scalability** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Open-source** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Custom Metrics** | - | ✓ | - | - | - | - | - | ✓ | - | ✓ | - |
| **Dataset Agnostic** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **UI/Dashboard Support** | - | ✓ | - | ✓ | - | ✓ | - | ✓ | ✓ | ✓ | ✓ |
| **RAG Tools** | - | - | - | ✓ | ✓ | - | ✓ | - | - | - | - |

# Chapter 4

# Experimental Results

## 4.1 Datasets

My RAG system will be evaluated on a generic and a legal dataset. On one side, MS MARCO (Microsoft MAchine Reading COmprehension) [66] is a large general-purpose dataset, containing factual user queries that are derived from Bing's public search logs. This dataset is a great choice for benchmarking retrieval and RAG-based systems, because it examines 3 aspects of the system:

- It evaluates whether the system retrieved useful text segments for answering the initial query or not, and if possible, the system should generate an answer.
- It assesses whether the system can generate a coherent and context-aware answer (or "well-formed") based on the retrieved text segments.
- It inspects the way the system ranks the retrieved text segments.

I used the dataset's "v1.1" version, and its instances are composed of the following attributes:

1) The user query ("query").
2) The query's numerical identifier ("query_id").
3) The query's category ("query_type").

4) The passages dictionary, which, on average, contains 10 text segments in a list (”passage_text”), with the corresponding url source (”url”). Human labelers annotate which text segments contributed to the query’s answer (1) and which didn’t (0).

5) The answer to the query, which has been produced by human labelers that have investigated the retrieved text segments.

6) The ”well-formed” answers, which are produced by human labelers, as an effort to rephrase and improve the system’s generated answer.

On the other side, LegalBench-RAG [116] is the first legal benchmark that assesses RAG systems on their capacity to understand legal jargon and retrieve pertinent (to a question) legal documents. The benchmark focus on precise retrieval by extracting the answer spans from the legal documents. LegalBench-RAG compiles legal documents from 4 datasets:

- Privacy Question Answering (PrivacyQA) [117]: PrivacyQA consists of 1,750 questions about the contents of privacy policies of mobile applications, with over 3,500 annotations from experts.

- Contract Understanding Atticus Dataset (CUAD) [118]: CUAD consists of more than 500 legal documents and more than 13,000 annotations, made by legal experts that are members of the Atticus Project (a non-profit organization of legal experts). The documents extend up to 41 label categories.

- Merger Agreement Understanding Dataset (MAUD) [119]: MAUD is a reading comprehension dataset including over 39,000 examples and over 47,000 annotations, originating from the American Bar Association’s 2021 Public Target Deal Points Study.

- Contract Natural Language Inference (ContractNLI) [120]: ContractNLI contains 607 legal contracts and addresses the contract review automation task. The system is assigned to figure out whether a set of hypotheses can be entailed in a specific document.

The 4 datasets sum up to 698 legal text documents, and 6,889 question-answer pairs, which are stored in 4 JSON files and each QA instance consists of:

1) The input query (”query”).

2) A list of answer snippets (”snippets”), with each snippet containing:

i) The path to the corresponding document (”file_path”).

ii) The index span containing the correct answer (”span”).

iii) The answer contained in the indexed span (”answer”).

## 4.2 RAG Pipelines

For the evaluation experiments, I utilized 2 RAG pipelines, 1 for each dataset. In more detail, both pipelines employed the same generator model, ”gpt-4o-mini” [121], which is a small, fast, and affordable model, whose cost and low latency has enabled the execution of applications that require a significant amount of API calls and feed large datasets to the model. This LLM has a context window of 128,000 tokens, and the maximum tokens accepted per request, are 16,384.

However, the vector store component, which is also configured to act as the retriever during the testing process, is dependent on the input dataset. More specifically, the MS MARCO’s pipeline employs the Chroma [122] vector database, while the LegalBench-RAG’s pipeline utilizes the SQLiteVec [123] vector database. Chroma was the initial choice for storing the chunk embeddings of both datasets, but when I tested the pipeline on LegalBench-RAG, almost all of the queries didn’t receive a definite answer, as the LLM responded with ”I don’t know.”. In addition, the retrieved context passages were partially relevant, as a portion of them originated from the correct source file, but their content didn’t address the original query, and the rest were completely irrelevant.

For the development of the pipelines, I used LangChain [124], a Python framework that provides tools for building LLM-based applications, including RAG systems. LangChain offers open-source components, and a series of third-party integrations that include LLMs (OpenAI, HuggingFace, Anthropic, Cohere, AWS, etc), and vector stores (Chroma, SQLiteVec, FAISS, Pinecone, etc).

## 4.3 Implementation Strategy

### 4.3.1 Dataset Preprocessing

On one side, the MS MARCO dataset consists of a training set (82,326 instances), a validation set (10,047 instances), and a test set (9,650 instances), but I only selected the train-

ing portion of the dataset. For every training example, I converted each context segment of the passages dictionary, into a LangChain 'Document' object and stored them in a list that represents the retrieval corpus. Due to the inherent organization of the context passages, I decided to consider each one of them, a chunk that will later be embedded and stored in the Chroma vector database. After the preprocessing, the retrieval corpus consisted of 676,193 Documents or chunks.

On the other side, the LegalBench-RAG dataset consists of 698 text legal documents, and 6,889 question-answer instances. Similar to the first case, I stored each one of the legal texts as a LangChain 'Document' object. Unlike MS MARCO whose question-answer pairs are accompanied by the context passages, I organized the question-answer pairs of LegalBench-RAG in a list structure, extracting them from their corresponding JSON files.

Additionally, I calculated the average size of the legal documents to be 105,009 characters. Considering that gpt-4o-mini has a context window of 128,000 tokens, and that instructing the pipeline to retrieve the top-10 most similar context chunks for a query will likely exceed the context window limit, I decided to split the retrieval corpus into chunks of 2,000 characters with a 500-character overlap. The chunking process is carried out by LangChain's "RecursiveCharacterTextSplitter". After chunking, the retrieval corpus contains 58,387 Document chunks.

### 4.3.2 Database Initialization

For the encoding of the corpus chunks, both RAG pipelines utilize OpenAI's "text-embedding-3-large" [125] model, which encodes the input tokens into 3072-dimensional embeddings. Moreover, during the initialization phase of the vector stores, the instances of both datasets are inserted in batches of size 1000, in order to reduce the number of OpenAI API calls, as well as track the insertion progress, using the 'tqdm' module [126]. Each instance is assigned a unique 128-bit identifier, when stored in the database, using the 'uuid' module [127]

### 4.3.3 Pipeline Testing

Both pipelines instantiate their respective vector store as the retriever component, with Chroma performing retrieval using the MMR algorithm, and SQLiteVec performing retrieval using Similarity search. Both retrievers fetch the top-10 relevant to the input query context chunks.

Subsequently, taking into account that the number of input question-answer pairs is quite large, and that the main source of latency in the pipeline is the calculation of the input queries' embeddings and their comparison with the embeddings of the vector store, I decided to randomly extract 100 question-answers pairs from both datasets, and test the pipelines on them.

Moving on, I also defined the input prompt to gpt-4o-mini as illustrated below:

```
# Defining the prompt template of the RAG pipeline
prompt_template = """Answer the following question:
\n\n
{question}
\n\n

Using the following list of context passages:
\n\n
{context}
\n\n"""
```

Figure 4.1: The template for the RAG system's input prompt

Before the final selection of the RAG pipelines, the "rlm/rag-prompt" prompt template of the LangChain hub was tested, and after visually inspecting some of the tested examples, my prompt template was proven to be far more beneficial to the system, as a larger portion of the tested queries resulted in more relevant responses.

Next, the RAG pipeline is iteratively tested on the 100 randomly selected dataset instances, and the results are stored in a JSON file, as a checkpoint for skipping to the evaluation process, in the respective Jupyter Notebooks. The format of the results is:

1) For the MS MARCO results:

```
[
    {
        "query_id": <The input query's identifier>,
        "query": <The actual input query>,
        "ground_truth_answer": <The ground-truth answer provided in the dataset>,
        "generated_answer": <RAG pipeline's generated response>,
        "retrieved_chunks": [
            {
                "page_content": <The context passage of the retrieved chunk>,
                "metadata": {
                    "query_id": <The input query's identifier that ensures
                                 the retrieved chunk is related to the input query>,
                    "query_type": <The category of the input query>,
                    "url": <The url source of the retrieved chunk>
                }
            }
        ]
    }
    .......
]
```

Figure 4.2: JSON format of the RAG results of the MS MARCO dataset

2) For the LegalBench-RAG results:

```
[
    {
        "query_id": <The input query's identifier>,
        "query": <The actual input query>,
        "ground_truth_answer": <The ground-truth answer provided in the dataset>,
        "generated_answer": <RAG pipeline's generated response>,
        "retrieved_chunks": [
            {
                "page_content": <The context passage of the retrieved chunk>
            }
        ]
    }
    .......
]
```

Figure 4.3: JSON format of the RAG results of the LegalBench-RAG dataset

## 4.4 Evaluation Results

### 4.4.1 RAGChecker

The RAGChecker framework is set up by executing the following commands:

1) pip install ragchecker

2) python -m spacy download en_core_web_sm (This command downloads the English small language model, en_core_web_sm)

Before executing the evaluation its necessary to transform the RAG pipeline's results into the JSON format that's required by RAGChecker:

```
{
    results: [
        {
            "query_id": <query's identifier as a string value>,
            "query": <The actual input query>,
            "gt_answer": <The ground-truth answer provided in the dataset>,
            "response": <RAG pipeline's generated response>,
            "retrieved_context": [ <The list of the retrieved chunks, which are pertinent to the input  query>
                {
                    "doc_id": <The document identifier of the retrieved context passage as a string value>,
                    "text": <The actual retrieved context passage>
                },
                {
                    "doc_id": <The document identifier of the retrieved context passage as a string value>,
                    "text": <The actual retrieved context passage>
                }
                ......
            ]
        }
        ......
    ]
}
```

Figure 4.4: Input JSON format for the RAG results required by RAGChecker

The evaluation process of RAGChecker employs two LLMs, the extractor and the checker. The former model is assigned to extract the individual claims from an input text passage, while the latter model is assigned to check whether a given input claim belongs in a ground-truth or reference text passage. The choice for both of those models was also gpt-4o-mini, and RAGChecker assigns a batch size of 10 for the extractor and checker models, in both RAG pipeline evaluations.

RAGChecker outputs the results in the scale of 0-100, and are presented as percentages:

Table 4.1: RAGChecker Evaluation Results on Overall Metrics

| **Dataset** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|
| MS MARCO | 64.0% | 73.8% | 61.9% |
| LegalBench-RAG | 66.1% | 64.5% | 60.9% |

LegalBench-RAG's pipeline performed slightly better than MS MARCO's pipeline, in retrieving relevant documents, but it's was notably worse in fetching larger portions of relevant documents, resulting in lower recall. Both pipelines performed similarly, in terms of F1-score.

Table 4.2: RAGChecker Evaluation Results on Retrieval Metrics

| **Dataset** | **Claim Recall** | **Context Precision** |
|---|---|---|
| MS MARCO | 93.5% | 83.9% |
| LegalBench-RAG | 82.6% | 91.6% |

MS MARCO's pipeline is exceptional at including a high portion of ground-truth claims in the retrieved context passages, but struggles slightly with the precision of the retrieved context. On the contrary, LegalBench-RAG's pipeline finds it challenging to retrieve as many ground-truth claims, but its retrieved context passages are more precise and relevant to the input query. The significant variations in the retrieval metrics likely occur due to the usage of different vector stores for each pipeline, as the MS MARCO corpus is stored in Chroma, while the LegalBench-RAG corpus is stored in SQLiteVec. Another factor contributing might be the

different retrieval algorithms used, as Chroma performed MMR and SQLiteVec performed similarity search.

Table 4.3: RAGChecker Evaluation Results on Generation Metrics - (1/2)

| Dataset | Context Utilization | Relevant Noise Sensitivity | Irrelevant Noise Sensitivity |
|---|---|---|---|
| MS MARCO | 77.8% | 28.4% | 2.7% |
| LegalBench-RAG | 75.6% | 26.3% | 0.7% |

Both RAG pipelines fair pretty well, in terms of their ability to use the retrieved context to answer the queries. Also, they have good tolerance to noise in relevant chunks that were actually used to answer the queries, and they're are exceptional at ignoring the noise of irrelevant chunks.

Table 4.4: RAGChecker Evaluation Results on Generation Metrics - (2/2)

| Dataset | Hallucination | Self Knowledge | Faithfulness |
|---|---|---|---|
| MS MARCO | 4.5% | 0.8% | 94.7% |
| LegalBench-RAG | 6.8% | 0.8% | 92.3% |

Additionally, "gpt-4o-mini" displayed low levels of hallucination and even lower levels of employing its own internal knowledge to answer the queries. Last but not least, both pipelines exhibited excellent levels of faithfulness, as the generated answers were created by claims that originate in the retrieved context.

### 4.4.2 RAGAs

For the evaluation of the RAG pipelines with RAGAs, it was necessary to install RAGAs in an independent environment, due to some dependency conflicts of python modules required by RAGChecker and RAGAs, such as transformers and scikit-learn. That's why RAGAs is set up in a separate anaconda environment with the commands:

1) conda create –name rag_eval_ragas python=3.11.5

2) conda install jupyter

3) pip install ragas

RAGAs provides functions that compute all the metrics that are presented in section 3.2, but when I tested them for a few examples of RAG results, their evaluation was a resource-intensive process that almost required 10 minutes for each example. Given that, I decided to evaluate the 100 question-answer examples, only on Context Relevance (or Context Recall as defined in the code), Answer Relevance (or Response Relevancy as defined in the code), and Answer Faithfulness. The results range from 0 to 1, but the below table displays them in percentages:

Table 4.5: RAGAs Evaluation Results

| **Dataset** | **Context Relevance** | **Answer Relevance** | **Faithfulness** |
|---|---|---|---|
| MS MARCO | 91.3% | 66.7% | 83.3% |
| LegalBench-RAG | 87.2% | 77.6% | 56.8% |

The RAGAs evaluation results are quite interesting. Firstly, although both pipelines display great context relevance, the MS MARCO pipeline slightly outperforms the LegalBench-RAG pipeline. Contrary to that, LegalBench-RAG pipeline surpasses the MS MARCO pipeline be a considerable margin, regarding the answer's relevance to the input query. Finally, the MS MARCO pipeline significantly exceeds the LegalBench-RAG pipeline in terms of the generated response's faithfulness to the retrieved context, with the latter having, surprisingly, mediocre efficiency.

## 4.5 Comparison of Evaluation Results

The evaluation results produced by RAGChecker and RAGAs highlight key differences in how those frameworks perform the assessment process. More specifically, both frameworks align on the evaluation of **Context Relevance**, as RAGChecker's claim recall (93.5% and 82.6% - for each pipeline respectively) and context precision (83.9% and 91.6%) are similar

to RAGAs' context relevance (91.3% and 87.2%), with the MS MARCO pipeline achieving higher claim recall, while the LegalBench-RAG is far greater in context precision.

Furthermore, while RAGChecker doesn't provide an exact **Answer Relevance** metric, RAGAs' answer relevance (66.7% and 77.6%) can be partially compared with RAGChecker's context utilization (77.8% and 75.6%). The former metric estimates how well the generated answer addresses the input question, while the latter estimates how effectively the generated answer incorporates the retrieved context. Both metrics exhibit similar performance, except the MS MARCO pipeline slightly struggles to align the generated response to the input question, despite that both pipelines leverage the retrieved context with similar efficiency. Although these metrics assess different aspects of the generated response, they both examine how well the generator model can leverage the input data, either the question or the retrieved context.

Finally, when it comes to the **Faithfulness** of the generated answer to the retrieved context, both frameworks provide the exact metric. RAGChecker's faithfulness measurements (94.7% and 92.3%) significantly exceed the equivalent RAGAs measurements (83.8% and 56.8%), and RAGAs' faithfulness indicates that the LegalBench-RAG pipeline considerably struggles to generated responses that are grounded to the retrieved context. This discrepancy in faithfulness most likely originates from the evaluation methodology of the frameworks, as RAGChecker uses an extractor model to collect a set of individual claims from the responses and context, while RAGAs instructs another LLM to estimate the metrics, with metric-specific prompts.

## 4.6 Limitations

While the experiments provided valuable insights into the performance of the RAG pipelines both on general and legal data, it's equally important to acknowledge the challenges I encountered during the building process of the RAG pipelines, as well as the implementation process of the RAG evaluation frameworks. These limitations can be considered additional areas of experimentation that could further enhance the research's quality.

### 4.6.1 Vector Stores & Retrievers

Although selecting LangChain as the development framework of my RAG pipelines was straightforward, due to its extensive integrations with LLM and vector store providers, choosing a vector store proved to be more challenging. Initially, I employed Chroma [122], both as a vector store and the retriever, for both types of datasets. The retrieval performed better with MMR search compared to similarity search. However, the pipeline significantly underperformed on the LegalBench-RAG dataset for both retrieval algorithms, as displayed by the retrieval results (which are not provided) of a few dataset instances (further discussed in Section 4.2). To resolve this issue, I tested SQLiteVec [123], which used simple similarity search when used as a retriever, and the retrieval results on the same few dataset instances were sufficient enough, so that I decided to create 2 distinct RAG pipelines.

### 4.6.2 Generator & Embedding Models

The selection of gpt-4o-mini as the generator model and text-embedding-3-large as the embedding model of the retrieval corpus, was also straightforward. However, testing the RAG pipelines on the full MS MARCO and LegalBench-RAG datasets would incur significant costs. Taking into account OpenAI's API cost for input ($0.150 / 1M tokens) and output ($0.600 / 1M tokens) tokens, and the fact that the retrieval process returns the 10 most pertinent text chunks to the input query, I decided to test the RAG pipelines on small samples of 100 instances of each dataset.

Moving on, as an attempt to experiment with the datasets at their full scale, I tried substituting gpt-4o-mini with Meltemi [128], the first open LLM for Greek (which is also suitable for English data), but my search for a publicly-available API, or LangChain integration, was unsuccessful.

### 4.6.3 Evaluation Frameworks

The initial goal of this thesis was to evaluate the retrieval and generation capabilities of a RAG pipeline, on both generic and legal data, using the RAGChecker, RAGAs, ARES, and AutoRAG evaluation frameworks. Having already decided to develop 2 RAG pipelines and compromised with a significantly lesser portions of the selected datasets, the evaluation process with RAGChecker didn't pose any serious challenges. On the contrary, I couldn't exploit

the full range of metrics provided by RAGAs and only calculated the 3 most basic metrics (Context Relevance, Answer Relevance, and Faithfulness) due to the extensive computation time required, even for a minor subset of the 100 sampled instances.

### ARES

ARES requires that the following 3 configuration parameters were set before the evaluation process:

- An **in-domain prompts dataset**, which is a .tsv file that contains a set of few-shot examples, each represented as a Query-Document-Answer triple, as well as 'YES' and 'NO' labels for each of the context relevance, answer relevance, and answer faithfulness metrics.

  I arbitrarily decided to create 10 such instances, using gpt-4o-mini and the following prompt template:

```
# Invoking GPT-4o mini to create 10 examples with different combinations of [[Yes]] and [[No]] labels
# for context relevance, answer relevance, and answer faithfulness
prompt_template = """
You are tasked with generating ten combinations of `Query - Document - Answer` triples
from the examples provided. Each combination should result in various combinations of the following labels:

- Context Relevance Label: [[Yes]] or [[No]]
- Answer Relevance Label: [[Yes]] or [[No]]
- Answer Faithfulness Label: [[Yes]] or [[No]]

Definitions:
1. **Context Relevance**: A context passage is relevant if it provides meaningful information about the query.
2. **Answer Relevance**: An answer is relevant if it logically answers the query based on the provided context passage.
3. **Answer Faithfulness**: An answer is faithful if it aligns with the factual information presented in the context passage.

The input examples are provided in a Python list, where each example is a dictionary that contains the query, a list of context passages,
and a list of answers:

{examples}
"""
```

Figure 4.5: Prompt Template for the creation of the few In-domain examples for ARES

- An **unlabeled evaluation set**, which is a .tsv file that consists of the input Query-Document-Answer triples to be evaluated. I create this set by storing the RAG results in a .tsv file.

- The **model** used by ARES, which is gpt-4o-mini.

The problem of ARES appears in the execution of the main evaluation command and it gives the following error message:

Loading widget...
Attempt 1 failed with error: Client.__init__() got an unexpected keyword argument 'proxies'
Attempt 2 failed with error: Client.__init__() got an unexpected keyword argument 'proxies'

**AutoRAG**

The implementation of AutoRAG begins with storing the RAG results in .parquet file. The retrieval and generation capacities of the RAG pipeline are evaluated by 2 decorated functions that execute the retrieval and generation processes independently in order to compute the corresponding metrics.

However, one of the decorator parameters is the ID of the retrieved context passage that actually addresses the question ("retrieval_gt"). To compute this ID, I traversed the dataset subset of 100 samples (which I used to test the RAG pipelines) and found the relevant context passage for each instance, extracted its url, and compared it with the url's of the retrieved passages. If I found the desired retrieved context passage, I assigned "retrieval_gt" the passage's query ID. Otherwise, I assigned "retrieval_gt" the default value of -1.

Also, the retrieval evaluation function requires a list of similarity scores between each query and its corresponding retrieved context passages. Nevertheless, the retrieval process of Chroma in the first RAG pipeline used MMR to search for the relevant context passages. That's why I computed the cosine similarity between the embeddings, produced by text-embedding-3-large, of every query and its corresponding retrieved passages. Before executing the AutoRAG, the aforementioned .parquet file gets augmented with columns of the "retrieval_gt", query embeddings, and retrieved passages embeddings, of every instance in the dataset.

The problem of AutoRAG's implementation becomes evident in the script's output, which generates a matrix containing columns for the required inputs and each evaluation metric. where instead of their values, However, instead of outputting the actual measurement values, it displays **None** for all of metrics across all examples.

## 4.7 Project Directory Structure

This thesis includes the following files:

1. **RAG_Evaluation_MS_Marco.ipynb -** Jupyter Notebook that describes the building

process of the RAG pipeline that uses the MS MARCO dataset, it's testing, and the evaluation process using RAGChecker and RAGAs. Also, it includes the unsuccessful attempts of ARES and AutoRAG.

2. **RAG_Evaluation_LegalBench-RAG.ipynb -** Jupyter Notebook that describes the building process of the RAG pipeline that uses the LegalBench-RAG dataset, it's testing, and the evaluation process using RAGChecker and RAGAs, but skips the unsuccessful trials of ARES and AutoRAG.

3. **./Datasets/ -** Directory that stores the MS MARCO (microsoft___ms_marco folder) and LegalBench-RAG (contractnli, cuad, maud, privacy_qa, and benchmark folders) datasets.

4. **./Vector_Stores/ -** Directory that contains the MS MARCO and LegalBench-RAG vector stores.

5. **./Output_Files/ -** Directory that stores all the produced files from the 2 Jupyter Notebooks, such as the RAG pipeline results, the RAGChecker and ARES evaluation results, as well as the files created before the unsuccessful trials of ARES and AutoRAG.

The Jupyter Notebooks and the directories of datasets, vector stores, and output files can be found at the MEGA Drive: `https://mega.nz/fm/rzhTUB5S`.

# Chapter 5

# Synopsis and Prospects

In this thesis, I explored the foundational concepts of LLMs and RAG systems. More specifically, I introduced the fundamental Transformer architecture that paved the way for the development of numerous LLMs, and its most popular variations, followed by a short review of some representative models from each one. Then, I described the pre-training and fine-tuning processes of LLMs, including the selection and preprocessing of the pre-training corpus, tokenization methods and pre-training objectives. I also discussed the two important fine-tuning paradigms, instruction-tuning, which attempts to teach LLMs to follow task-specific instructions, and alignment-tuning, whose goal is to make LLMs embrace human values and generate helpful and harmless responses.

Moving on, I defined a RAG system and introduced its two key components, the retriever, which is responsible for fetching document chunks that are relevant to an input question, and the generator, which is assigned to leverage the retrieved context as well as its internal knowledge to generate a factual and coherent answer. I thoroughly analyzed the retrieval process of a RAG pipeline, describing the chunking and indexing process of the retrieval corpus, and reviewing the well-known retrievers: BM25, DPR, ColBERT, and REALM. Finally, I categorized RAG applications in 3 paradigms, Naive RAG, Advanced RAG, and Modular RAG, each category extending the range of functionalities offered by the previous one.

In the next section, I presented my research on the available RAG Evaluation Frameworks, thoroughly examining the capabilities of tools such as RAGChecker, RAGAs, ARES and AutoRAG, etc, to assess multiple aspects of the retrieval and generation quality of a RAG pipeline. I built 2 RAG pipelines, using the LangChain library that conveniently provides integrations with numerous LLM and Database providers. Given the emphasis of the thesis in

the legal field, but also the need to perform a detailed evaluation on my pipelines, one RAG pipeline uses the generic MS MARCO dataset as the retrieval corpus which is stored in the Chroma vector store, and the other uses the legal LegalBench-RAG dataset which is stored in the SQLiteVec vector store. Both pipelines utilize the gpt-4o-mini model as the generator. I provided and discussed the evaluation results of RAGChecker and RAGAs and display the values of the evaluation metrics in a set of tables.

## 5.1 Future Work

After completing the experiments in this thesis, it was evident that evaluating the RAG pipelines only on 2 evaluation frameworks is insufficient, as further trials with other evaluation tools are required in order to acquire more confident estimations of the retrieval and generation performance of the RAG pipelines.

I can summarize potential future directions as follows. First of all, solving the errors of the ARES and AutoRAG implementations would fulfill the initial goal of this thesis. Second, evaluating the RAG results using the full extent of RAGAs metrics - including traditional LLM metrics, such as BLEU and ROUGE - would allow for a more thorough assessment of the RAG pipelines and might provide useful insights in terms of the system's capacity to handle generic and legal data.

In addition, the same pipelines can be tested on other generic (Natural Questions [64], HotpotQA [65], etc) and legal (CaseHOLD [129], LexGLUE [130], etc) datasets. This way we can examine the system's ability to generalize across diverse open-domain contexts and various dataset features, as well as adapt to different types of legal jargon across a wide range of legal documents. Moreover, we can significantly expand the scope of this research if we consider utilizing alternative vector stores (FAISS, Milvus, Pinecore, Redis, all of which have LangChain integrations [131]) for the retrieval, and different generation models (Anthropic, Cohere, Bedrock, Mistral, all of which have LangChain integrations [132]). Combining the plethora of tools in various configurations might result in novel RAG architectures that could potentially achieve greater retrieval and generation performance.

Finally, despite the fact that LangChain offers high-level API integrations to numerous vector store and LLM providers, it still lacks the functionalities that allow for parallel execution of LLM-based applications. An important future direction of this thesis could be to

explore frameworks such as Ray [133], which allows for parallel and distributed execution of AI/ML applications and large-scale model inference, and Dask [134], which specializes in large-scale data preprocessing. Ray could be be used to parallelize the RAG pipeline execution, allowing multiple queries of the input datasets to be simultaneously processed by the system, and Dask could parallelize the non-LLM procedures, such as document chunking, and indexing of the document embeddings in the vector store. Integrating those frameworks could enhance the scalability and efficiency of the RAG pipelines, enabling testing of even larger datasets.

# Bibliography


[1] Shane Greenstein. Preserving the rule of law in the era of artificial intelligence (AI). *Artificial Intelligence and Law*, 30(3):291–323, 2022. Published online: 17 July 2021, Issue date: September 2022.

[2] Navneet Shankar Pandey, Poonam Rawat, Samta Kathuria, Rajesh Singh, Gunjan Chhabra, and Gargi Pant. Artificial Intelligence Assistance in the Domain of Law. In *2023 IEEE International Conference on Contemporary Computing and Communications (InC4)*, volume 1, pages 1–4, 2023.

[3] Robert Walters and Marko Novak. *Artificial Intelligence and Law*, pages 39–69. Springer Singapore, Singapore, 2021.

[4] McKinsey & Company. What is Generative AI? `https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-generative-ai`, 2023.

[5] Clio. AI Tools for Lawyers: How Artificial Intelligence Can Benefit Legal Practices. `https://www.clio.com/resources/ai-for-lawyers/ai-tools-for-lawyers/`, 2024.

[6] Checkbox Team. Legal AI Tools: A Guide for Lawyers and Legal Professionals. `https://www.checkbox.ai/blog/legal-ai-tools-a-guide-for-lawyers-and-legal-professionals`, 2024.

[7] Jerry Levine. 7 Benefits of AI in the Legal Industry. `https://contractpodai.com/news/ai-benefits-legal/`, 2024.

[8] Valerie McConnell. What is CoCounsel? An introduction to CoCounsel, the world's first AI legal assistant. https://help.casetext.com/en/articles/7040012-what-is-cocounsel, 2023.

[9] Arvin Faustino. Harvey AI: Everything You Need to Know. https://capforge.com/harvey-ai-everything-you-need-to-know/, June 2 2023.

[10] Hunter Cyran. New Rules for a New Era: Regulating Artificial Intelligence in the Legal Field. *Case Western Reserve Journal of Law, Technology & the Internet*, 15(1):1, 2024. Available at: https://scholarlycommons.law.case.edu/jolti/vol15/iss1/1.

[11] Joseph Anderson. AI and the Legal Puzzle: Filling Gaps, But Missing Pieces. *Mercer Law Review*, 75(5), 2024. Available at: https://digitalcommons.law.mercer.edu/jour_mlr/vol75/iss5/8.

[12] Natassha Selvaraj. What is Retrieval Augmented Generation (RAG)? https://www.datacamp.com/blog/what-is-retrieval-augmented-generation-rag, January 30 2024.

[13] Lexemo LLC. Retrieval-Augmented Generation (RAG) in Legal Research. https://www.linkedin.com/pulse/retrieval-augmented-generation-rag-legal-research-lexemo-llc-r4uie/, June 5 2024.

[14] James Evans and Brennan Whitfield. What Is Retrieval Augmented Generation (RAG)? https://builtin.com/articles/retrieval-augmented-generation-explained, August 16 2024.

[15] Summer Thompson. 3 Practical Applications of Retrieval-Augmented Generation for Legal Teams. https://onna.com/blog/practical-applications-of-retrieval-augmented-generation-for-legal-teams, 2024. Product Marketing @ Onna.

[16] MyScale. 3 Ways RAG Technology Transforms Legal Research and Analysis. https://myscale.com/blog/rag-technology-legal-research-analysis-transformations/, March 15 2024.

[17] IBM. Large Language Models (LLMs). `https://www.ibm.com/topics/large-language-models`.

[18] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A Comprehensive Overview of Large Language Models, 2024.

[19] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the Opportunities and Risks of Foundation Models, 2022.

[20] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang,

Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A Survey of Large Language Models, 2024.

[21] Joseph Weizenbaum. ELIZA—A Computer Program for the Study of Natural Language Communication Between Man and Machine. *Communications of the ACM*, 9(1):36–45, 1966.

[22] Yorick Wilks. Natural Language Understanding Systems Within the AH Paradigm: A Survey and Some Comparisons. Memo AIM-237 STAN-CS-74-436, Stanford Artificial Intelligence Laboratory, Computer Science Department, December 1974.

[23] P Majumder, M Mitra, and BB Chaudhuri. N-gram: a language independent approach to IR and NLP. In *International conference on universal knowledge and language*, volume 2, 2002.

[24] Talal Almutiri and Farrukh Nadeem. Markov models applications in natural language processing: a survey. *Int. J. Inf. Technol. Comput. Sci*, 2:1–16, 2022.

[25] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative Study of CNN and RNN for Natural Language Processing, 2017.

[26] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A Neural Probabilistic Language Model. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[27] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12:2493–2537, 2011.

[28] I Sutskever. Sequence to Sequence Learning with Neural Networks. *arXiv preprint arXiv:1409.3215*, 2014.

[29] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate, 2016.

[30] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional Sequence to Sequence Learning, 2017.

[31] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[32] Josep Ferrer. How Transformers Work: A Detailed Exploration of Transformer Architecture. `https://www.datacamp.com/tutorial/how-transformers-work`, Jan 9 2024.

[33] Michael Phi. Illustrated Guide to Transformers - Step by Step Explanation. `https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0`, May 1 2020.

[34] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization, 2016.

[35] 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. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[36] M Lewis. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[37] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota, 2019.

[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019.

[39] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, 2020.

[40] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, 2020.

[41] Alec Radford. Improving language understanding by generative pre-training. *OpenAI Technical Report*, 2018.

[42] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[43] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[44] OpenAI, Josh Achiam, Steven Adler et, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse,

Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Win-

ter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. GPT-4 Technical Report, 2024.

[45] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models, 2023.

[46] Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[47] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[48] Common Crawl Team. Commoncrawl. `https://commoncrawl.org/`.

[49] Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. The pushshift reddit dataset. In *Proceedings of the international AAAI conference on web and social media*, volume 14, pages 830–839, 2020.

[50] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling, 2020.

[51] Danny Hernandez, Tom Brown, Tom Conerly, Nova DasSarma, Dawn Drain, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Tom Henighan, Tristan Hume, Scott Johnston, Ben Mann, Chris Olah, Catherine Olsson, Dario Amodei, Nicholas Joseph, Jared Kaplan, and Sam McCandlish. Scaling Laws and Interpretability of Learning from Repeated Data, 2022.

[52] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting Training Data from Large Language Models, 2021.

[53] Rico Sennrich. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[54] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016.

[55] Taku Kudo. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates, 2018.

[56] Tianyi Tang, Junyi Li, Wayne Xin Zhao, and Ji-Rong Wen. MVP: Multi-task Supervised Pre-training for Natural Language Generation, 2023.

[57] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Tali Bers, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. Multitask Prompted Training Enables Zero-Shot Task Generalization, 2022.

[58] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning Language Models with Self-Generated Instructions, 2023.

[59] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. A General Language Assistant as a Laboratory for Alignment, 2021.

[60] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022.

[61] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017.

[62] 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, 2021.

[63] 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, 2024.

[64] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 08 2019.

[65] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering, 2018.

[66] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset, 2018.

[67] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding, 2021.

[68] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models, 2016.

[69] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering, 2024.

[70] Shangyu Wu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Huang, Xue Liu, Tei-Wei Kuo, Nan Guan, and Chun Jason Xue. Retrieval-Augmented Generation for Natural Language Processing: A Survey, 2024.

[71] Michael Borck. Optimising Language Models with Advanced Text Chunking Strategies. *arXiv preprint arXiv:...*, 2024.

[72] S. Joshua Johnson, M. Ramakrishna Murty, and I. Navakanth. A detailed review on word embedding techniques with emphasis on word2vec. *Multimedia Tools and Applications*, 83(13):37979–38007, 2024.

[73] OpenAI. Embedding Models - OpenAI Documentation.

[74] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2020.

[75] Shailja Gupta, Rajesh Ranjan, and Surya Narayan Singh. A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions, 2024.

[76] MyScale Blog. Best Match 25 (BM25) Ranking Algorithm Explained, 2024.

[77] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense Passage Retrieval for Open-Domain Question Answering, 2020.

[78] Omar Khattab and Matei Zaharia. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT, 2020.

[79] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-Augmented Language Model Pre-Training, 2020.

[80] Yunfan Gao, Yun Xiong, Meng Wang, and Haofen Wang. Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks, 2024.

[81] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query Rewriting for Retrieval-Augmented Large Language Models, 2023.

[82] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models, 2023.

[83] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le, and Denny Zhou. Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models, 2024.

[84] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 02 2024.

[85] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. Learning Maximal Marginal Relevance Model via Directly Optimizing Diversity Evaluation Measures. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 113–122, New York, NY, USA, 2015. Association for Computing Machinery.

[86] Xintao Wang, Qianwen Yang, Yongting Qiu, Jiaqing Liang, Qianyu He, Zhouhong Gu, Yanghua Xiao, and Wei Wang. KnowledGPT: Enhancing Large Language Models with Retrieval and Storage Access on Knowledge Bases, 2023.

[87] Xin Cheng, Di Luo, Xiuying Chen, Lemao Liu, Dongyan Zhao, and Rui Yan. Lift Yourself Up: Retrieval-augmented Text Generation with Self-Memory. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 43780–43799. Curran Associates, Inc., 2023.

[88] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than Retrieve: Large Language Models are Strong Context Generators, 2023.

[89] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP, 2023.

[90] Dongyu Ru, Lin Qiu, Xiangkun Hu, Tianhang Zhang, Peng Shi, Shuaichen Chang, Cheng Jiayang, Cunxiang Wang, Shichao Sun, Huanyu Li, Zizhao Zhang, Binjie Wang, Jiarong Jiang, Tong He, Zhiguo Wang, Pengfei Liu, Yue Zhang, and

Zheng Zhang. RAGChecker: A Fine-grained Framework for Diagnosing Retrieval-Augmented Generation, 2024.

[91] Amazon Science. RAGChecker: Code for Evaluation Framework for Retrieval-Augmented Generation Systems. https://github.com/amazon-science/RAGChecker/tree/main, 2024.

[92] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. RAGAS: Automated Evaluation of Retrieval Augmented Generation, 2023.

[93] Exploding Gradients. RAGAs: Retrieval-Augmented Generation Assessment Suite. https://github.com/explodinggradients/ragas, 2023.

[94] Exploding Gradients. RAGAs Documentation: Available Metrics. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/, 2023.

[95] Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems, 2024.

[96] Stanford Future Data Systems Group. ARES: Automated Retrieval-augmented Generation Evaluation Suite. https://github.com/stanford-futuredata/ARES/tree/main, 2024.

[97] Dongkyu Kim, Byoungwook Kim, Donggeon Han, and Matouš Eibich. AutoRAG: Automated Framework for optimization of Retrieval Augmented Generation Pipeline, 2024.

[98] Marker Inc. Korea. AutoRAG: Automated Retrieval-Augmented Generation Framework. https://github.com/Marker-Inc-Korea/AutoRAG, 2024.

[99] Marker Inc. Korea. AutoRAG Documentation. https://docs.auto-rag.com/, 2024.

[100] Evidently AI. Ranking Metrics. https://www.evidentlyai.com/ranking-metrics, 2024.

[101] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

[102] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating Text Generation with BERT, 2020.

[103] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment, 2023.

[104] David Rau, Hervé Déjean, Nadezhda Chirkova, Thibault Formal, Shuai Wang, Vassilina Nikoulina, and Stéphane Clinchant. BERGEN: A Benchmarking Library for Retrieval-Augmented Generation, 2024.

[105] NAVER LABS. BERGEN: BEnchmarking Retrieval-augmented GENeration. `https://github.com/naver/bergen`, 2024.

[106] Tonic AI. Tonic Validate. `https://docs.tonic.ai/validate/about-rag-metrics/tonic-validate-rag-metrics-summary`, 2024. GitHub repository and documentation `https://github.com/TonicAI/tonic_validate`.

[107] Fate UBW. RAGLAB, 2024. GitHub repository.

[108] Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, Xinfeng Li, Zhengran Zeng, Zhen Wu, Wei Ye, Wenyuan Xu, Yue Zhang, Xinyu Dai, Shikun Zhang, and Qingsong Wen. RAGLAB: A Modular and Research-Oriented Unified Framework for Retrieval-Augmented Generation, 2024.

[109] Truera. TruLens. `https://www.trulens.org/getting_started/core_concepts/rag_triad/`, 2024. GitHub repository and documentation `https://github.com/truera/trulens`.

[110] Raga AI Hub. RagaAI. `https://docs.raga.ai/ragaai-catalyst/ragaai-metric-library/rag-metrics`, 2024. GitHub repository and documentation `https://github.com/raga-ai-hub/raga-llm-hub`.

[111] Arize AI. Phoenix. https://docs.arize.com/phoenix/tracing/use-cases-tracing/rag-evaluation, 2024. GitHub repository and documentation https://github.com/Arize-ai/phoenix.

[112] Giskard AI. Giskard/RAGET. https://docs.giskard.ai/en/stable/open_source/testset_generation/rag_evaluation/index.html, 2024. GitHub repository and documentation https://github.com/Giskard-AI/giskard.

[113] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FactScore: Fine-grained Atomic Evaluation of Factual Precision in Long Form Text Generation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12076–12100, Singapore, December 2023. Association for Computational Linguistics.

[114] Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling Large Language Models to Generate Text with Citations. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6465–6488, Singapore, December 2023. Association for Computational Linguistics.

[115] TruLens. Honest, Harmless, and Helpful Evaluations. https://www.trulens.org/getting_started/core_concepts/honest_harmless_helpful_evals/.

[116] LegalBench-RAG: A Benchmark for R]etrieval-Augmented Generation in the Legal Domain, author=Nicholas Pipitone and Ghita Houir Alami, year=2024, eprint=2408.10343, archiveprefix=arXiv, primaryclass=cs.AI, url=https://arxiv.org/abs/2408.10343.

[117] Abhilasha Ravichander, Alan W Black, Shomir Wilson, Thomas Norton, and Norman Sadeh. Question Answering for Privacy Policies: Combining Computational and Legal Perspectives, 2019.

[118] Dan Hendrycks, Collin Burns, Anya Chen, and Spencer Ball. CUAD: An Expert-Annotated NLP Dataset for Legal Contract Review, 2021.

[119] Steven H. Wang, Antoine Scardigli, Leonard Tang, Wei Chen, Dimitry Levkin, Anya Chen, Spencer Ball, Thomas Woodside, Oliver Zhang, and Dan Hendrycks. MAUD: An Expert-Annotated Legal NLP Dataset for Merger Agreement Understanding, 2023.

[120] Yuta Koreeda and Christopher D. Manning. ContractNLI: A Dataset for Document-level Natural Language Inference for Contracts, 2021.

[121] OpenAI. GPT-4o mini: advancing cost-efficient intelligence, 18 July 2024.

[122] LangChain. Chroma Integration in LangChain, `https://python.langchain.com/docs/integrations/vectorstores/chroma/`.

[123] LangChain. SQLiteVec Integration in LangChain, `https://python.langchain.com/docs/integrations/vectorstores/sqlitevec/`.

[124] LangChain Team. LangChain Documentation, `https://python.langchain.com/docs/introduction/`.

[125] OpenAI. New embedding models and API updates,. `https://openai.com/index/new-embedding-models-and-api-updates/`, 25 January 2024.

[126] tqdm Developers. tqdm: A Fast, Extensible Progress Bar for Python and CLI,. `https://tqdm.github.io/`.

[127] UUID Generator. UUID Generator - Developer Corner: Python. `https://www.uuidgenerator.net/dev-corner/python`.

[128] Leon Voukoutis, Dimitris Roussis, Georgios Paraskevopoulos, Sokratis Sofianopoulos, Prokopis Prokopidis, Vassilis Papavasileiou, Athanasios Katsamanis, Stelios Piperidis, and Vassilis Katsouros. Meltemi: The first open Large Language Model for Greek, 2024.

[129] Lucia Zheng, Neel Guha, Brandon R. Anderson, Peter Henderson, and Daniel E. Ho. When Does Pretraining Help? Assessing Self-Supervised Learning for Law and the CaseHOLD Dataset. `https://arxiv.org/abs/2104.08671`, 2021.

[130] Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Martin Katz, and Nikolaos Aletras. LexGLUE: A Benchmark Dataset for Legal Language Understanding in English, 2022.

[131] LangChain. Vectorstores - LangChain Integrations. https://python.langchain.com/docs/integrations/vectorstores/.

[132] LangChain. Chat Models - LangChain Integrations. https://python.langchain.com/docs/integrations/chat/.

[133] Ray Team. Ray: Scalable Distributed Computing for AI. https://www.ray.io/.

[134] Dask Development Team. Dask: Scalable Analytics in Python. https://www.dask.org/.