



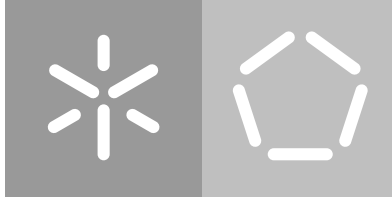
Universidade do Minho

Escola de Engenharia

José Pedro Martins Magalhães

**Exploration and refinement of language models
for applications in the food industry**

October, 2024



Universidade do Minho

Escola de Engenharia

José Pedro Martins Magalhães

Exploration and refinement of language models for applications in the food industry

Master's Dissertation

Master's in Informatics Engineering

Work supervised by

Miguel Rocha

Ricardo Pereira

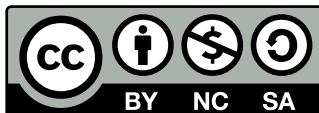
COPYRIGHT AND TERMS OF USE OF THIS WORK BY A THIRD PARTY

This is academic work that can be used by third parties as long as internationally accepted rules and good practices regarding copyright and related rights are respected.

Accordingly, this work may be used under the license provided below.

If the user needs permission to make use of the work under conditions not provided for in the indicated licensing, they should contact the author through the RepositoriUM of Universidade do Minho.

License granted to the users of this work



**Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International
CC BY-NC-SA 4.0**

<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.en>

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

_____, _____
(Location) (Date)

(José Pedro Martins Magalhães)

Acknowledgements

I would like to express my deep gratitude to everyone who has contributed significantly to the completion of this master's thesis.

First of all, I would like to thank my supervisor, Professor Miguel Rocha, for his innovative ideas, guidance, and unwavering support. His insights and encouragement were crucial to the realization of this project. To my co-supervisor, Professor Ricardo Pereira, I am sincerely grateful for his tireless guidance, valuable feedback, and constant support throughout this journey. His dedication and shared wisdom were instrumental in the development of this work.

I would also like to extend my thanks to Diogo Macedo for his tireless help, constant weekly assistance, and his invaluable input, which helped shape this thesis. My gratitude also goes to the team at Omnium AI for their continuous support, encouragement, and all their contributions, which were essential to the success of this project.

I am immensely thankful to my friends, the "mitralhões" for their companionship, unwavering support, and for making these five years of university unforgettable. Their friendship and the countless memories we've shared have made this journey lighter and much more enjoyable.

Lastly, to my parents and family, I offer my heartfelt thanks for their endless love, understanding, and support, which has been a constant source of motivation throughout my academic journey.

To everyone mentioned, my sincere and deepest thanks.

Abstract

This thesis explores the utilization of [Large Language Models \(LLMs\)](#) in the food industry to enhance decision-making processes and improve product safety and quality. Leveraging advancements in [Natural Language Processing \(NLP\)](#), this work aims to develop a [Retrieval Augmented Generation \(RAG\)](#) system. This system interacts intelligently with domain experts by retrieving and generating responses based on curated corpora of scientific literature and regulatory documents.

Data sources include PubMed, [Code of Federal Regulations \(CFR\)](#) Title 21 regulations, and other authoritative materials, which were processed and stored in a vector index to enable efficient and contextually relevant retrieval. To assess the performance of the system, a custom dataset, "LLM Eval", was created to simulate real-world food engineering queries, allowing for both human and automatic evaluation of model responses.

This work also explores the use of rerankers and corrective mechanisms to improve answer relevance and factual accuracy. The result is an intelligent conversational agent capable of supporting industry professionals with informed, fact-based recommendations, aiding in compliance, safety improvements, and sustainable food production practices.

Keywords: RAG, AI, LLM, NLP, Transformers

Resumo

Esta tese explora a utilização de [Large Language Models \(LLMs\)](#) na indústria alimentar para melhorar os processos de tomada de decisão e aumentar a segurança e a qualidade dos produtos. Aproveitando os avanços em [Natural Language Processing \(NLP\)](#), este trabalho tem como objetivo desenvolver um sistema de [Retrieval Augmented Generation \(RAG\)](#). Este sistema interage de forma inteligente com especialistas da área, recuperando e gerando respostas com base em corpus selecionados de literatura científica e documentos regulamentares.

As fontes de dados incluem PubMed, as regulamentações do [Code of Federal Regulations \(CFR\)](#) Título 21 e outros materiais relevantes, que foram processados e armazenados num índice vetorial para permitir uma recuperação eficiente e contextualmente relevante. Para avaliar o desempenho do sistema, foi criado um conjunto de dados personalizado, "LLM Eval", para simular consultas do mundo real na engenharia alimentar, permitindo tanto a avaliação humana como automática das respostas do modelo.

Este trabalho explora também a utilização de rerankers e mecanismos de correção para melhorar a relevância das respostas e a precisão dos factos recolhidos. O resultado é um agente conversacional inteligente capaz de apoiar os profissionais da indústria com recomendações informadas e baseadas em factos, auxiliando na conformidade, melhorias de segurança e práticas de produção alimentar sustentável.

Palavras-chave: AI, LLM, RAG, NLP, Transformers

Contents

List of Figures	ix
List of Tables	xi
Acronyms	xii
1 Introduction	1
1.1 Context and motivation	1
1.2 Objectives and expected results	2
1.3 Thesis organization	2
2 State of the Art	4
2.1 Food engineering	4
2.1.1 Nutritional value	4
2.1.2 Food reformulation	5
2.1.3 Food safety and regulations	5
2.1.4 Generally recognized as safe	5
2.2 Machine learning	5
2.3 Deep learning	6
2.3.1 Artificial neuron	7
2.3.2 Deep neural networks	9
2.3.3 Training a neural network	10
2.3.4 Overfitting and underfitting	14
2.4 Word embeddings	16
2.5 The curse of dimensionality	16
2.6 Specialized neural architectures	17
2.6.1 Convolutional neural network	18
2.6.2 Recurrent neural network	19
2.6.3 Long short-term memory	20
2.6.4 Transformer	21
2.7 Large language models	25

2.7.1	Pre-training	27
2.7.2	Fine-tuning	28
2.7.3	Reinforcement learning from human feedback	30
2.7.4	Mixture of experts	31
2.7.5	Prompt engineering	32
2.7.6	Metrics	34
2.7.7	Benchmarks	35
2.7.8	Rating	36
2.8	Retrieval augmented generation	36
2.8.1	Retrieval mechanism	37
2.8.2	RAG evaluation	41
2.8.3	RAG in food engineering	42
3	Development	44
3.1	Technologies	44
3.2	GumGum package	45
3.2.1	Package Structure	45
4	Experimental validation	53
4.1	Benchmarking datasets	53
4.1.1	LLM Eval	54
4.1.2	PubMedQA	55
4.1.3	CFR	57
4.2	Data preprocessing	58
4.2.1	LLM Eval	58
4.2.2	PubMedQA	58
4.2.3	CFR	58
4.3	Cost Estimation	59
4.4	Evaluation Method	61
4.4.1	LLM Evaluation	62
4.4.2	Embedding Model Evaluation	63
4.4.3	Filters Evaluation	65
5	Results	68
5.1	LLM Evaluation	68
5.2	Embedding Model Evaluation	70
5.3	Filters Evaluation	71
5.4	Demo App	72

6 Conclusions and Future Work	76
Bibliography	78

List of Figures

2.1	Relationship between AI, ML and DL.	7
2.2	The artificial neuron [16].	7
2.3	Examples of classic activation functions [17].	8
2.4	Examples of rectifier-based activation functions [17].	8
2.5	Examples of parameterized standard activation functions [17].	9
2.6	Example architecture of a Deep Neural Network (DNN).	9
2.7	Illustration of the phenomena of Overfitting and Underfitting [27].	15
2.8	Syntactic relationships between word embeddings.	16
2.9	Example of the curse of dimensionality. Among 1,000 (x, y) points in which both x and y are normally distributed with a mean of 0 and s.d. $\sigma = 1$, only 6% fall within σ of (x, y) = (1.5, 1.5) (blue circle). However, when the data are projected into a lower dimension - shown by histograms - about 30% of the points (all bins within blue solid lines) are within σ of 1.5. Blue bins in histograms correspond to the blue points [35].	17
2.10	Convolutional Neural Network (CNN) Architecture.	19
2.11	Recurrent Neural Network (RNN) unrolled in time.	20
2.12	Long Short-Term Memory (LSTM) unrolled in time.	21
2.13	Transformer Encoder (left) Decoder (right) Architecture [47].	22
2.14	Scaled Dot-Product Attention [47].	23
2.15	Multi-Head Attention [47].	24
2.16	LLMs parameter count from 2018 to 2022 [57].	27
2.17	RLHF Training Cycle [65].	30
2.18	MoE Layer. In this example the sparse gating network selects two experts to feed the input. Their outputs are modulated by the outputs of the gating network [69].	32
2.19	CoT reasoning process [72].	33
2.20	ToT reasoning process [73].	34
2.21	RAG System Architecture [83].	37
2.22	Retrieval Encoders Training Architecture.	38
2.23	Example of a search on an HNSW graph [84].	40
3.1	GumGum package module structure	45

3.2	RAG Agent behaviour	51
3.3	CRAG Agent behaviour	52
4.1	Histogram of question lengths (Number of words)	55
4.2	Distribution of question categories	55
4.3	Histogram of question lengths (Number of words)	56
4.4	Histogram of context lengths (Number of words)	57
4.5	Histogram of section lengths (Number of words)	58
4.6	Staged Evaluation process	61
4.7	Argilla human feedback platform	62
5.1	Example of answer generated by Gemma 7B Instruct	69
5.2	Comparison of embedding models results	71
5.3	Comparison of filtering combinations results (values represent distance to the origin)	72
5.4	RAG demo answer example. CFR contexts database with GPT-3.5 Turbo and no reranker.	73
5.5	RAG demo deployment diagram	75

List of Tables

2.1	Types of score function f . k is a key. v , b , W , W_1 , W_2 are learnable parameters. d_k is the dimension of the input vector. The act is a nonlinear activation function, such as tanh and ReLU.	23
2.2	Domain-specific LLMs and RAG systems	43
4.1	Analysis of LLM Eval feedback dataset	54
4.2	Analysis of PubMedQA evaluation dataset	56
4.3	Analysis of CFR knowledge dataset	57
4.4	Price comparison for different combinations of embedding models, GPT versions, and dataset sizes.	61
4.5	OpenLM Chatbot Arena Leaderboard [106]	63
4.6	Candidate embedding models	63
4.7	Candidate reranker models	66
5.1	LLM Elo score on LLM Eval dataset	69
5.2	Embedding models evaluation results	70
5.3	Filtering mechanisms evaluation results	72

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
BLEU	Bilingual Evaluation Understudy
CFR	Code of Federal Regulations
CNN	Convolutional Neural Network
CoT	Chain of Thought
CRAG	Corrective Retrieval Augmented Generation
DL	Deep Learning
DNN	Deep Neural Network
FDA	Food and Drugs Administration
FNN	Feedforward Neural Network
GLUE	General Language Understanding Evaluation
GPT	Generative Pretrained Transformer
GPU	Graphics Processing Unit
GRAS	Generally Recognized as Safe
HACCP	Hazard Analysis and Critical Control Points
HNSW	Hierarchical Navigable Small Worlds
HTTPS	Hyper Text Transfer Protocol Secure
IDE	Integrated Development Environment
IVF	Inverted File

JSON	JavaScript Object Notation
LLM	Large Language Model
LSH	Locality Sensitive Hashing
LSTM	Long Short-Term Memory
METEOR	Metric for Evaluation of Translation with Explicit Ordering
ML	Machine Learning
MLM	Masked Language Modeling
MLP	Multilayer Perceptron
MoE	Mixture of Experts
NAG	Nesterov Accelerated Gradient
NLP	Natural Language Processing
NSP	Next Sentence Prediction
PPO	Proximal Policy Optimization
RAG	Retrieval Augmented Generation
RAM	Random Access Memory
REST	Representational State Transfer
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RoPE	Rotary Positional Encoding
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SGD	Stochastic Gradient Descent
SLM	Small Language Model
SQUAD	Stanford Question Answering Dataset
ToT	Tree of Thoughts

TPU Tensor Processing Unit

USDA United States Department of Agriculture

WHO World Health Organization

Introduction

1.1 Context and motivation

The food industry stands as a keystone in societal well-being, shaping health, economics, and environmental sustainability. Its optimization in quality, safety, and efficiency is pivotal, and the integration of technology emerges as a crucial catalyst in this pursuit. Within this landscape, the emergence of conversational tools that interface seamlessly with industry systems holds immense promise. These tools, capable of delivering pertinent and contextual information [1], signify a transformative step towards streamlining processes within the food sector.

The advancements witnessed in language models [2, 3], notably in the realm of [Natural Language Processing \(NLP\)](#), present a compelling opportunity to revolutionize communication and operational efficiency across diverse domains within the food industry. Leveraging these models entails their training or fine-tuning [4] to comprehend and generate text [3], drawing insights from extensive datasets. This ability equips these models to swiftly and accurately respond to inquiries with intelligent and contextually relevant information [1].

This project aims to explore and refine advanced language models to meet the specific needs of the food industry, enhancing communication and access to information, which will result in significant benefits for the industry.

This thesis was developed within OmniumAI, aimed at contributing to the development of a new generation of [Artificial Intelligence \(AI\)](#)-based systems, most notably the CibusAI platform, to improve food products through the use of [Machine Learning \(ML\)](#), being able to interact with users and explain the decisions being proposed when it comes to the replacement of functional ingredients.

1.2 Objectives and expected results

The main aim of this work will be the exploration of [Large Language Models \(LLMs\)](#), more specifically the development of a [Retrieval Augmented Generation \(RAG\)](#) system using relevant textual sources, to improve their proficiency in human interaction with experts from the food industry/ technology fields. The aim is to create a system with the potential to enhance decision-making in the food industry, based on facts, as a means to improve the safety and quality of the food products. This will be achieved addressing the following scientific/technological goals:

- Systematic review of existing literature on the use of [LLMs](#) specially those focused on the food industry and related biomedical fields;
- Collection and preprocessing of text corpora from relevant sources such as papers (e.g. PubMed) or patents, and systems for retrieving information from selected databases (e.g. PubChem, FoodDB);
- Exploration and implementation of the corpora in a vector database system;
- Exploration of selected [LLMs](#) and embedding models to create a [RAG](#) agent capable of answering user questions based on passages collected from a database;
- Analysis and qualitative and quantitative evaluation of the obtained results, by defining appropriate metrics;
- Creation of an effective conversational agent capable of presenting the obtained results.

After completing the aforementioned stages, the creation of an efficient conversational agent is anticipated, one capable of articulating and substantiating the obtained results based on previously indexed scientific articles. The expectation is for an intelligent system with the potential to enhance decision-making within the food industry, relying on facts to improve the safety and quality of produced goods in a sustainable manner.

1.3 Thesis organization

This document is structured into different chapters to facilitate the understanding and flow of the concepts discussed.

Chapter 1 introduces the context of this work, providing a description of the problem domain and the motivation behind the study. The chapter also outlines the objectives of the research and the expected results.

In Chapter 2, the relevant concepts for the state of the art in this dissertation are presented. These include topics related to food engineering and [Artificial Intelligence \(AI\)](#), with a focus on [Large Language Models \(LLMs\)](#), [Retrieval Augmented Generation \(RAG\)](#) systems, and vector indexing technologies.

Chapter 3 covers the technologies employed in this work, providing an in-depth explanation of the development process for the Python package "GumGum".

Chapter 4 presents the experimental validation, including the analysis and preprocessing of benchmarking datasets. It also discusses a cost study, evaluating and making decisions about the expenses associated with using [LLMs](#). Finally, it outlines the experimental scenarios designed to assess the best configuration for the [RAG](#) system.

In Chapter 5, the results of these experimental scenarios are analyzed, leading to conclusions about the optimal configuration for the system within the scope of this thesis.

Lastly, Chapter 6 offers final conclusions on the work carried out and proposes future directions and improvements that could enhance the long-term results of this study.

State of the Art

2.1 Food engineering

Food engineering is a multidisciplinary field that combines various scientific principles with engineering methodologies to revolutionize the way we produce, process, and distribute food. It is the backbone of our food supply chain, encompassing a spectrum of practices aimed at optimizing food production, ensuring safety, and preserving nutritional quality.

At its core, food engineering intertwines the principles of chemical engineering, microbiology, and biochemistry. It is responsible for designing processes that preserve the nutritional content of foods while maintaining safety standards and extending shelf life.

2.1.1 Nutritional value

In a world inundated with an abundance of food choices, the critical factor often overlooked is the nutritional value of what we consume. Global health organizations underscore the pivotal role diet plays in shaping our overall health. The [World Health Organization \(WHO\)](#) emphasizes the alarming impact of poor dietary habits on the prevalence of non-communicable diseases, such as cardiovascular ailments, diabetes, and certain cancers [5].

This reality becomes more evident when considering the prevalence of malnutrition worldwide. It is not merely a question of inadequate food, it is about the quality of the food consumed. Processed foods, often laden with excessive sugar, sodium, and unhealthy fats, dominate many diets, contributing to widespread malnutrition [5].

2.1.2 Food reformulation

Amid this scenario, food reformulation emerges as a potential solution. It is the process of recalibrating existing food recipes to enhance their nutritional profiles without compromising taste, texture, or safety. This approach aligns with evolving dietary guidelines and aims to create healthier alternatives for consumers [6].

Reformulation entails a balance of reducing detrimental components, like sugars and unhealthy fats, while fortifying foods with essential nutrients, such as fibers, vitamins, and minerals. Through the application of food engineering principles, manufacturers can innovate and transform staple foods into healthier options that resonate with consumer preferences [6, 7].

2.1.3 Food safety and regulations

Ensuring the safety of food products is a critical pillar of food engineering. Food safety refers to the measures taken to prevent foodborne illnesses, contamination, and hazards throughout the food production process, from raw material procurement to consumer consumption. Food engineers play a vital role in developing systems that maintain food safety and ensure compliance with regulatory frameworks. Key regulatory bodies such as the U.S. [Food and Drugs Administration \(FDA\)](#) and the [United States Department of Agriculture \(USDA\)](#) establish stringent standards to protect public health. One regulatory framework in this context is the [Hazard Analysis and Critical Control Points \(HACCP\)](#) system, which mandates manufacturers to identify, evaluate, and control potential food safety hazards at critical stages of production [8].

2.1.4 Generally recognized as safe

The [Generally Recognized as Safe \(GRAS\)](#) designation is a crucial aspect of food safety regulations in the United States, where food additives, chemicals, and substances used in food production must be evaluated for their safety before they can be used in food products. However, some substances are exempt from this pre-market approval process if they are classified as [GRAS](#) by qualified experts.

[GRAS](#) substances are ingredients or additives that have been scientifically proven to be safe under their intended use conditions, or have a long history of common use in food without causing harm. The [FDA](#) maintains a list of [GRAS](#) substances, ensuring transparency and allowing food engineers to innovate while adhering to safety protocols. The [GRAS](#) process enables the development of new food formulations and additives that enhance flavor, texture, and preservation without compromising safety [9].

2.2 Machine learning

Within the vast landscape of [AI](#), we can find the field of [Machine Learning \(ML\)](#), as shown in Figure 2.1, where [AI](#) systems can learn from data and improve their performance over time without being explicitly programmed. This kind of algorithms and models can identify patterns, extract insights, and make

predictions based on large amounts of data. Machine learning encompasses various approaches, each tailored to different learning scenarios. These approaches can be broadly categorized into three fundamental paradigms: supervised learning, unsupervised learning, and reinforcement learning [10]. Each of these paradigms offers unique methodologies for teaching AI systems to adapt, evolve, and make intelligent decisions, making them powerful tools in solving a wide range of complex problems.

- **Supervised Learning:** Supervised learning encompasses methods where the model is trained on a labeled dataset, which means that the input data is paired with the correct output (or target). The algorithm's goal is to train a model that correctly maps the input to the output and, therefore, can be used to predict the correct output for new, unseen data [10].
- **Unsupervised Learning:** Unsupervised learning, in contrast to supervised learning, involves training on an unlabeled dataset, where the algorithm aims to find patterns, structures, or relationships in the data without explicit guidance in the form of labeled output. In essence the algorithm must discover the inherent structure of the data [10].
- **Reinforcement Learning:** Reinforcement Learning (RL) is a type of machine learning where an agent learns from the environment by making sequences of decisions in order to maximize a cumulative reward. The agent receives feedback in the form of rewards or penalties based on its actions, and it uses this feedback to improve its decision-making process over time [10].

2.3 Deep learning

Deep Learning (DL) is a subfield of machine learning, as shown in Figure 2.1, that seeks to mimic the human brain's ability to process information by building artificial neural networks, particularly deep neural networks, with multiple layers of interconnected artificial neurons. These artificial neurons, organized into layers, are designed to automatically capture and transform data representations from raw input. What sets deep learning apart is its capacity to autonomously learn increasingly abstract features from vast datasets [10]. This capability empowers deep learning models to excel at a wide array of complex tasks, such as image [11] and speech recognition [10], natural language understanding [10], recommendation systems [12], and even playing board games with superhuman capabilities [13].

With the surge of computational power [14] and the availability of large-scale datasets [15], deep learning has grown into a dominant force within the artificial intelligence landscape, continually pushing the boundaries of what machines can achieve in terms of perception, cognition, and decision-making [10].

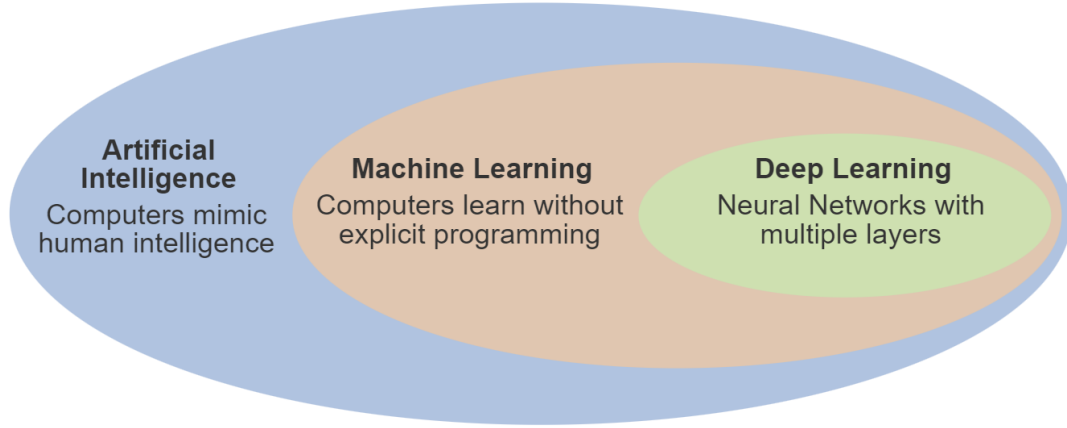


Figure 2.1: Relationship between AI, ML and DL.

2.3.1 Artificial neuron

At the core of deep learning lies the artificial neuron, a fundamental unit in artificial neural networks. An artificial neuron is a mathematical model inspired by the biological neuron but designed for computation. As shown in Figure 2.2 and Equation 2.1, it takes a set of inputs, each associated with a weight, and computes a weighted sum of these inputs. This sum is then passed through an activation function, which introduces non-linearity and determines the output of the neuron.

For the artificial neuron to be useful in learning tasks, its weights need to be adjusted in a way that reduces the difference between the network's predictions and the actual target values. This is done through a process called backpropagation, where the error between the output and the target is calculated and then used to update the weights. This process is repeated many times across the dataset, allowing the neuron to "learn" and improve its predictions over time [16].

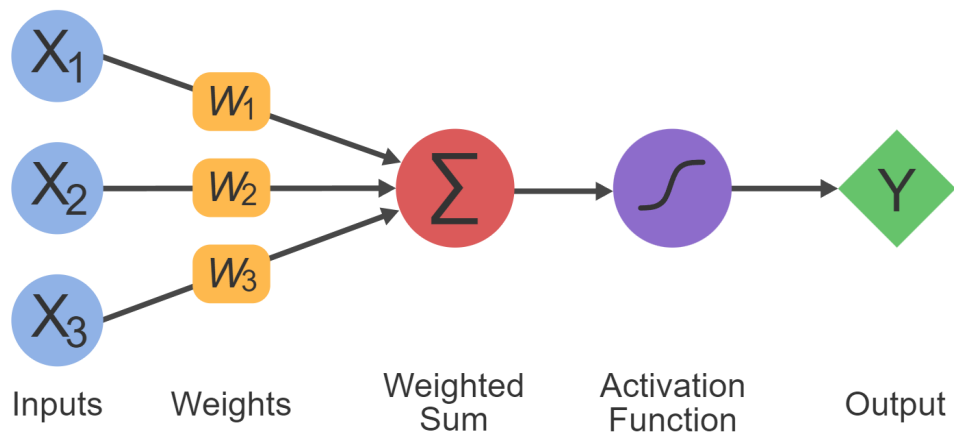


Figure 2.2: The artificial neuron [16].

$$\text{Output} = \text{Activation Function} \left(\sum_{i=1}^n (\text{Input}_i \times \text{Weight}_i) + \text{Bias} \right) \quad (2.1)$$

Activation functions

The output of each neuron passes through an activation function. Activation functions fall in two major categories:

- **Fixed-Shape Activation Functions:** Fixed-shape activation functions are predetermined, non-trainable functions that are applied to the output of neurons in a neural network. These functions, shown in Figure 2.3 and 2.4, have fixed shapes and parameters that do not change during the training process. They are defined mathematically and are the same for all neurons within a particular layer [17].

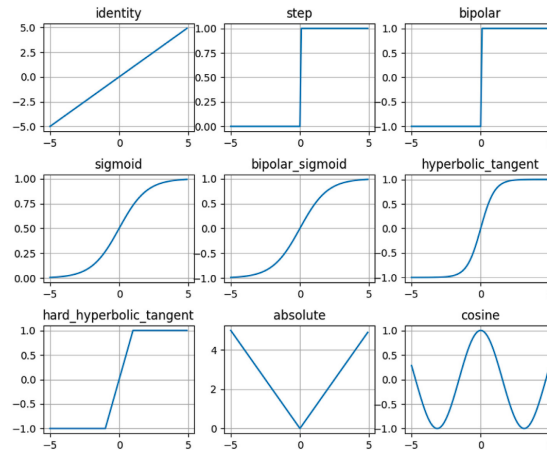


Figure 2.3: Examples of classic activation functions [17].

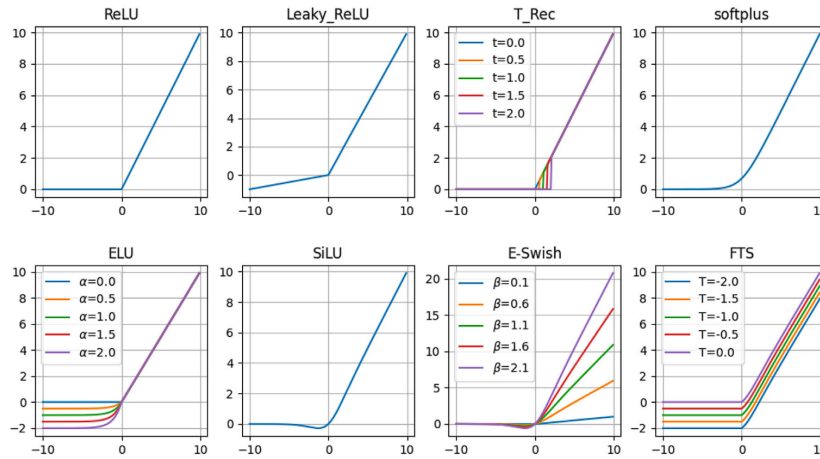


Figure 2.4: Examples of rectifier-based activation functions [17].

- **Trainable Activation Functions:** Trainable activation functions, on the other hand, are functions whose parameters are learned and adjusted during the training process, as shown in Figure 2.5. They are an extension of fixed-shape activation functions and are often used to provide the neural network with more flexibility in modeling complex data patterns [17].

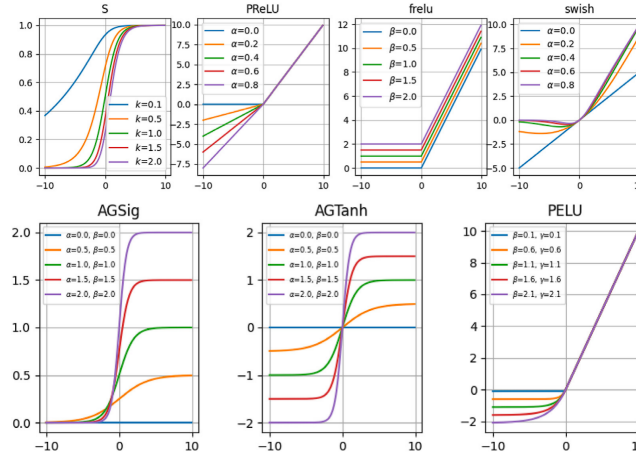


Figure 2.5: Examples of parameterized standard activation functions [17].

2.3.2 Deep neural networks

Deep Neural Networks (DNNs), or **Multilayer Perceptrons (MLPs)** [18] in their simplest form, are a natural progression from the single-layer networks. These networks are constructed by connecting multiple layers of artificial neurons, with each layer consists of several neurons (fully connected layer). The neurons in one layer are connected to those in the next layer, as shown in Figure 2.6, creating a hierarchical structure that allows for the learning of complex, non-linear relationships in data.

One of the key advantages of **DNNs** over single-layer models is their ability to capture and represent hierarchical features. Unlike a single-layer network, which can only model linearly separable functions, **DNNs** can approximate highly non-linear functions by composing multiple layers of feature transformations. Each layer in a **DNN** extracts increasingly abstract features from the raw input data, making them highly effective for tasks such as image classification, natural language processing, and other complex problem domains [19].

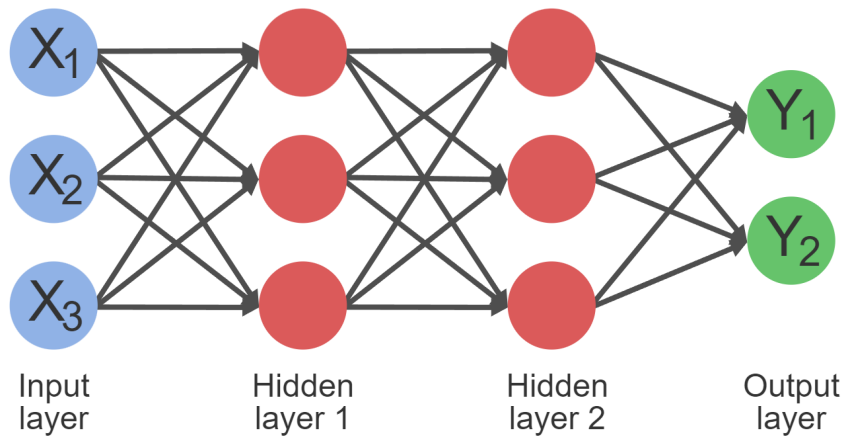


Figure 2.6: Example architecture of a **Deep Neural Network (DNN)**.

2.3.3 Training a neural network

To train a neural network, several methods can be used but, in supervised learning the most commonly used is Backpropagation [20]. It is a mathematical technique that updates the network weights proportionally to their influence in the network error, allowing it to learn.

The process begins with a forward pass. Input data flows through the network, where each neuron calculates a weighted sum of its inputs, applies an activation function, and passes the result to the next layer, as described in Section 2.3.1. The forward pass continues until the network produces an output.

After the forward pass, a loss function is applied to measure the disparity between the predicted output and the actual target. The aim is to minimize this loss since a lower loss indicates a better-performing model.

In the backward pass, the network is ran in the reverse direction, making use of the chain rule from calculus, shown in Equation 2.2, in a process called gradient descent to compute the gradients with respect to the model's parameters. These gradients provide information about how each parameter contributes to the overall loss and, in what direction they should be updated [21].

$$\frac{d}{dx}F(g(x)) = F'(g(x)) \cdot g'(x) \quad (2.2)$$

Loss functions

A loss function quantifies how well or poorly a neural network performs on a given task. It provides a measure of the difference between the predicted outputs of the network and the actual target values. This difference is often referred to as the "error" or "loss." The primary objective in training a neural network is to minimize this loss, thereby improving the network's performance and predictive accuracy [21].

- **Mean Squared Error:** It is used in regression tasks and calculates the average of the squares of the errors between the predicted values and the actual target values [21]. The formula for MSE is given by the Equation 2.3.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2.3)$$

where:

N is the number of data points.

y_i is the true value.

$\hat{y}_i$ is the predicted value.

- **Cross-entropy Loss:** Is primarily used for classification tasks. It measures the performance of a classification model whose output is a probability value or distribution between 0 and 1. The cross-entropy loss increases as the predicted probability diverges from the actual label [21]. For binary

classification, the cross-entropy loss is defined as shown in Equation 2.4.

$$\text{Cross-Entropy} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (2.4)$$

where:

N is the number of data points.

y_i is the true label (0 or 1).

p_i is the predicted probability of the positive class.

Gradient descent variants

When computing the gradients, several approaches can be used [22].

- **Batch Gradient Descent:** Batch gradient descent, or vanilla gradient descent computes the gradient of the cost function with respect to the parameters for the entire training dataset and updates the model accordingly. The update involves subtracting the product of the learning rate and the gradient from the current parameter values, as shown in Equation 2.5. However, it can be slow for large datasets and does not support online updates. The algorithm iterates over the dataset for a fixed number of iterations (epochs), updating parameters based on the computed gradients. Despite its limitations, batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces [22].

$$\theta = \theta - \alpha \cdot \nabla J(\theta) \quad (2.5)$$

where:

θ is the vector of parameters.

α is the learning rate.

$\nabla J(\theta)$ is the gradient of the cost function with respect to the parameters.

- **Stochastic Gradient Descent (SGD):** In SGD, the parameter update is performed for each training example individually, as shown in Equation 2.6. This individual update approach makes SGD faster and suitable for online learning. However, it introduces high variance, leading to fluctuations in the objective function. These fluctuations, on the one hand, enable it to jump to new and potentially better local minima [22].

$$\theta = \theta - \alpha \cdot \nabla J(\theta; x^{(i)}, y^{(i)}) \quad (2.6)$$

where:

$x^{(i)}$ is the input for the i -th training example.

$y^{(i)}$ is the corresponding output.

- **Mini-batch Gradient Descent:** Mini-batch gradient descent strikes a balance between batch gradient descent and stochastic gradient descent by updating model parameters for every mini-batch of 'm' training examples, as shown in Equation 2.7. This approach offers the advantage of reducing parameter update variance, leading to more stable convergence. Moreover, mini-batch gradient descent leverages highly optimized matrix operations common in state-of-the-art deep learning libraries, enhancing the efficiency of gradient computations for mini-batches. SGD is commonly used even when mini-batches are employed [22].

$$\theta = \theta - \alpha \cdot \frac{1}{m} \sum_{i=1}^m \nabla J(\theta; x^{(i)}, y^{(i)}) \quad (2.7)$$

where:

m is the mini-batch size.

One of the challenges that can be encountered during backpropagation is the issue of vanishing and exploding gradients. These problems are related to the way gradients are computed, as the network's error is propagated backwards through the layers.

- **Vanishing Gradient:** Vanishing gradients occur when the gradients of the loss function with respect to the model's parameters become extremely small as they are backpropagated through the network layers. This can have a detrimental impact on training, as it means that the network is not effectively updating its weights in response to the training data. When gradients vanish, it essentially implies that the network is not learning, and the weights are not changing significantly.

The main cause of vanishing gradients is the use of activation functions with derivatives that are close to zero. For example, the sigmoid and hyperbolic tangent (tanh) activation functions have areas where the derivatives are very close to 0, making it more likely for gradients to vanish. This problem can also arise from very deep paths that the gradient has to go through in the network.

To mitigate vanishing gradients, rectifier-based activations are often used. This has constant derivative values, usually 0 or 1 for most of their input range, which helps to alleviate the vanishing gradient problem. In the case of deep gradient paths, residual connections [23] can be used to reduce the path between the outputs and each layer [24, 25].

- **Exploding Gradient:** Exploding gradients occur when the gradients of the loss function with respect to the model's parameters become extremely large as they are backpropagated. This can lead to numerical instability during training and result in large weight updates that can cause the model to diverge.

The primary cause of exploding gradients is often due to the use of weight initialization or learning rates that are too high. When gradients become extremely large, they can lead to weight updates

that are far too large, causing the model's weights to oscillate wildly and making convergence impossible.

To address this, techniques such as gradient clipping can be employed. Gradient clipping involves scaling the gradients if they exceed a certain threshold, which prevents them from becoming too large and causing instability during training [24].

Once the gradients are computed, an optimization algorithm, like stochastic gradient descent is used to update the model's parameters according to the direction of the gradient [18].

Optimizers

Optimizers come into play during backpropagation as they are responsible for determining how much and in what direction the model's parameters should be updated to minimize the loss. Different optimizers use various strategies and heuristics to control the step size and direction of parameter updates. They aim to strike a balance between the need to reach the minimum of the loss function and to avoid overshooting or getting stuck in local minima. One key factor in this optimization is the learning rate, representing the magnitude of the parameter updates. It influences the size of steps taken during optimization, impacting the convergence and stability of the training process [22].

- **Momentum:** Momentum optimization enhances the learning algorithm by introducing a momentum term. This term helps the optimization process by accumulating past gradients and using them to accelerate the descent through areas with steep curves. In essence, momentum allows the optimization algorithm build up speed in the direction it has been consistently moving, facilitating faster convergence and better handling of challenging terrain in the optimization landscape [22].
- **NAG:** Nesterov Accelerated Gradient (NAG) is an optimization algorithm designed to enhance the efficiency of momentum-based approaches by introducing a sense of anticipation. It achieves this by calculating the gradient not with respect to the current parameters but with respect to the approximate future position. The update involves making a big jump in the direction of the previous accumulated gradient, measuring the gradient, and then making a correction. This anticipatory update prevents the algorithm from moving too fast and results in increased responsiveness [22].
- **Adagrad:** Adagrad is a gradient-based optimization algorithm designed to dynamically adjust the learning rate for individual parameters, enabling larger updates for infrequent and smaller updates for frequent parameters. The algorithm's update rule involves modifying the learning rate at each time step based on past gradients computed for each parameter [22].
- **Adadelata:** Adadelata is a smart optimization algorithm that improves upon Adagrad by addressing its aggressive learning rate issue. Instead of storing all past squared gradients, Adadelata uses a decaying average of squared gradients to update model parameters. This prevents the learning

rate from decreasing too quickly. Unlike other methods, Adadelta does not require setting a default learning rate, as it dynamically adjusts during training, making it a self-sufficient and efficient optimization technique [26].

- **RMSprop:** RMSprop is an adaptive learning rate algorithm that tackles the problem of diminishing learning rates in Adagrad by dividing the learning rate by an exponentially decaying average of squared gradients [22].
- **Adam:** The Adam optimization algorithm, short for Adaptive Moment Estimation, is a method that calculates adaptive learning rates for each parameter during model training. Unlike other algorithms such as Adadelta and RMSprop, Adam maintains two exponentially decaying averages: one for past gradients (mt) similar to momentum, and another for past squared gradients (vt) [22].

2.3.4 Overfitting and underfitting

The need to develop models capable of making accurate predictions on new, unseen data is often challenged by the twin adversaries of overfitting and underfitting. These two phenomena represent the delicate balance that models must strike between complexity and generalization. Overfitting arises when models become excessively intricate, capturing noise and idiosyncrasies within the training data, while underfitting occurs when models are too simplistic to discern essential patterns.

Overfitting emerges when a model excessively incorporates noise, outliers, or specific fluctuations from the training data, as shown in Figure 2.7 right, becoming overly complex. This complexity leads the model to memorize the training data, resulting in high sensitivity to these specificities. Consequently, the model exhibits exceptional performance on the training dataset but significantly falters when presented with new, unseen data. Overfitting often arises from models with an abundance of parameters relative to the dataset size, insufficient data for training, or the absence of adequate constraints or regularization techniques to manage model complexity.

In contrast, underfitting materializes when a machine learning model lacks the complexity to grasp the underlying patterns within the training data. This results in a poor estimation of the target function and poor performance both on the training data and new, unseen datasets, as shown in Figure 2.7 left. Underfitting commonly occurs due to overly simplistic models that fail to capture the complexities inherent in the data, limited features or representations, or an excessive use of regularization techniques that overly constrain the model's capacity to learn from the data.

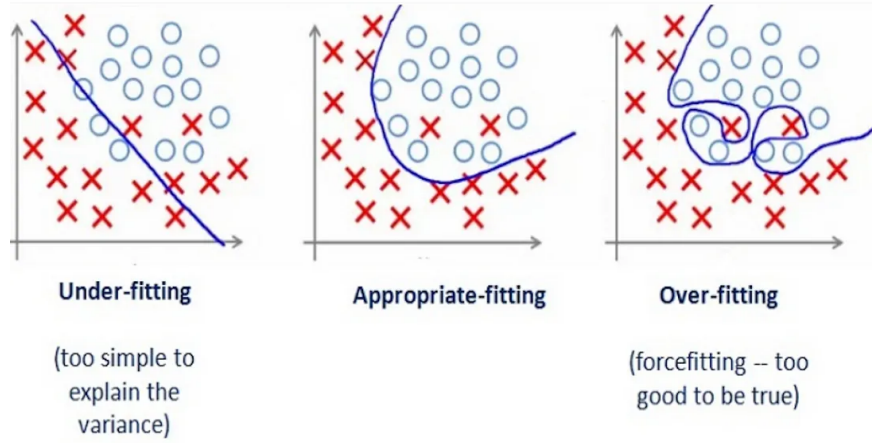


Figure 2.7: Illustration of the phenomena of Overfitting and Underfitting [27].

Regularization techniques

Regularization techniques serve as indispensable tools to mitigate overfitting and enhance the generalization capabilities of neural networks. Regularization, in essence, introduces constraints or penalties to the learning process, guiding the model to avoid overly complex representations that might cater excessively to the training data, but may not generalize well to new, unseen data. These techniques are crucial because they strike a balance between fitting the training data well and retaining the ability to make accurate predictions on new, unseen samples.

- **L1 and L2 Regularization:** Adds a penalty term proportional to the absolute value of the weights, in L1, or the square of the weights, in L2, to the cost function (error of the batch), encouraging sparsity by pushing some weights to zero. It helps in preventing large weight values, promoting a more distributed representation of features [28, 29].
- **Dropout:** Dropout is a technique where randomly selected neurons are ignored or dropped out during training. It is applied as a layer that sets a predetermined percentage of the outputs of the previous layer to 0. This prevents reliance on specific neurons and encourages robustness by preventing co-adaptation of feature detectors. It acts as a form of ensemble learning within a single model, reducing overfitting and enhancing generalization [29, 30].
- **Data Augmentation:** Data Augmentation involves artificially expanding the training dataset by applying transformations like rotation, flipping, scaling, or cropping to the existing samples (in the case of images). This technique introduces variability, making the model more robust and less sensitive to minor variations in the input data distribution [29].
- **Early Stopping:** It monitors the model's performance on a validation set during training. It halts the training process when the model starts to overfit, indicated by a degradation in performance on the validation set, thus preventing the model from learning noise in later training epochs [29].

2.4 Word embeddings

Word Embeddings are a pivotal concept in natural language processing, created to improve the way computers understand and process language. They are dense, continuous-valued vector representations of words, designed to capture the semantic nuances and contextual relationships between words in a high-dimensional space. Word embeddings were created to address the limitations of traditional, sparse word representations like one-hot encoding. Traditional methods lacked the ability to convey word semantics or similarities effectively, hindering many NLP tasks. By learning word embeddings from large corpora of text data, models can understand language with a depth that was previously unattainable [31].

Word embeddings are constructed through supervised learning approaches, notably Word2Vec [32], GloVe [33], and FastText [34]. These methods leverage large text corpora to learn distributed representations of words. Word2Vec, for example, trains a neural network to predict the likelihood of a word given its context or vice versa, learning vector representations as a byproduct. GloVe focuses on global word co-occurrence statistics to derive word embeddings, while FastText introduces sub-word information, enhancing the representation of morphologically rich languages and handling out-of-vocabulary words more effectively [31].

One key property of word embeddings is the semantic similarity reflected in the spatial arrangement of words in the embedding space, as shown in Figure 2.8. Words with similar meanings or contextual usage are positioned closer to each other, forming clusters that encode semantic relationships. This proximity allows models to capture and leverage semantic similarities, enabling them to generalize well and understand context more effectively [31].

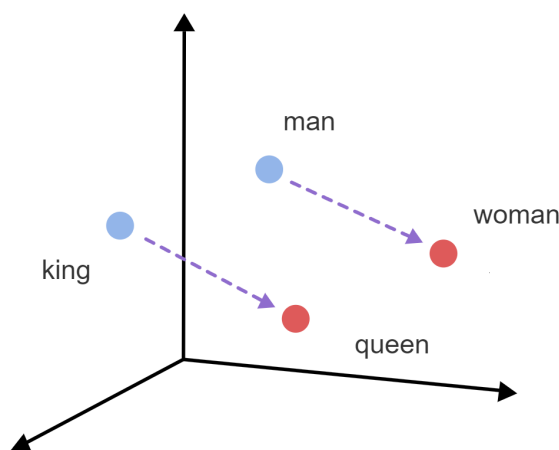


Figure 2.8: Syntactic relationships between word embeddings.

2.5 The curse of dimensionality

The curse of dimensionality refers to the problems that arise when working with high-dimensional data. As the number of variables (dimensions) increases, the data becomes sparse, making it difficult to find patterns or similarities. For example, in a 2D space, points may be close to each other, but in higher

dimensions, they become spread out, and neighborhoods are likely to be empty, as shown in Figure 2.9. This sparsity makes it challenging to gather enough representative data for analysis. This problem can lead to issues like overfitting, where a model becomes too complex and captures noise rather than the actual data patterns. With more dimensions, the distance between points increases, making it harder to identify clusters or correlations. As a result, high-dimensional data often requires more advanced techniques to manage and analyze effectively. This phenomena can also lead to loss of accuracy in vector retrieval systems. Since the meaningfulness of the distance values decreases as the dimensions increase, it becomes more difficult for retrieval systems to find accurate nearest neighbors [35, 36].

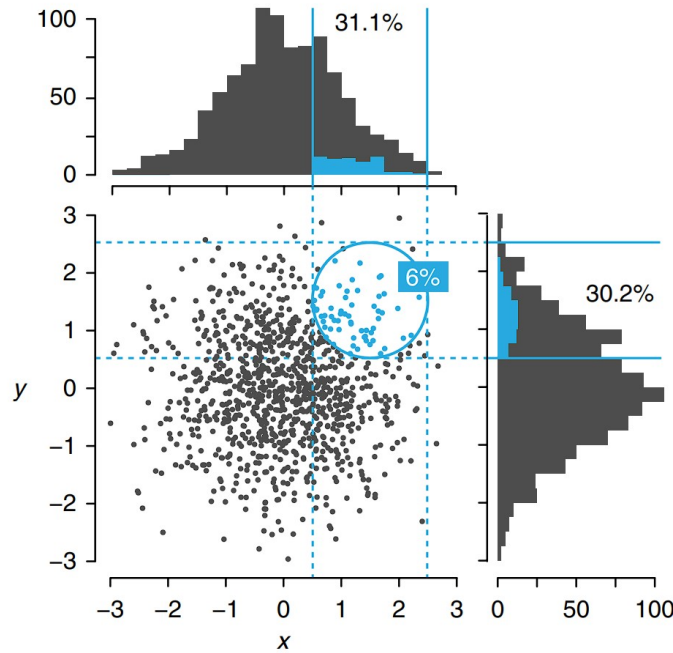


Figure 2.9: Example of the curse of dimensionality. Among 1,000 (x, y) points in which both x and y are normally distributed with a mean of 0 and s.d. $\sigma = 1$, only 6% fall within σ of $(x, y) = (1.5, 1.5)$ (blue circle). However, when the data are projected into a lower dimension - shown by histograms - about 30% of the points (all bins within blue solid lines) are within σ of 1.5. Blue bins in histograms correspond to the blue points [35].

2.6 Specialized neural architectures

Nowadays [Deep Learning \(DL\)](#) has to tackle very complex problems and simple architectures are not capable enough for all kinds of tasks.

To address these problems, task specific network structures were created, which are proven to be highly accurate and often outperform the traditional [Feedforward Neural Networks \(FNNs\)](#) in a range of real-world applications. The advantages of specialized neural architectures over traditional [FNNs](#) can be attributed to their ability to overcome some of the limitations inherent in the latter:

- **Handling complex data:** One of the key advantages of neural architectures is their adaptability to complex data types. While [MLPs](#) treat all data uniformly, some neural architectures are specifically

designed to deal with spatial and sequential data [37].

- **Parameter efficiency:** Specialized architectures often utilize fewer parameters than [MLPs](#) by employing weight sharing or layer sharing, significantly reducing the number of parameters in the model, leading to faster training and more robust models [37].
- **Efficiency and speed:** Specialized architectures are engineered for efficiency, which translates into faster training and inference times. Some mechanisms like convolutions and shared weights allow for parts of the network to run in parallel, reducing time and cost during training and inference [37].

Some architectures need to be flexible enough to accommodate a range of applications, from tasks with simple one-to-one relationships to those with complex many-to-many relationships. To accommodate the diverse requirements of different tasks and types of data, different architectural configurations are used in neural networks to handle various input-output relationships.

- **One-to-One:** In a one-to-one model, there is a one-to-one relationship between the input and output. It is the simplest type, resembling traditional [Feedforward Neural Networks \(FNNs\)](#) [18].
- **One-to-Many:** In a one-to-many model, a single input leads to multiple outputs or timesteps. Useful in tasks such as image captioning [38].
- **Many-to-One:** In a many-to-one model, multiple inputs lead to a single output. This is often employed in tasks where multiple pieces of information are combined to make a single decision such as sentiment analysis [39].
- **Many-to-Many:** In a many-to-many model, the neural network takes a sequence of inputs and produces a sequence of outputs. Usually needed in language generation or language translation tasks [40].

2.6.1 Convolutional neural network

[Convolutional Neural Networks \(CNNs\)](#) [41] represent a specialized type of neural architecture designed primarily for capturing spacial patterns, which makes them the primary architecture for tasks involving visual data, such as image and video analysis. [CNNs](#) have played a pivotal role in revolutionizing the field of computer vision, making it possible for machines to "see" and understand images in a manner reminiscent of human perception.

At the core of a [CNN](#) are convolutional layers, which are fundamental to its operation. These layers enable the network to scan an input image with a set of learnable filters, also called kernels. These filters slide across the input image, performing a convolution operation, which involves element-wise multiplication of the filter with a local region of the image, followed by summation. The result of this operation is called a feature map, which highlights the presence of particular patterns or features in the image. [CNNs](#) typically

stack several convolutional layers to hierarchically extract increasingly more complex features from the image, as shown in Figure 2.10. These can then be fed to fully connected layers to classify the image [42, 43].

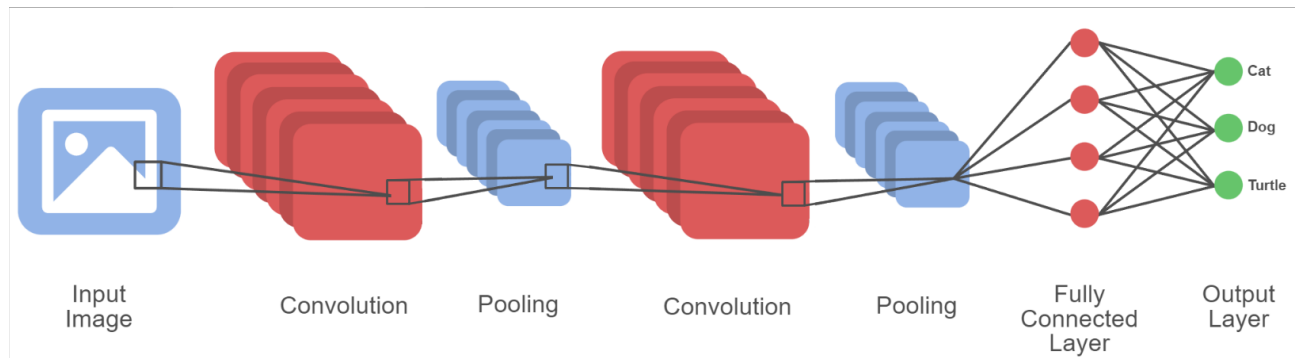


Figure 2.10: Convolutional Neural Network (CNN) Architecture.

2.6.2 Recurrent neural network

Recurrent Neural Networks (RNNs) are a type of neural architecture designed to process sequences of data, making them a fundamental tool in the realm of natural language processing, speech recognition, time series analysis, and many other applications. RNNs were invented to address the limitations of MLPs regarding sequential data. They were developed to handle data with temporal dependencies, where the order and context of elements are critical.

What sets RNNs apart from other architectures is their recurrent connections. These connections create a feedback loop, allowing information to be passed from one step in the sequence to the next. For instance, in language processing, each step could represent a word in a sentence or a character in a text.

At each step, an RNN takes an input from the sequence (x_t) and the previous hidden state (h_{t-1}) and produces the new hidden state (h_t) that serves as a memory of previous steps, as shown in Figure 2.11. This hidden state is updated at each time step and influences the current step's output. This allows the network to use the same set of weights in each time step, facilitating the learning of patterns and dependencies across the entire sequence [44, 45].

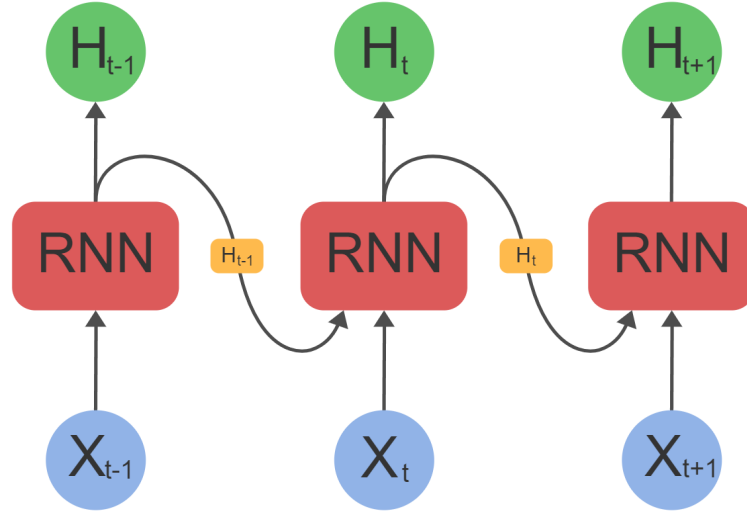


Figure 2.11: Recurrent Neural Network (RNN) unrolled in time.

2.6.3 Long short-term memory

Long Short-Term Memory (LSTM) [46] is a specialized RNN architecture designed to address the limitations of traditional RNNs in processing and modeling sequential data. Its inventors, Hochreiter and Schmidhuber, sought to create a more powerful architecture that could capture long-range dependencies, making it particularly well-suited for tasks such as speech recognition, machine translation, and video analysis, where context over extended sequences is crucial.

LSTMs are capable of achieving this long-term memory by incorporating a cell state, which serves as a persistent memory. The cell state can be updated and maintained over many time steps, allowing important information to be stored and passed along from one time step to another without significant loss. A set of input, forget, and output gates, controlled by sigmoid and tanh functions, regulate the flow of information into, out of, and within the cell state. This gate mechanism effectively mitigates the vanishing gradient problem, enabling LSTMs to capture and remember long-range dependencies in sequential data.

At each time step, it takes in an input (x_t), the previous cell state (C_{t-1}) and the previous hidden state (h_{t-1}) and produces the new hidden state (h_t) and the new cell state (C_t), as shown in Figure 2.12 [45].

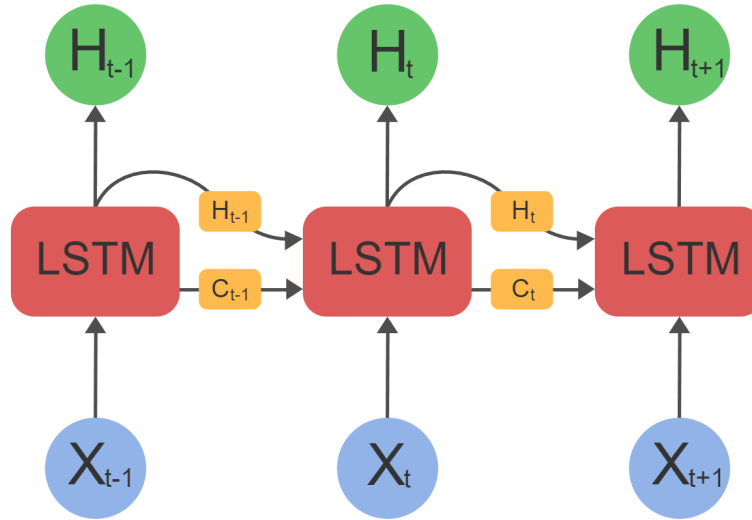


Figure 2.12: Long Short-Term Memory (LSTM) unrolled in time.

2.6.4 Transformer

The Transformer [47] is a groundbreaking neural network architecture that had a profound impact on various NLP and machine learning tasks. Transformers are sequence-to-sequence models and were created to address several limitations of earlier sequence models.

- **Long-range dependencies:** Transformers solve the long-range dependency problem by employing an attention mechanism that allows each input token to attend to all other tokens in the sequence, enabling the model to capture and relate distant contextual information effectively, without the vanishing gradient issues encountered in traditional recurrent networks [47].
- **Parallelization:** Since the attention mechanism in transformers does not depend on sequential operations, the computations can be parallelized across elements of a sequence. This enables faster training and inference compared to sequential models, making them highly efficient and suitable for GPUs and TPUs [47].
- **Reducing vanishing gradient problem:** Transformers utilize residual connections and layer normalization to stabilize the training of deep neural networks, effectively addressing the vanishing gradient problem, which is common in deep RNNs [25].

Transformers consist of an encoder and decoder stack, as shown in Figure 2.13, where the encoder processes the input sequence and the decoder generates the output sequence. Within each layer of the encoder and decoder, attention modules are employed to weigh the significance of different tokens in the input sequence, allowing the model to focus on relevant information. Additionally, feed-forward neural networks, residual connections, and layer normalization are integrated into the architecture to enhance the model's representational power and facilitate training. The decoder has an additional cross-attention layer that performs attention over the encoder's outputs [47].

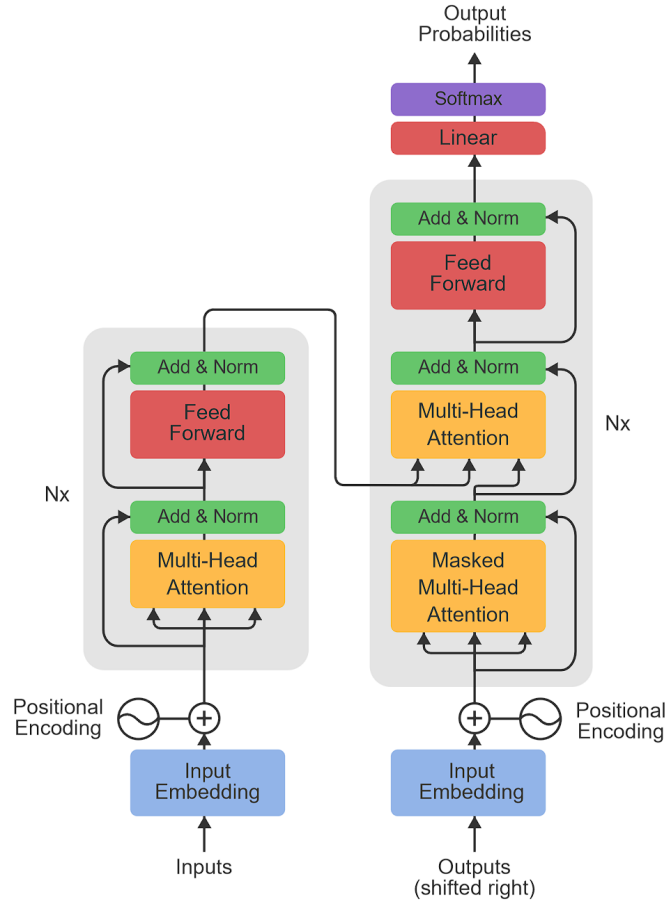


Figure 2.13: Transformer Encoder (left) Decoder (right) Architecture [47].

Self Attention mechanism

Attention [48], is a mechanism that allows a model to weigh the importance of different elements in an input sequence when making predictions or encoding information. It mimics the way humans focus on specific parts of a scene or a sentence to understand and process it better. Attention is characterized by some key components.

- **Queries:** The token for which we want to compute the attention weights [48].
- **Keys:** All other tokens, which serve as reference points for the attention calculation [48].
- **Values:** The information associated with each key token [48].
- **Attention Scores:** A measure of how much attention each key token should receive from the query token (correlation between queries and keys) [48].

The attention mechanism calculates the correlation between the queries and the keys (attention scores), then a softmax function is applied in order to normalize the scores and multiplies them with the values, as shown in Figure 2.14 and Equation 2.8. To calculate the attention scores, a score function (also called

energy function [49]) is needed. The original transformer [47] architecture uses scaled dot-product attention (or scaled multiplicative attention) because it is computationally less expensive than other types of attention, referenced in Table 2.1 [48].

Table 2.1: Types of score function f . k is a key. v, b, W, W_1, W_2 are learnable parameters. d_k is the dimension of the input vector. The act is a nonlinear activation function, such as $\tanh$ and ReLU .

Name	Equation	Ref.
Additive	$f(q, k) = v^T \text{act}(W_1 k + W_2 q + b)$	[50]
Multiplicative (dot-product)	$f(q, k) = q^T k$	[50]
Scaled multiplicative	$f(q, k) = \frac{q^T k}{\sqrt{d_k}}$	[47]
General	$f(q, k) = q^T W k$	[51]
Concat	$f(q, k) = v^T \text{act}(W[k; q] + b)$	[51]
Location-based	$f(q, k) = f(q)$	[51]
Similarity	$f(q, k) = \frac{q^* k}{\ q\ \ k\ }$	[52]

In some situations, when computing attention, an attention mask is needed to control or modify the behavior of the attention mechanism. They operate by being added to the attention scores before softmax, restricting the flow of attention between certain pairs of tokens. For example, when predicting time series, the model cannot pay attention to future tokens to predict the current token. Therefore, the attention mask (a matrix of 0s and $-\infty$ s) is added to the attention scores, which after the softmax results in attention values of zero for illegal connections.

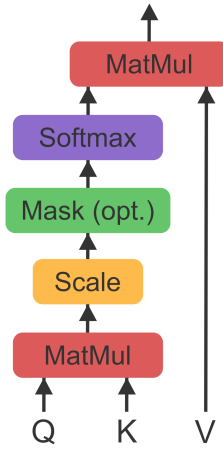


Figure 2.14: Scaled Dot-Product Attention [47].

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \quad (2.8)$$

Multi-head attention

Multi-head attention is a natural progression of the basic attention mechanism in transformers. It extends the concept of attention by allowing the model to focus on different aspects or features of the input sequence simultaneously. This results in a richer representation of relationships and dependencies within

the data.

Multi-head attention addresses the attention limitations by introducing multiple sets of query, key, and value vectors, each called an "attention head", as shown in Figure 2.15. These heads work in parallel and learn different aspects of the input sequence. Each head's queries, keys and values pass through a linear layer (called a projection layer) before computing the attention. In the end all heads are concatenated and pass through a last linear layer [47].

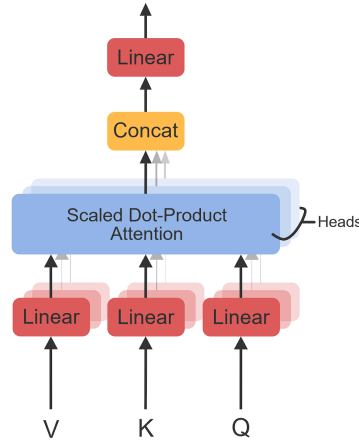


Figure 2.15: Multi-Head Attention [47].

Positional encoding

Positional encoding is a critical component of transformers that addresses one of the fundamental limitations of the model: the lack of positional information. Transformers process input data, such as text or sequences, in parallel, however, they do not have an inherent sense of order or position within a sequence. To overcome this limitation, positional encoding is added to the input embeddings to provide the model with information about the position of each token within the sequence [47].

Positional encoding is typically achieved by adding a position embedding to the input embeddings of the tokens. These positional embeddings are designed in a way that encodes information about the position of the token within the sequence. There are several common methods to create positional encodings:

- **Absolute Positional Encoding:** These types of encoding provide a unique representation for each position in the sequence that is added to each word embedding representing its absolute position in the sequence in a way that can be interpreted by the model.

Sine and cosine functions

The original transformer model introduced a method based on the use of sine and cosine functions to create positional encoding. This approach assigns a unique frequency to each position in the sequence, ensuring that the positional information is encoded in a way that the model can interpret during training [47].

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{(2i/d_{\text{model}})}}\right) \quad (2.9)$$

$$PE(pos, 2i + 1) = \cos\left(\frac{pos}{10000^{(2i/d_{\text{model}})}}\right) \quad (2.10)$$

Learned Positional Embeddings

Instead of using fixed functions like sine and cosine, you can also learn positional embeddings. In this approach, the model learns a set of embeddings for each position during training. These embeddings are added to the token embeddings directly, and the model updates them as it learns from data [53].

- **Relative Positional Encoding:** In contrast to absolute positional encodings, relative positional encodings provide information about the relative distance between tokens. This method is particularly useful in scenarios where the position of tokens relative to each other is more important than their absolute positions, like language modeling. Relative positional encodings allow the model to focus on the relationships between tokens, regardless of their positions in the overall sequence.

Rotary Position Embeddings

[Rotary Positional Encoding \(RoPE\)](#) is a method of encoding positional information that combines aspects of both absolute and relative positional encodings. [RoPE](#) embeddings rotate the input embeddings in a way that incorporates positional information directly into the attention mechanism. The rotation is done by multiplying the embeddings by a rotation matrix. This method maintains the benefits of relative positional encodings by preserving the relative distances between tokens while also providing a unique representation for each position, improving the performance of transformer models, particularly in tasks involving long sequences [54].

$$\mathbf{R}(p) = \begin{bmatrix} \cos(p) & -\sin(p) & 0 & 0 & \cdots & 0 & 0 \\ \sin(p) & \cos(p) & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos(p) & -\sin(p) & \cdots & 0 & 0 \\ 0 & 0 & \sin(p) & \cos(p) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos(p) & -\sin(p) \\ 0 & 0 & 0 & 0 & \cdots & \sin(p) & \cos(p) \end{bmatrix} \quad (2.11)$$

2.7 Large language models

[Large Language Models \(LLMs\)](#) represent a groundbreaking advancement in the field of [Artificial Intelligence \(AI\)](#) and [Natural Language Processing \(NLP\)](#). These models are designed to mimic and understand human language to an unprecedented degree.

At their core, [Large Language Models \(LLMs\)](#) are [Artificial Neural Networks \(ANNs\)](#) that have been trained on massive datasets containing text from the internet, books, articles, and various other sources

[15]. These models are known for their sheer size and complexity, often consisting of tens of billions of parameters (weights) [2]. Most of them make use of the transformer architecture [47], which enables them to process and generate text in a more coherent and contextually relevant manner than ever before [2, 3], mainly because of their capacity to model long-range dependencies [47].

To train an LLM, several tasks can be used to help models develop a deep understanding of language.

- **Masked Language Modeling (MLM):** In MLM, a certain percentage of words in a sentence are randomly replaced with a special "[MASK]" token. The model's objective is to predict the masked words based on the surrounding context [55].
- **Autoregressive (or Causal) Language Modeling:** In Autoregressive Language Modeling, models predict the next word in a sentence based on the preceding words (using attention masks). This task trains models to generate text sequentially and understand dependencies between words in a forward direction [3].
- **Permutation Language Modeling:** This approach, used in models like XLNet [56], models the joint probability of a sequence of words in any order. It involves predicting words in random orders, which helps the model capture complex dependencies in the data [56].
- **Next Sentence Prediction (NSP):** NSP is used in models like BERT. The model is trained to predict whether two sentences are consecutive in the original text. This task helps the model understand sentence-level relationships [55].
- **Translation Language Modeling:** In this task, a model is trained to translate text from one language to another. It helps models understand the relationships between words and phrases in different languages [47].

Language modeling is a computationally expensive and complex task, and research like the GPT-3 LLM [2] demonstrates how increasing the parameter count leads to better model performance across tasks. As such, over the last few years parameter count of state of the art models have been consistently increasing, as shown in Figure 2.16.

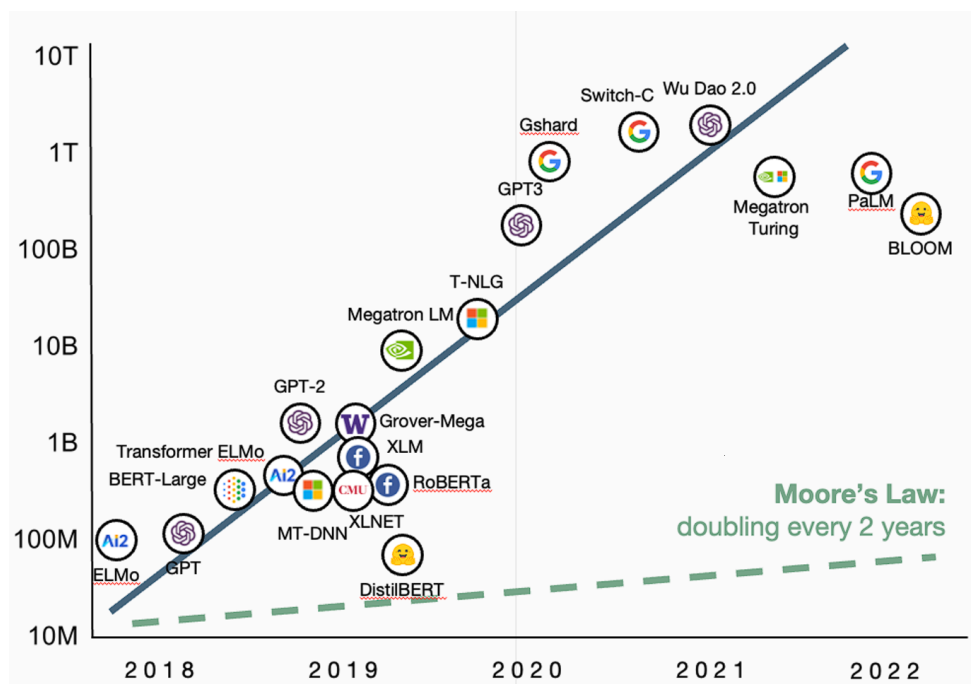


Figure 2.16: LLMs parameter count from 2018 to 2022 [57].

To train a **Large Language Model (LLM)** for chat applications, training can be divided in three main stages, pre-training, fine-tuning and **RLHF**. These stages are completed sequentially and each latter one uses less training data than the previous.

2.7.1 Pre-training

Pre-training is the initial phase where the model is trained on a massive corpus of text data to learn language patterns, grammar, world knowledge, and general language understanding. During pre-training, the model learns to predict the next word in a sentence based on its context, effectively learning the statistical properties of language.

During this stage the model is trained on massive amounts of data, usually in an autoregressive way (predicting the next word). It starts by collecting an extensive and diverse dataset of text from the internet. This corpus includes a wide range of text sources, such as books, articles, websites, and more. The data can be in multiple languages and cover a variety of topics.

During pre-training several characteristics of the dataset can severely impact the training quality.

- **Dataset size:** The more tokens in the training data, the more diverse linguistic patterns and contexts the model can learn. A larger dataset helps the model understand a wider range of language nuances, leading to potentially better performance. The Chinchilla LLM [58] showed that GPT-3 [2] and similar models should have used 11x more training data. The Chinchilla scaling laws showed that we need around 20 text tokens per model parameter [58].
- **Dataset quality:** High-quality datasets significantly impact the performance of models like the Phi-1 SLM [59]. This model emphasizes how the use of clean, specialized, and non-contradictory

data plays a crucial role in determining the final quality of the model. When the training data is carefully curated to be free of errors, follows a specific structure, focuses on specific domains, and lacks conflicting information, it greatly shapes the capabilities of the model. For instance, clean data ensures the model does not learn inaccurate language patterns, specialized data helps it excel in particular areas allowing for a smaller model size, and non-contradictory information aids in building coherent understandings that do not confuse the model. These elements collectively contribute to an enhanced model that demonstrates a superior grasp of language nuances and delivers more accurate and reliable outputs across various tasks and scenarios [59].

Sampling from an LLM

LLMs are trained to produce a vector of logits (raw model outputs) that are converted into probabilities using the softmax function. Generating text involves sampling from these distributions using certain probabilistic strategies.

- **Temperature:** Temperature is a parameter that controls the randomness of the predictions made by an LLM. It scales the logits before applying the softmax function to convert them into probabilities. A higher temperature (e.g., $T > 1$) makes the probability distribution more uniform, leading to more diverse and less deterministic output. Conversely, a lower temperature (e.g., $T < 1$) sharpens the probability distribution, making the model more confident in its choices and thus less random [60].
- **Top-K Sampling:** Top-k sampling restricts the model's output to the top k most probable tokens. This technique involves selecting from a truncated distribution, where only the k most likely tokens are considered. The remaining tokens are assigned zero probability. This method helps control the diversity of the generated text by focusing on a smaller subset of tokens [60].
- **Top-P Sampling (Nucleus Sampling):** Top-p sampling, also known as nucleus sampling, dynamically selects a subset of tokens such that their cumulative probability mass exceeds a threshold p . Instead of choosing a fixed number of top tokens as in top-k sampling, top-p sampling adjusts the size of the subset based on the cumulative probability of the distribution [60].
- **Repetition Penalty:** Repetition penalty is used to reduce the likelihood of generating repeated or overly repetitive text. This technique penalizes tokens that have already appeared in the generated text, thereby discouraging the model from repeating the same token or sequence of tokens [61].

2.7.2 Fine-tuning

Fine-tuning is the process of taking a pre-existing model that has been trained on a vast amount of general data and adapting it to perform a specific task or to specialize in a particular domain. This approach involves adjusting the parameters of an already trained model by exposing it to new, task-specific data to enhance its performance on that particular task.

The core principle behind fine-tuning is leveraging the knowledge and learned representations encoded within a pre-trained model and fine-tuning its parameters to better suit a narrower, more specific task or domain. In natural language processing, a pre-trained LLM like BERT [55] or GPT [3] may have learned general language patterns from a wide range of text sources. Fine-tuning allows tailoring this model to perform tasks such as sentiment analysis, text classification, question answering, or language generation within a specialized field like medicine, law, or finance [62].

Pre-trained models have already learned valuable features and patterns from large-scale datasets, thus requiring less data for further adaptation. This significantly reduces the computational resources and time needed to achieve good performance compared to training a model entirely from the ground up. Moreover, fine-tuning capitalizes on transfer learning, enabling the model to benefit from the general knowledge learned during pre-training, which can be crucial, especially when limited labeled data is available for a specific task [4, 62]. Fine-tuning is usually favored over training a model from scratch due to some of its advantages.

- **Fast convergence:** Since the model starts with pre-existing knowledge, it converges faster when exposed to new task-specific data [63].
- **Reduced data required:** Fine-tuning requires less data compared to training from scratch, making it feasible for tasks with limited size datasets [63].
- **Improved performance:** By leveraging the learned representations, fine-tuning often leads to improved performance on the target task compared to training from scratch [63].

However, even being the preferred choice when solving a problem, fine-tuning also presents certain challenges and potential drawbacks.

- **Overfitting:** There is a risk of overfitting to the new task or domain, especially if the fine-tuning dataset is small or not representative [63].
- **Catastrophic forgetting:** Excessive fine-tuning on new data may cause the model to forget previously learned information from the pre-training phase [63].
- **Task-specific bias:** Fine-tuning on specific datasets may introduce biases that were not present in the original pre-trained model, affecting its generalizability [63].

Instruction tuning

Instruction tuning is a type of the fine-tuning that introduces a specialized approach that tailors models not just to generalize well to a broad spectrum of data, but specifically to adhere closely to prescribed instructions or guidelines. Unlike conventional fine-tuning, which optimizes models for overall performance on a specific task or dataset, instruction tuning focuses on optimizing models to follow explicit directives or constraints during inference.

This technique is used to refine and adapt existing LLMs to precisely follow or execute instructions provided in conversational form. The objective is to ensure that the model not only comprehends the instructions, but also accurately executes the implied tasks in a conversational setting. Instruction tuning is the primary tool to allow LLMs to escape the traditional text completion objective into a more useful conversational and instruction following format [64].

2.7.3 Reinforcement learning from human feedback

Reinforcement Learning from Human Feedback (RLHF) [65] stands as a pioneering approach within the realm of LLMs. This method diverges from conventional reinforcement learning paradigms by emphasizing the incorporation of human feedback into the training process, elevating the model's learning trajectory by integrating human guidance.

At its core, RLHF involves a cyclical process where a machine learning model interacts with human-generated feedback to refine its decision-making abilities and enhance its performance on a specified task. In traditional Reinforcement Learning (RL), the architecture typically consists of an agent that interacts with an environment. This agent makes decisions or takes actions based on a policy, and the environment provides feedback in the form of rewards or penalties, signaling the quality of the actions taken by the agent. The agent then adjusts its policy to maximize the cumulative reward over time. The core components of RL architecture include the agent, environment, action space, state space, and a reward mechanism [66].

Conversely, RLHF modifies the traditional RL setup to integrate human feedback as a critical signal in the learning process. One of the key architectural modifications is the inclusion of a "reward model" that predicts the rewards or evaluations based on the feedback received from humans, as shown in Figure 2.17. Usually this model might involve neural networks or other methods trained specifically to predict human-provided evaluations or rewards [65–67].

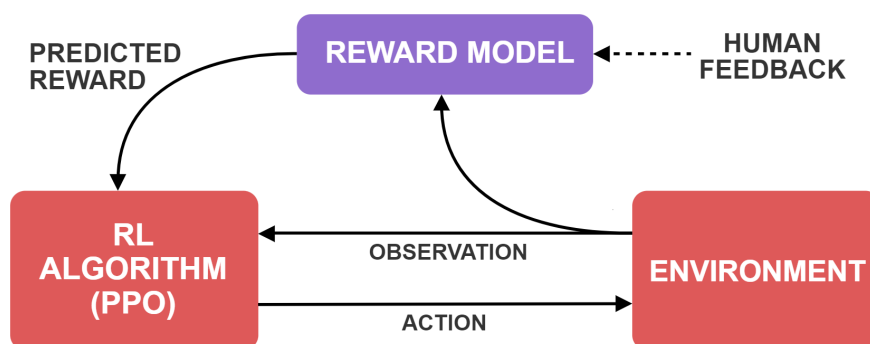


Figure 2.17: RLHF Training Cycle [65].

The adoption of RLHF in LLMs stems from its distinct advantages over traditional training methods like fine-tuning. While fine-tuning leverages pre-existing knowledge within a model and adapts it to specific tasks, RLHF introduces a more dynamic and interactive learning paradigm by directly integrating human insight giving it some advantages over fine-tuning.

- **Adaptability and versatility:** RLHF can handle situations where labeled data or predefined reward functions are unavailable or hard to define. It adapts to different tasks and scenarios based on human-provided feedback, making it versatile [67].
- **Safety and ethical considerations:** RLHF can incorporate human feedback to enforce safety constraints and ethical considerations more effectively, ensuring that the learned behavior aligns with human preferences and norms [67].

While RLHF presents compelling advantages in integrating human expertise and adapting to various scenarios, it also faces some challenges related to the quality, scalability and biases.

- **Feedback quality and consistency:** The quality and consistency of human feedback can vary significantly, impacting the learning process. Inaccurate or inconsistent feedback may hinder the agent's learning or lead to suboptimal behavior [67].
- **Biases:** If the human-provided feedback consistently favors certain actions or behaviors due to cultural or personal inclinations, the learning agent might adopt these biases, resulting in skewed or unfair decision-making [67].
- **Scalability:** Scaling RLHF to large-scale problems might be challenging. It heavily relies on human involvement, which can be time-consuming and costly for extensive or continuous learning scenarios [67].

Proximal policy optimization

At the core of RLHF algorithm is Proximal Policy Optimization (PPO) [68], used for training artificial intelligence agents to perform tasks in environments where actions influence future states. It belongs to the family of policy gradient methods, designed to optimize policies by iteratively improving them based on experiences gathered from the environment.

PPO focuses on optimizing policies in a stable and efficient manner, while ensuring that updates to the policy do not deviate too far from the previous policy. This is achieved by constraining the policy updates using a concept called the "clipped surrogate objective". The key idea behind it is to balance the trade-off between exploring new policies (to potentially find better ones) and exploiting the current policy (to maximize cumulative rewards). It does this by collecting experiences from the environment, evaluating the policy's performance, and then updating the policy to improve its behavior. This cycle continues iteratively, gradually improving the policy's effectiveness [68].

2.7.4 Mixture of experts

The Mixture of Experts (MoE) is an architecture designed to enhance the performance and efficiency of neural networks by dividing tasks among specialized sub-models, or "experts". Instead of relying on a

single model to handle all inputs, **MoE** dynamically selects and combines the outputs of several experts, each of which is trained to specialize in different parts of the input space or to focus on different aspects of the problem [69, 70].

The main components of an **MoE** system are the expert networks, a gating network, and an output combining mechanism. The gating network determines which expert(s) should handle a given input, routing the data accordingly. Each expert then processes its assigned inputs, and their outputs are combined (often through weighted averaging) to produce the final result [69, 70]. This behaviour can be seen in Figure 2.18.

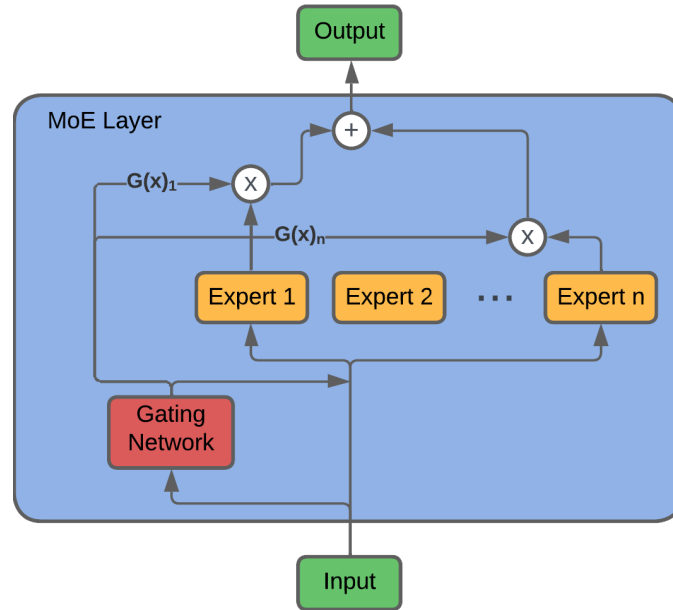


Figure 2.18: **MoE** Layer. In this example the sparse gating network selects two experts to feed the input. Their outputs are modulated by the outputs of the gating network [69].

2.7.5 Prompt engineering

Prompt engineering is a strategic process that involves crafting, refining, and tailoring prompts or instructions given to **LLMs** to elicit desired responses, playing a pivotal role in shaping the behavior and output of **LLMs**, particularly in tasks like text generation or question-answering. The core of prompt engineering lies in its ability to provide specific guidance to models, ensuring they produce accurate, contextually relevant, and task-specific outputs. This precision is achieved by designing prompts that effectively communicate the desired task, constraints, or context to the model.

Crafting the prompt involves designing an instruction that conveys the task accurately to the model. Experimentation and iteration are key components, involving testing different variations of prompts to identify the most effective ones. This may include incorporating context, examples, constraints, or specific wording to guide the model's understanding [71].

Chain-of-thought prompting

One powerful advancement in prompt engineering is the concept of generating a "chain of thought"—a series of intermediate reasoning steps, as shown in Figure 2.19. This approach significantly enhances the ability of [Large Language Models \(LLMs\)](#) to perform complex reasoning tasks. [Chain of Thought \(CoT\)](#) prompting involves providing exemplars that demonstrate a series of intermediate reasoning steps. The method naturally emerges in sufficiently [LLMs](#), offering a simple yet powerful technique to unlock reasoning abilities. Unlike traditional methods that rely on fine-tuning or [RLHF](#) approaches, [CoT](#) prompting leverages the innate capabilities of [LLMs](#) to reason through natural language intermediate steps [72].

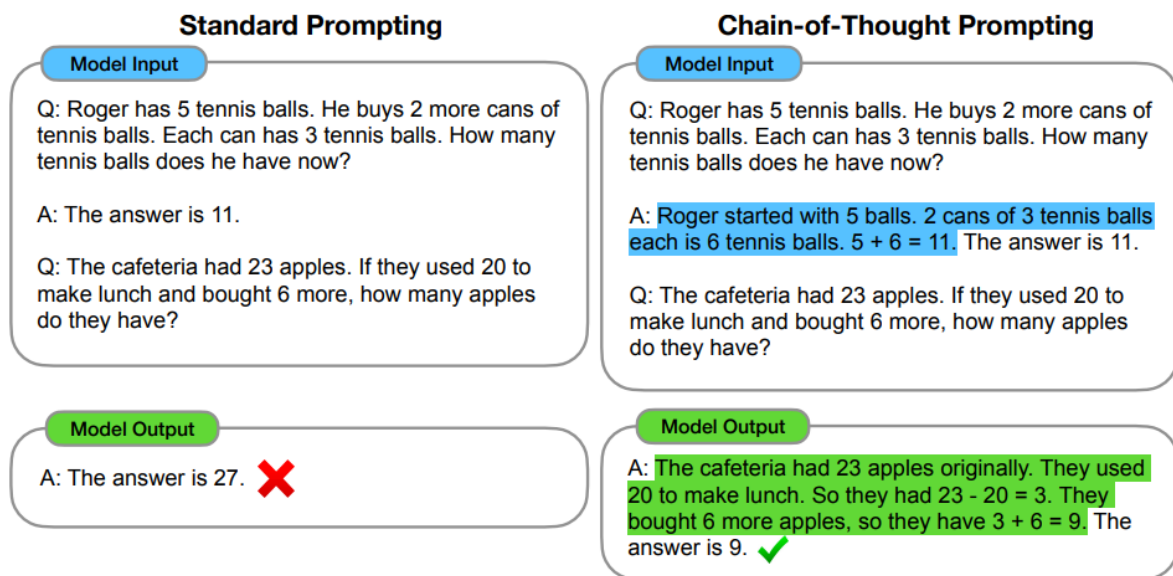


Figure 2.19: [CoT](#) reasoning process [72].

Tree of thoughts

[Tree of Thoughts \(ToT\)](#) is an evolution of [Chain of Thought \(CoT\)](#) prompting. In traditional token-level, left-to-right decision-making processes during inference, [LLMs](#) may fall short in tasks requiring exploration, strategic lookahead, or where initial decisions are pivotal. To address these challenges, the [ToT](#) framework generalizes the [CoT](#) approach, enabling exploration over coherent units of text ("thoughts") that serve as intermediate steps toward problem-solving. The [ToT](#) framework actively maintains a tree of thoughts, where each thought is a coherent language sequence serving as an intermediate step toward problem-solving, as shown in Figure 2.20. This high-level semantic unit allows [LLMs](#) to self-evaluate progress through a deliberate reasoning process instantiated in language. The implementation of search heuristics via language model self-evaluation is a more robust and dynamic problem-solving approach in comparison to traditional token-level decisions [73].

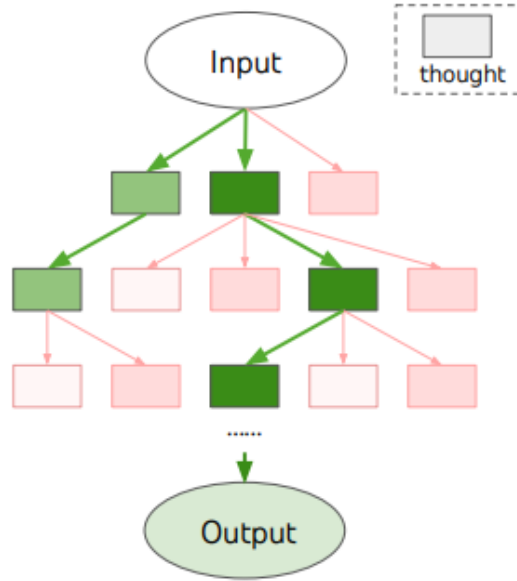


Figure 2.20: ToT reasoning process [73].

2.7.6 Metrics

Evaluating [Large Language Models \(LLMs\)](#) involves assessing how effectively these models understand and generate human-like language. To measure their performance, various metrics are employed, each designed to quantify different aspects of language understanding and generation.

These metrics play a crucial role in the field of [Natural Language Processing \(NLP\)](#) by providing standardized ways to gauge the quality of generated text, aiding the benchmarking of models, comparing different approaches, and tracking progress in language technology.

- **Perplexity:** Perplexity measures how well an [LLM](#) predicts a sequence of words. It quantifies the model's uncertainty or surprise in predicting the next word in a text sequence. It is computed using the probability assigned by the model to each word in a validation or test dataset. A lower perplexity indicates that the model is more confident and accurate in its predictions, reflecting the model's ability to capture the underlying structure and patterns within the data [74].
- **Bilingual Evaluation Understudy (BLEU):** [BLEU](#) assesses the quality of machine-translated text by comparing it to one or more human-generated reference translations. It measures the similarity between n-grams (sequences of words) in the machine-generated text and those in the reference text [75].
- **Recall-Oriented Understudy for Gisting Evaluation (ROUGE):** [ROUGE](#) evaluates the quality of texts by comparing the overlap of n-grams and word sequences between the generated text and human-written references. It is commonly used in summarization tasks and helps quantify how well a summary aligns with human references, indicating the summary's informativeness and conciseness [76].

- **Metric for Evaluation of Translation with Explicit Ordering (METEOR):** METEOR evaluates machine translation quality while emphasizing semantic similarity and incorporating word ordering. It tries to address some of the limitations of BLEU by considering a wider range of language variations and providing a more holistic evaluation of translations [77].

2.7.7 Benchmarks

Benchmarks are indispensable pillars that serve as standardized assessments, providing a common ground for evaluating and comparing the performance of diverse models and algorithms. Crucially, benchmarks streamline the evaluation process by offering a framework that encompasses multiple metrics and real-world application scenarios. This standardization ensures objective assessments that gauge the progress in the field.

One of the standout features of benchmarks lies in their ability to offer a holistic evaluation. They encapsulate various facets of model performance, considering diverse tasks and metrics within a single comprehensive framework. This makes benchmarks particularly valuable for comprehensive model assessments, allowing for a more nuanced understanding of a model's capabilities. Several renowned benchmarks illustrate this depth of evaluation, each highlighting different aspects of language generation.

- **General Language Understanding Evaluation (GLUE):** GLUE aims to benchmark models on a diverse set of natural language understanding tasks, including textual entailment, sentiment analysis, and question answering. It encourages models to exhibit generalization across various linguistic phenomena, evaluating their ability to understand language across different domains and tasks [78].
- **SuperGLUE:** An extension of GLUE, SuperGLUE emphasizes more complex language understanding tasks that require deeper reasoning and contextual understanding. It comprises more challenging tasks, including multi-sentence reasoning, commonsense reasoning, and narrative understanding [79].
- **Stanford Question Answering Dataset (SQUAD):** SQUAD is dedicated to machine reading comprehension, where models are required to answer questions based on provided passages. It includes a collection of Wikipedia articles with associated questions and answers, serving as a benchmark for evaluating a model's ability to comprehend text and extract relevant information [80].
- **HellaSwag:** HellaSwag evaluates commonsense reasoning by providing situations where models must predict the final sentence in a given context, requiring nuanced understanding of common knowledge [81].

2.7.8 Rating

Besides evaluating LLMs against metrics or benchmarking them with evaluation datasets, LLMs can be ranked between themselves in order to determine their relative performance and effectiveness. One common method used for this ranking is the Elo rating system, originally designed for chess. Elo ratings are adapted to compare the models based on their performance in various tasks, such as text generation, question answering, and language understanding. The Elo system assigns a numerical score to each model, which changes based on the outcomes of direct comparisons between pairs of models on benchmark datasets or specific evaluation tasks. The scores reflect the relative performance of one model compared to another.

$$R' = R + K(S - E) \quad (2.12)$$

where:

- R is the current rating of the player.
- R' is the new rating of the player.
- K is a constant (the K-factor) that determines the maximum possible change in rating.
- S is the actual score in the game (1 for a win, 0.5 for a draw, and 0 for a loss).
- E is the expected score, calculated using:

$$E = \frac{1}{1 + 10^{(R_{\text{opponent}} - R)/400}} \quad (2.13)$$

where R_{opponent} is the current rating of the opponent and R is current rating of the player.

2.8 Retrieval augmented generation

Retrieval Augmented Generation (RAG) is a technique that combines the strengths of retrieval-based and generative models to enhance the quality and relevance of generated text. It involves integrating a retrieval mechanism, often based on dense vector search, with a generative LLM, as shown in Figure 2.21.

In RAG, the retrieval aspect involves leveraging a large repository, such as a document database or a collection of passages, to retrieve contextually relevant information. This retrieved information acts as a source of knowledge, guiding the generative model during text generation. The generative model then uses this retrieved information to produce coherent, contextually relevant, and informative text.

One of the primary motivations behind RAG is to address the limitations of purely generative models, such as their tendency to generate factually inaccurate or irrelevant information. By incorporating retrieval, RAG models can access a vast amount of structured or unstructured data, enabling them to ground their generated text in factual or contextually relevant information.

This mechanisms allow for improved factual accuracy, enhanced coherence, and relevance in generated text. By retrieving information from a knowledge source, [RAG](#) models have a higher likelihood of producing accurate and contextually appropriate responses, making them suitable for tasks requiring knowledge integration and factual correctness, like question answering [\[82\]](#).

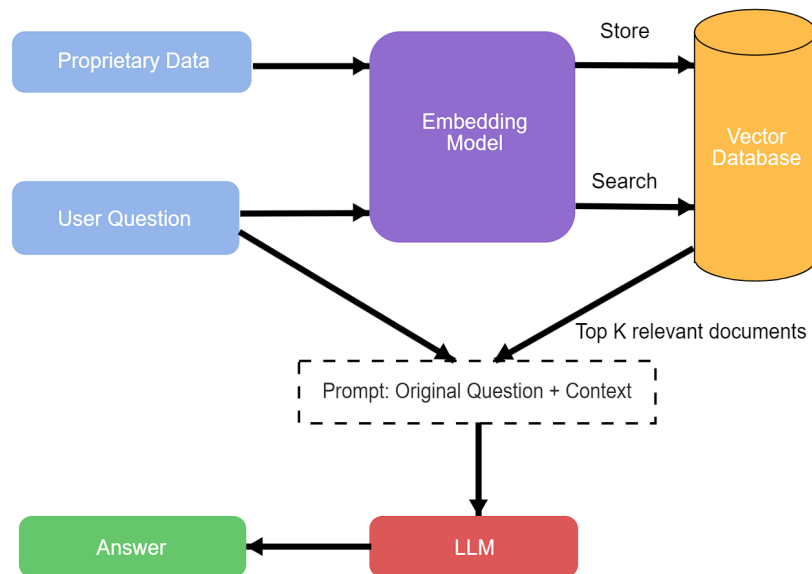


Figure 2.21: [RAG](#) System Architecture [\[83\]](#).

2.8.1 Retrieval mechanism

The retrieval system functions as a sophisticated information retrieval engine, tasked with efficiently accessing and extracting relevant information from a vast corpus or dataset. It operates as a repository of knowledge, storing diverse and extensive textual information that spans a wide array of topics and domains.

At its core, the retrieval mechanism consists of an embedding model, responsible for matching the user queries and the stored contexts in the embedding space, and a vector database responsible for retrieving the top-k contexts most similar to the question embedding.

Prompt condensation

Prompt condensation is a common initial step in the retrieval mechanism of a [RAG](#) system. When interacting with a conversational system and asking factual questions, users often make context dependent questions that rely on the previous conversation and are meaningless or ambiguous on their own. Using such ambiguous or context dependent questions, such as "Why?", in the retrieval mechanism would yield poor results. To address this, a condensation system is typically employed. This system provides the current user input along with the conversation history to the [LLM](#), which then generates a standalone, unambiguous question. An example of this process is shown in Listing 2.1.

1 User: Do product ingredients need to be displayed in a product label?

```

2   Assistant: Yes ...
3   User: Why?
4
5   Condensed prompt: Why do product ingredients need to be displayed in a
    ↪ product label?

```

Listing 2.1: Prompt condensastion example

Embedding models

The embedding models are critical components responsible for converting textual information into numerical representations, or embeddings, which capture semantic relationships and contextual information. RAG systems typically consist of a query encoder and a context encoder, each designed to process and encode textual data in a way that facilitates effective retrieval and generation.

Regarding training, both the query encoder and context encoder can be trained to generate similar embeddings when a query matches a context. One approach is to keep the context encoder's weights fixed after pre-training it on a vast dataset and adjust the query encoder to create similar embeddings for a given query. Another method involves simultaneous training of both query and context encoders, aiming to maximize the similarity between the generated embeddings, as shown in Figure 2.22 [82, 83].

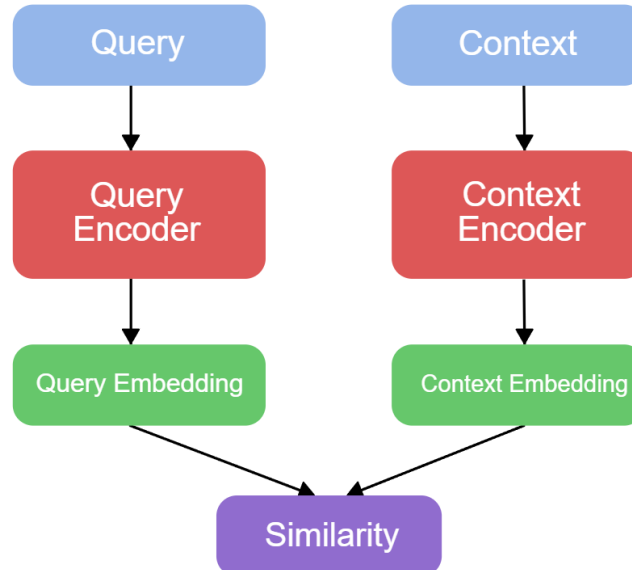


Figure 2.22: Retrieval Encoders Training Architecture.

Retrieval strategy

In the inference phase of a RAG system, minimizing latency is crucial given the real-time conversational nature of RAG. The system not only demands a fast Large Language Model (LLM), but also relies on a retrieval mechanism capable of rapidly sifting through a vast pool of millions or even billions of documents to pinpoint relevant contexts.

To accomplish this, contexts are typically stored within a vector database, a specialized system engineered for rapid searches within an embedding space. This database swiftly identifies the top k most similar candidates to a given query.

The operational principle of a vector database involves the storage of embeddings within a vector index. This index constitutes a structured storage format optimized for fast retrieval based on vector similarity. Diverse types of vector indices exist, each presenting trade-offs between speed and accuracy. Consequently, the selection of an appropriate index type is dependent upon the specific requirements of the task at hand [82, 83].

- **Flat Index:** Flat indexing stores vectors as they are, without modifications. It computes the similarity between the query vector and every other vector in the index, returning the K most similar vectors. Even though it is simple to implement and has perfect accuracy, it is inefficient and slow in large datasets given that it has to compute similarity across the entire space [84].
- **Locality Sensitive Hashing (LSH) Index:** LSH hashes similar vectors to the same bucket using hashing functions. It groups nearby vectors into buckets, allowing for approximate nearest neighbor searches by computing similarity within these buckets. It is appropriate for large datasets even though it sacrifices accuracy for speed, offering approximate matches instead of exact ones [84, 85].
- **Inverted File (IVF) Index:** IVF partitions the vector space into clusters, finding centroids for each cluster. It maps the query vector to the closest centroid and searches within the associated cluster. Even though it is better than LSH in large datasets, it has potential issues when the query vector lies near the edges of multiple clusters, requiring searches across multiple clusters [84, 86].
- **Hierarchical Navigable Small Worlds (HNSW) Index:** HNSW builds a multi-layered graph structure, grouping similar vectors together. Lower layers capture all vectors, and higher layers group data exponentially based on similarity, reducing the number of vectors per layer, as shown in Figure 2.23. It is highly efficient and fast, being effective for large datasets without significantly sacrificing accuracy [84, 87].

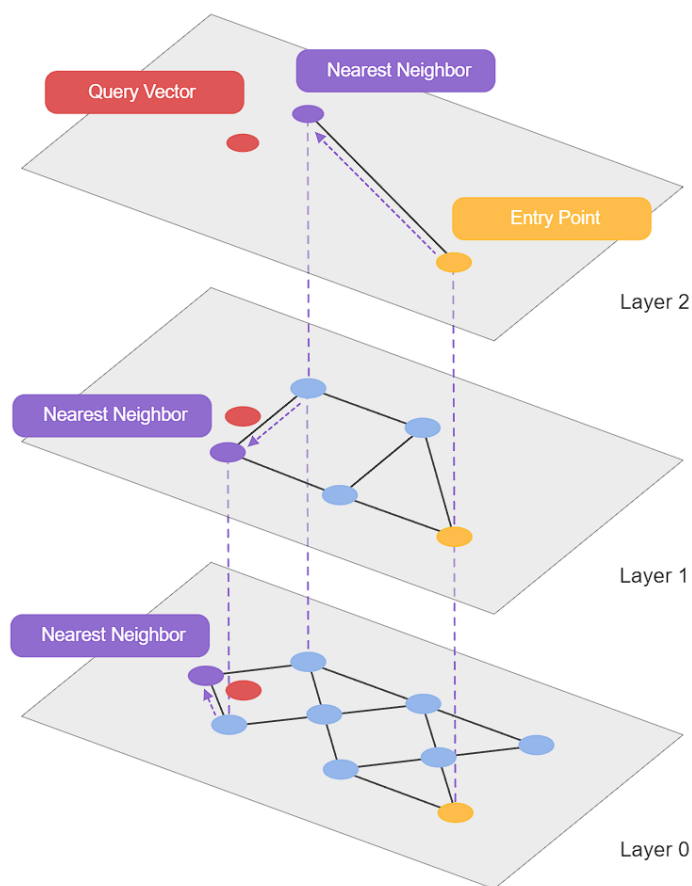


Figure 2.23: Example of a search on an HNSW graph [84].

Similarity metrics

The selection of appropriate similarity metrics holds paramount importance in retrieving relevant contexts for generating coherent and contextually precise responses. These metrics serve as the cornerstone for evaluating the resemblance or divergence between vectors, enabling the extraction of contextually rich information pivotal for refining the subsequent generation process.

- **Cosine similarity:** One of the most widely used measures, especially in text analysis. It calculates the cosine of the angle between two vectors, treating them as multidimensional vectors in an n -dimensional space. Cosine similarity measures the similarity in direction rather than magnitude, providing a value between -1 (completely dissimilar) and 1 (identical) [88].
- **Euclidean distance:** Measures the straight-line distance between two points in Euclidean space. It calculates the square root of the sum of squared differences between corresponding elements of the vectors. It is a measure of magnitude or length and is more often used in clustering algorithms [89].
- **Manhattan distance (city block distance):** Calculates the distance between two points by summing the absolute differences between their corresponding coordinates. It is akin to navigating through a city block, moving only along the grid [90].

- **Jaccard Similarity:** Primarily used for comparing the similarity and dissimilarity of finite sample sets. It measures the similarity between two sets by calculating the intersection divided by the union of the sets [91].

Reranker

A reranker is an optional component that reevaluates the context retrieved from the vector database. It serves as a filter, reordering and assessing the retrieved results from the initial retrieval based on specific criteria. This process aims to select the most suitable candidates that closely align with the context and requirements of the task at hand.

The need for a reranker arises due to limitations in simple retrieval mechanisms, which often retrieve a broad set of candidates related to the context but may lack fluency, coherence, or may contain outliers that don't relate to the context at all. This usually happens due to the compression of the context into a fixed size vector what can lead to collisions in embedding space. The reranker addresses these limitations by reevaluating and re-scoring retrieved candidates consuming the context and the query and outputting a new and improved similarity score [92, 93].

Even though rerankers are architecture agnostic, they are usually transformers, due to their previously mentioned advantages which allows the capture of nuanced contextual information and production of more coherent, contextually appropriate responses, enhancing the overall performance of RAG systems [92, 93].

Corrective RAG

Corrective Retrieval Augmented Generation (CRAG) is a variation of a RAG system that implements an additional layer of context filtering after the reranker. The main purpose of this mechanism is to further evaluate and eliminate contexts that are unrelated with the user query and introduce irrelevant information. This system works by passing each retrieved context to the LLM individually along with the question made by the user. The LLM is prompted to rate the context as correct, ambiguous or incorrect which is then used to discard incorrect contexts [94].

2.8.2 RAG evaluation

Evaluating a RAG pipeline necessitates dissecting and evaluating these components separately and together to comprehend the pipeline's performance nuances and areas for enhancement. Quantitative evaluations become imperative to track progress and performance.

Several metrics were already discussed to evaluate several individual components of the RAG pipeline, but when it comes to evaluating the system as a whole, evaluations based on simple comparison scores do not take in consideration the full meaning of the provided answers in such an abstract problem as RAG. Therefore, the use of LLMs to conduct evaluations autonomously presents itself as the best current solution for the evaluation problem [95, 96].

Component-Level Metrics

- **Context precision:** Evaluates the retrieval mechanism by measuring the signal-to-noise ratio of retrieved context [97].
- **Context recall:** Evaluates the retrieval mechanism by assessing retrieval completeness, relying on human-annotated ground truth labels [97].
- **Faithfulness:** Evaluates the generation mechanism by measuring factual accuracy by comparing generated answers with retrieved context [97].
- **Answer relevancy:** Evaluates the generation mechanism by measuring relevance of generated answers concerning the query [97].

End-to-End Metrics

- **Answer semantic similarity:** Evaluates how closely the generated answer resembles the ground truth by measuring semantic similarity between the two [97].
- **Answer correctness:** Measures how accurately the generated answer aligns with the ground truth, combining semantic and factual similarity scores [97].

2.8.3 RAG in food engineering

The recent advancements of dynamic and interactive systems that can easily bridge the gap between technology and other scientific fields, have emerged as a revolutionary way to empower professionals with fast and reliable access to knowledge. RAG plays a pivotal role in this paradigm shift. RAG models, by combining the strengths of information retrieval and natural language generation, can offer food engineers an unprecedented tool to navigate the vast landscape of relevant scientific literature efficiently.

Drawing inspiration from the success of LLMs in the medical domain, exemplified by initiatives like MedPALM [98] and LLMs such as PubMedBERT [99], the potential impact on food engineering becomes apparent. The transformative capabilities of domain-specific LLMs can elevate the efficiency of knowledge retrieval in food engineering, akin to its contributions in medical research.

These LLMs, when empowered with retrieval mechanisms and access to large knowledge bases, can surpass the capabilities of standalone LLMs, which tend to hallucinate and lack the necessary technical knowledge to accurately answer domain-specific questions. A notable example in the literature is the RAG-based model, PaperQA [100], designed for answering questions over scientific literature. By performing information retrieval across full-text scientific articles and utilizing RAG to provide answers, PaperQA showcases the potential for large-scale, systematic processing of scientific knowledge. Its ability to exceed the performance of existing LLMs on science QA benchmarks highlights the efficacy of the RAG paradigm. The recent advancements in domain-specific conversational systems can be further supported by the works shown in Table 2.2.

Name	Type	Domain
PaperQA [100]	RAG Agent	Science
MedPALM [98]	LLM	Medical
PubMedBERT [99]	LLM	Medical
SciBERT [101]	LLM	Science
BioGPT [102]	LLM	Biology
DARWIN [103]	LLM	Science
RetA [104]	RAG	Biomedical
Almanac [105]	RAG	Medical

Table 2.2: Domain-specific LLMs and RAG systems

Development

3.1 Technologies

All the source code developed in this work was developed in the Python (version 3.10) programming language using *Visual Studio Code* as an IDE. During development several helper packages were used to help build a centralized LLM testing, evaluation and deployment platform.

- **argilla**: used to host and collect human feedback for the LLMs;
- **langchain**: to provide a centralized framework to access LLMs, embeddings, vector stores, rerankers and agents on which the developed package structure sits on;
- **langchain_openai**: providing an easy connection to any OpenAI compatible API;
- **sentence_transformers**: used to locally and efficiently run embedding models and rerankers;
- **huggingface**: to easily download open source LLMs, embedding models and rerankers;
- **ragas**: used to perform automated evaluations of the RAG system;
- **chainlit**: to deploy conversational agents demos for easy testing and evaluation;
- **fitz**: a library used to open and extract raw texts from pdf documents;

Besides these main packages, other helper packages were also used like *numpy*; *pandas*; *bs4* and *requests*.

The machine to generate the vector databases and run the evaluation process was kindly provided by the University of Minho, giving access to a shared linux server with 4 nvidia Tesla T4 GPUs with 16 GB of RAM and 100 GB of storage. All OpenAI GPTs instances used during testing and evaluation were provided

by Omnium AI who supported the API costs. Omnium AI has also given access to azure virtual machines to deploy an application demo and host the human feedback platform as well as their Databricks platform to run notebooks, small experiments and store data in a relational database. All the results and intermediate files were stored in *JSON* format for easy reading and parsing.

3.2 GumGum package

The development of a sophisticated *RAG* system requires exhaustive experimentation of different types of *LLMs*, embeddings models, rerankers and agents. Understanding how different combinations of these components impact the system's overall performance is crucial. Consequently, a package was created to centralize all essential components of an *LLM*-based system, including evaluation and feedback mechanisms. This package empowers Omnium AI to continuously iterate and enhance their *LLM* agents as new models and methodologies with advanced capabilities become available.

The primary objective of this package is to integrate the functionalities of widely-used industry tools, enabling them to work together seamlessly and efficiently. The package employs tools like *langchain* and *faiss* for system building and access models, *ragas* for automatic system evaluation and *argilla* to deploy and host a human feedback platform.

3.2.1 Package Structure

The GumGum package is organized into eight main modules focused on *RAG*, as illustrated in Figure 3.1. Each module represents a crucial component of the agent pipeline. These modules enhance the functionality of existing libraries, enabling the use of industry-tested tools while also being easily extendable for custom and broader use cases, as shown by the green submodules.

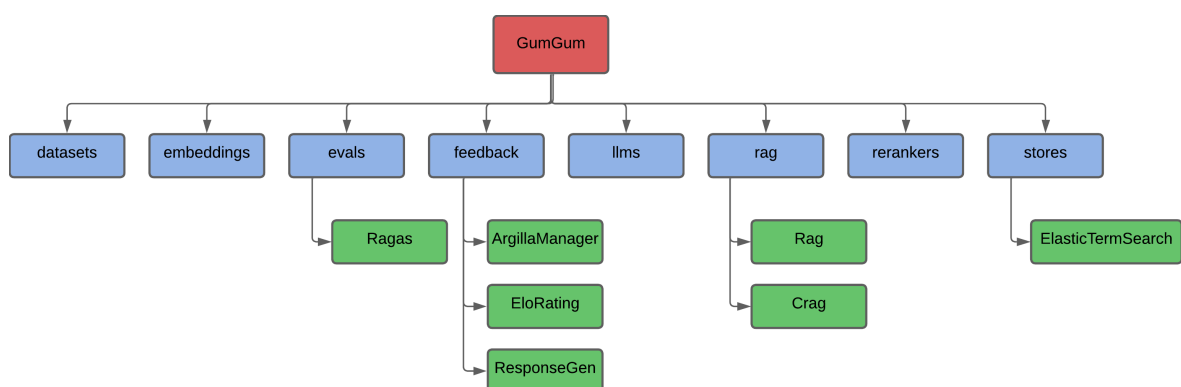


Figure 3.1: GumGum package module structure

- **datasets**: This module leverages the *datasets* package to transform and manipulate large datasets efficiently. It supports loading and saving substantial amounts of data in various file formats;
- **embeddings**: This module utilizes the *langchain* package to access embedding models, whether hosted locally or remotely;

- **evals:** This module employs the *ragas* framework to automatically evaluate any [RAG](#) based system using selected metrics;
- **feedback:** This module uses the *argilla* framework to host a dataset of [LLM](#) answered questions for human evaluators to rank. These rankings are then used to compute the ELO rating of each [LLM](#) based on human preferences;
- **llms:** This module also utilizes the *langchain* library to access over eighty different [LLM APIs](#) for prompt completion, whether they are hosted locally or remotely;
- **reranker:** This module uses the *langchain* library to access and extend eleven reranker [API](#) standards, available locally or remotely;
- **stores:** This module employs the *faiss* and *langchain* libraries to extend and implement retrieval stores. These stores can be based on vector search retrieval or other methods such as keyword or expression search in local or remote databases;
- **rag:** This module implements various types of [RAG](#) agents. These agents can incorporate any [LLM](#) model, embedding model, vector store, and optionally a reranker, to build a fully functional [RAG](#) agent following a predetermined execution formula.

Ragas

The Ragas submodule was developed to leverage the existing Ragas metrics, which can compute an evaluation score ranging from 0 to 1 for a given data sample. These metrics are the industry standard for model evaluation. The primary objective of this submodule is to enable Ragas to accept LangChain models and address some of its shortcomings, particularly in terms of parallelization. Ragas has a tendency to stall during evaluation without throwing any errors, so a custom evaluation loop was developed. This loop distributes the evaluation process among a specified number of workers, preventing stalls and accelerating the evaluation process.

Additionally, since [LLMs](#) sometimes return data incompatible with the required format (a string with a reason field and a verdict field), this incompatibility can generate a NaN score that propagates through the entire evaluation. To address this, all kinds of [LLM](#) format errors are logged, but they return a score of 0.0 to prevent corrupting the overall evaluation metric.

ArgillaManager

The argilla manager submodule is responsible for managing the remotely hosted argilla instance that stores and collects human feedback. Given a *url* and an [API](#) key, this submodule is responsible for implementing helper functions that upload and download the data or manage the users and their privileges.

- **upload:** This functionality allows to open a [JSON](#) dataset and construct the question and title fields for each model. To minimize biases, the order of the questions is randomized and at no point the

titles mention which [LLM](#) generated the answer, forcing and impartial evaluation. To remap the evaluations to the correct model on download, the questions and answers also have a model id field that stores the name of the model that answered the question.

- **download:** This function downloads the ranked dataset from the argilla platform storing the results in a [JSON](#) file. This file contains a dictionary where each question id maps to a dictionary containing each model name and the corresponding score. There are as many scores possibilities as models and different answers can get the same score in case they have similar quality;
- **create users:** A functionality that allows to create and upload a list of users to the platform. Each user is composed by a username, a first name, last name, password, role and workspace. After creation, the users are returned now containing an [API](#) key field that can be used to manage datasets;
- **delete users:** Also for user management, the delete function allows the admin to delete a list of users from the argilla platform.

EloRating

As a way to determine the best overall [LLM](#) from a set of comparative rankings, the Elo rating system was implemented as a submodule to generate Elo scores for each model given a ranking dataset. All models start with a predetermined Elo score of 1000 and for each question in the dataset with valid ranks, they are compared with every other model. Each comparison is evaluated as a win (1, 0), a loss (0, 1) or a tie (0.5, 0.5). With these results the new Elo scores are calculated using the formula in Equation 3.1 and stored in a [JSON](#) file for easy readability.

$$R' = R + K(S - E) \quad (3.1)$$

where:

- R is the current rating of the [LLM](#).
- R' is the new rating of the [LLM](#).
- K is a constant that determines the maximum possible change in rating. The default value of 10 was chosen.
- S is the actual score in the game (1 for a win, 0.5 for a draw, and 0 for a loss).
- E is the expected score, calculated using:

$$E = \frac{1}{1 + 10^{(R_{\text{opponent_llm}} - R)/400}} \quad (3.2)$$

where $R_{\text{opponent_llm}}$ is the rating of the opponent [LLM](#) and R is the current rating of the model.

ResponseGen

This submodule is responsible for generating answers for a dataset of questions using an [LLM](#), a system prompt, and the dataset of questions. Its functionality iterates over the questions sequentially, which is appropriate for handling small datasets and avoiding rate limiting of some models. The process begins by appending the system prompt to each question and then running it through the LangChain-compatible model. The generated answer content, along with the name of the model that produced it, is then appended to the list of answers for that question.

Once all questions have been processed, the submodule saves the new dataset as a copy of the original in a specified file. This approach allows for generating answers for the same dataset from multiple different models, facilitating later ranking and evaluation.

ElasticTermSearch

The ElasticTermSearch class complements some of the data stores already implemented in the LangChain framework. All of the stores in LangChain are vector stores, meaning they only perform vector similarity searches. However, it is sometimes useful to perform other kinds of searches, such as keyword or expression searches on different databases like Elastic Search.

This class implements the base VectorStore class from LangChain and its invoke method, which returns a list of LangChain Documents given a natural language question. It achieves this by using an [LLM](#) to generate keywords and expressions related to the user's question using the system prompt seen in Listing 3.1. These keywords are post-processed into a list of terms formatted into an [API](#) URL used in a GET request, where a top_k parameter is also specified. The returned texts are then transformed into LangChain Documents with contents and sources, which are subsequently returned.

```

1 Instructions:
2 You are an helpful assistant chatbot designed to help food engineers with
   ↳ their questions.
3 Given the following question, extract or write a list of keywords,
4 separated by commas, usefull to seach for food engineering related articles
   ↳ that are likely to answer the question.
5 Write only the keywords. Dont write explanations or justifications.
6 Most keywords must be related to food engineering.
7 Write the keywords and nothing else.
8 Write around 20 keywords, no more.
9 Be creative.
10
11 Question: {question}

```

12 Keywords:

Listing 3.1: ElasticTermSearch keyword generation prompt

RAG

The `RAG` class implements custom behavior to answer questions based on retrieved contexts with the help of an `LLM` and a vector store retriever. It is designed to act as an agent capable of maintaining a long conversation with the user and answering questions based on previous interactions. Therefore, it can receive a conversation history for increased contextual awareness.

Some user questions are contextual, meaning they are not standalone and depend on the previous conversation. This is not ideal for the retrieval mechanism, as the retriever has no state and requires standalone questions. To address this, there is an intermediate step called question condensation. This step takes the current question and the conversation history to generate a standalone question suitable for retrieval by passing it to the model with a special condensation prompt shown in Listing 3.2.

After the contexts have been retrieved, they are inserted along with the history and the question into the system prompt, shown in Listing 3.3, which is then provided to the model to generate the answer, as seen in Figure 3.2. It is important to note that this agent has a state that is rewritten each time it receives a new question. This design allows for easy use in `REST APIs` in the future. Consequently, the conversation history must be passed to the agent with each new question, which in turn returns the answer and updates the history variable with the new answer. This requires the client to store the conversation history themselves. This non-permanent state can also be accessed after the answer is given to retrieve the contexts and sources used to answer the question, providing justification for the answer and referencing the source material.

As `LLM` take a considerable amount of time to generate a complete answer, a streaming functionality was also implemented. This functionality calls the model in streaming mode and return a Python generator that yields each new token allowing for a more user friendly experience.

```

1 Given the following conversation and a follow up question, rephrase the follow
  ↳ up question to be a standalone question.
2 If there is no chat history or the question is good as it is just copy and
  ↳ paste it verbatim.
3 Write only one line.
4
5 Chat History:
6 {history}
7 Follow Up Input: {question}
8 Standalone question:

```

Listing 3.2: RAG Agent condensation prompt

```
1 Instructions:
2 You are an helpful assistant chatbot designed to help food engineers with
   ↳ their questions.
3 Answer the question based only on the contexts above and the chat history.
4 If the answer is not present in the contexts say you dont know the answer.
5 Its very important that you always write article references next to each fact
   ↳ like so (refs[name_of_ref_1, name_of_ref_2, ...]).
6 Only write references of contexts with relevant information. dont use the ones
   ↳ that are off topic.
7 Only write references present in the contexts and use the reference names
   ↳ verbatim as in the contexts.
8 Dont give examples.
9 If the question is not related to this just say what is your purpose and dont
   ↳ answer.
10 Respect the user instructions.
11 Write a complete and informative answer, not too short and not too long. One
   ↳ paragraph is good.
12 Dont state your purpose if you dont know the answer, unless a user asks you.
13
14 ###
15 Contexts:
16 {contexts}
17
18 ###
19 Chat History:
20 {history}
21
22 ###
23 Question: {question}
24 Assistant:
```

Listing 3.3: RAG Agent system prompt

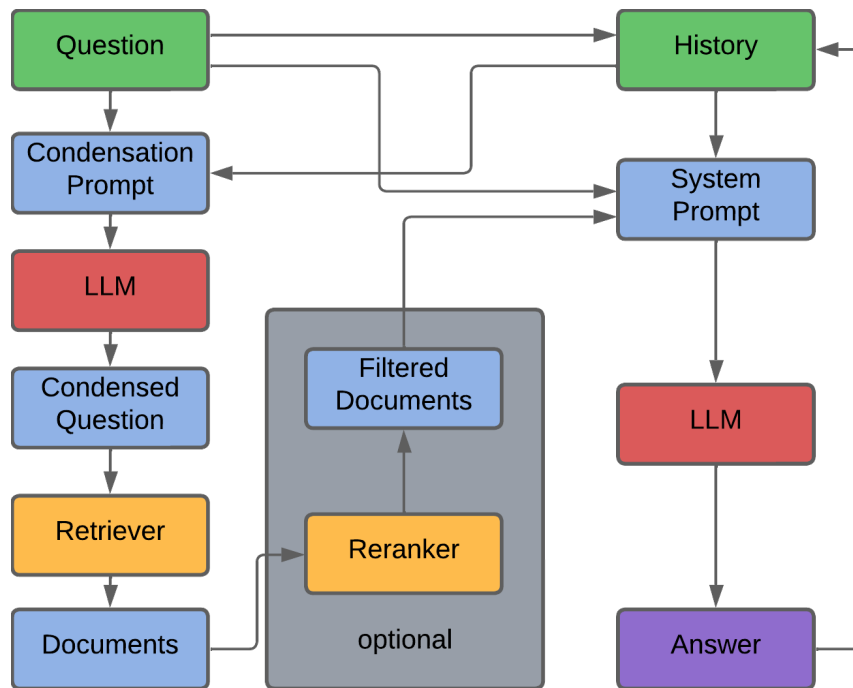


Figure 3.2: RAG Agent behaviour

CRAG

Just like the [RAG](#) class, the [CRAG](#) class implements custom behavior to answer questions based on retrieved contexts using an [LLM](#) and a vector store retriever. The "corrective" aspect of [CRAG](#) comes into play after the document retrieval phase. In this phase, the retrieved documents undergo an additional filtering stage, in this case conducted by the [LLM](#) itself.

This process is facilitated by a prompt, as shown in Listing 3.4, that takes both the question and a context. The model is then prompted to evaluate the context in relation to the question, categorizing it as "related", "ambiguous", or "unrelated". These evaluations are used to take appropriate actions such as eliminating the irrelevant documents, leaving only those that closely relate to the question and can potentially answer it.

This architecture, as shown in Figure 3.3, does however require an additional call to the [LLM](#), which have to be performed sequentially, therefore increasing both the computational costs and response wait times.

```

1 Given the following question and paragraph,
2 evaluate if the paragraph is related to the question as "related", "ambiguous"
  ↳ or "unrelated".
3 write the score word and nothing else.
4
5 ###
6 Paragraph

```

```

7 {context}
8
9 ###
10 Question
11 {question}
12
13 Evaluation:

```

Listing 3.4: CRAG Agent corrective prompt

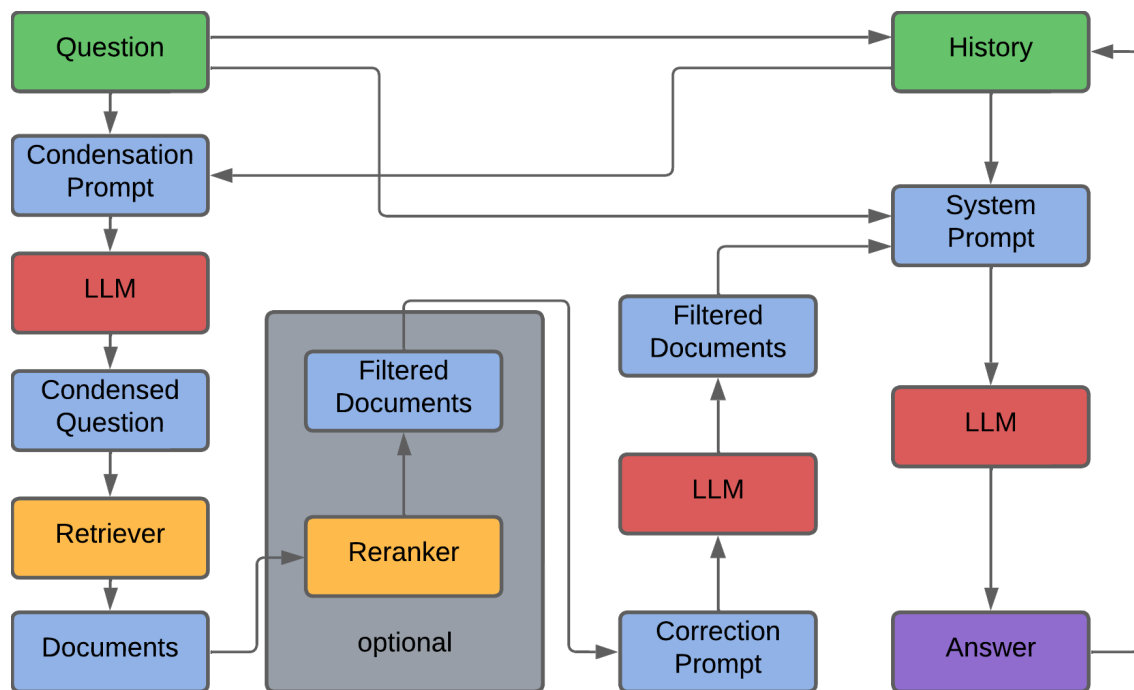


Figure 3.3: CRAG Agent behaviour

Experimental validation

The primary objective of this work is to develop an effective conversational agent capable of accurately responding to questions. To achieve this, it is essential to test and evaluate the performance of different agent configurations, utilizing various [LLMs](#), embedding models, rerankers, and agent types. The experiments that were developed try to explore the space of possible [RAG](#) systems and try to identify the best performing architecture in food engineering related fields such as biochemistry and biology.

4.1 Benchmarking datasets

In [RAG](#) systems, datasets serve as the knowledge foundation, enabling the model to retrieve relevant information to generate accurate and contextually appropriate responses. A significant challenge in this domain is the absence of publicly available RAG datasets specifically tailored to food engineering. Creating a specialized dataset for this field would require extensive resources, making it either prohibitively expensive or limited in scope and utility. Therefore, to address the need for a dataset to evaluate the best [LLM](#) for food engineering, and given the lack of public datasets, the "LLM Eval" dataset was developed at Omnium AI. This dataset is specifically designed to assess and compare models based on human feedback within the food engineering context.

To mitigate the broader issue of dataset scarcity when evaluating the system, an alternative approach involves leveraging existing datasets that exhibit similar technical language and complexity to food engineering. One such dataset is "PubMedQA", which encompasses a wide range of biomedical literature, including biochemistry and clinical trials. This dataset is suitable for the automatic evaluation of a [RAG](#) system, given the overlap in technical terminology and the structured format of questions and answers.

In addition, specific corpora relevant to food engineering laws, such as documents from the [CFR](#), were gathered. These sources provide authoritative information on regulations and guidelines pertinent to food engineering. However, since they do not come with pre-formed questions, answers, and relevant passages,

they are more suited for use as a knowledge base in the demo application rather than for evaluation or experimental purposes.

4.1.1 LLM Eval

To address the need for a dataset of questions tailored to compare and rank models based on human feedback, and considering the lack of publicly available datasets specific to food engineering, the "LLM Eval" dataset was developed with the help of Omnium AI. This dataset includes 18 questions spanning various categories, as illustrated in Figure 4.2, designed to reflect inquiries typically posed by food engineers using the Cibus platform.

The LLM Eval dataset is intended to be iteratively refined and expanded, supporting the continuous development and assessment of new LLMs. A key feature of this dataset is its alignment with real-world data. This alignment is evident in the diversity of question lengths, depicted in Figure 4.1, as well as in the presence of punctuation and grammatical errors, as detailed in Table 4.1, reflecting real world conditions that the model will be subjected to.

The dataset fields cover the following aspects:

- **id**: an integer field representing a unique identifier of each question;
- **author**: a string field with the name of the author of the question;
- **category**: a string field indicating the category the question belongs to;
- **question**: a string field containing the question;
- **answers**: a placeholder list of answers given by each model. Each one of these contain a "model" field with the model name and a "content" field with the generated answer. This field starts of empty and is later automatically filled by the generated answers from a selection of models;

Metric	Value
Number of questions	18
Min question length	3
Avg question length	6
Max question length	10
Questions that end with "?"	3
Questions with syntactical errors	8
Vocabulary size	49

Table 4.1: Analysis of LLM Eval feedback dataset

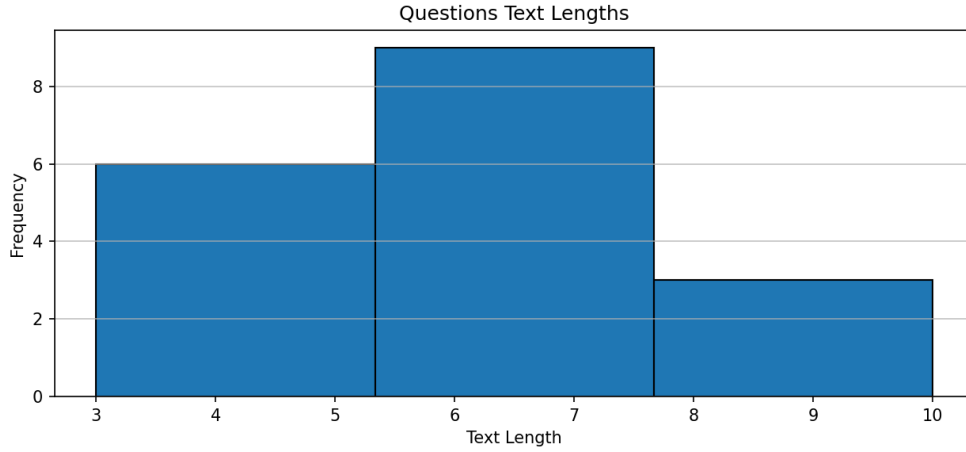


Figure 4.1: Histogram of question lengths (Number of words)

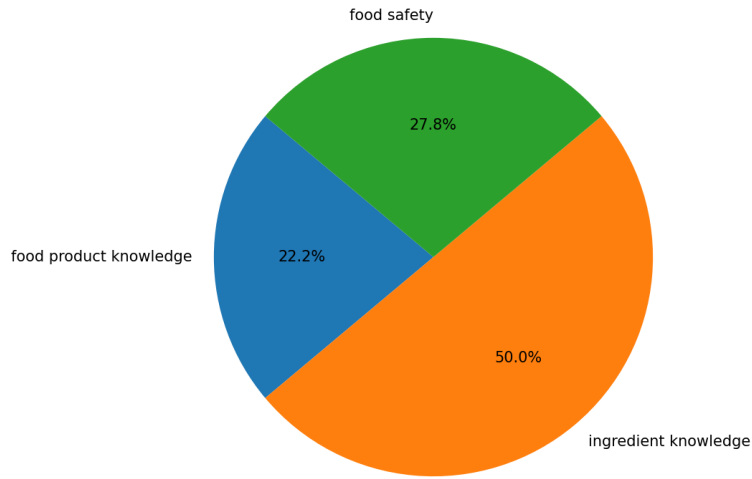


Figure 4.2: Distribution of question categories

4.1.2 PubMedQA

The "PubMedQA" dataset, available on the HuggingFace platform, can be easily accessed and downloaded and encompasses a wide range of biomedical literature, including biochemistry and clinical trials. It comprises three subsets with varying levels of labeling. Among these, the "pqa_artificial" subset has been selected due to its size and apparent data consistency. This subset contains 211,000 entries, each with the following fields:

- **pubid**: an integer fields containing the pubmed paper id from which the contexts were scrapped from;
- **question**: a string filed containing a question about the subject discussed in the paper;
- **context**: a string encoded [JSON](#) with a list of relevant passages to answer the question as well as topic related labels;

- **long_answer**: a string with the answer to the question in long form (about a paragraph);
- **final_decision**: a string with the answer to the question in "yes", "maybe" or "no" format;

Running experiments with 211,000 entries would be prohibitively expensive, as such only 5,000 were randomly selected for evaluation. These entries were then analysed by removing all the punctuation from the questions and contexts and counting the number of words producing the results showed in Table 4.2, Figure 4.3 and Figure 4.4.

Metric	Value
Number of questions	5000
Number of contexts	15440
Min question length	4
Avg question length	15.3326
Max question length	43
Min context length	1
Avg context length	64.6176
Max context length	352
Vocabulary size	36546

Table 4.2: Analysis of PubMedQA evaluation dataset

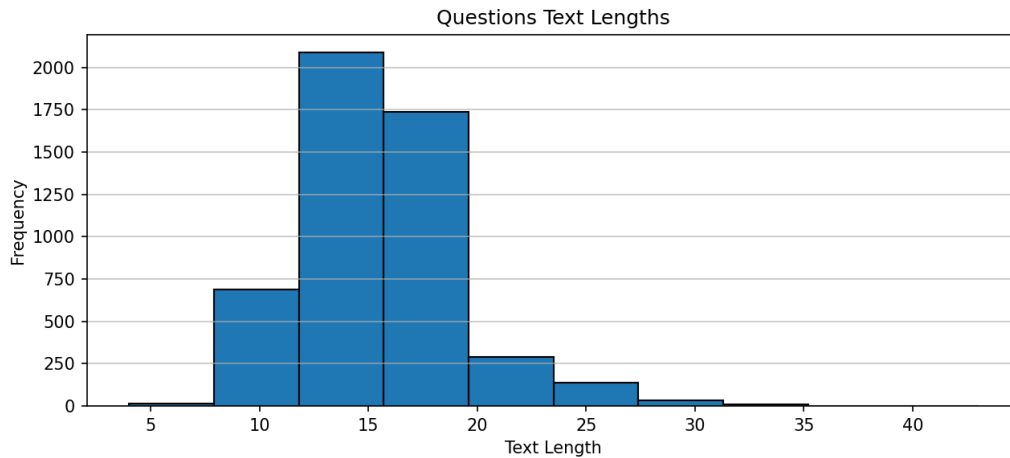


Figure 4.3: Histogram of question lengths (Number of words)

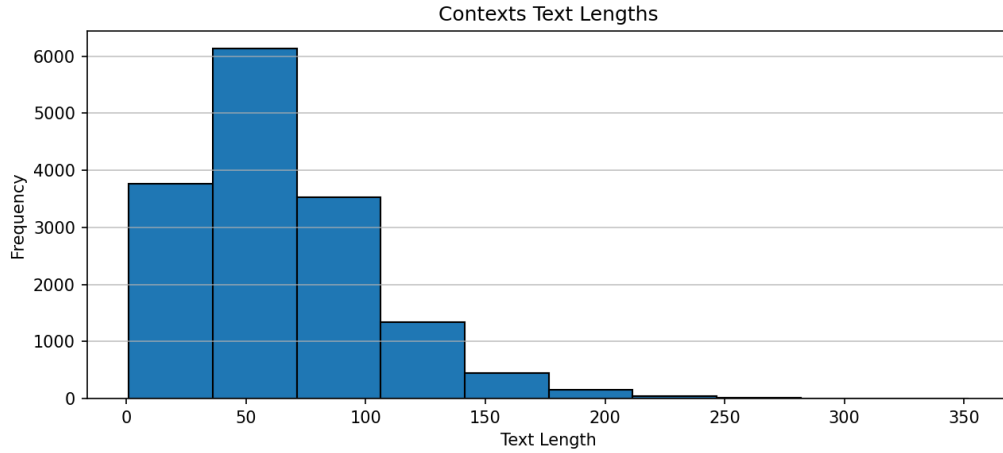


Figure 4.4: Histogram of context lengths (Number of words)

4.1.3 CFR

The [Code of Federal Regulations \(CFR\)](#) Title 21 - Food and Drugs - Subchapter B - Food for Human Consumption dataset comprises a specialized segment of the [CFR](#) dedicated to regulations and guidelines related to food engineering. This dataset includes detailed guidelines and requirements set by the U.S. [Food and Drugs Administration \(FDA\)](#), covering various aspects such as food labeling, food additives, good manufacturing practices, and standards of identity for different food products. By providing comprehensive and authoritative information, this dataset serves as a valuable resource for ensuring compliance with federal regulations and for supporting research and development in the food engineering field. The dataset was downloaded in markdown format from directly from the [CFR](#) website, and contains the characteristics that can be seen in Table 4.3 and Figure 4.5

Metric	Value
Number of words	655618
Number of sections	2123
Min section length	3
Avg section length	309.8162
Max section length	18157
Vocabulary size	33477

Table 4.3: Analysis of [CFR](#) knowledge dataset

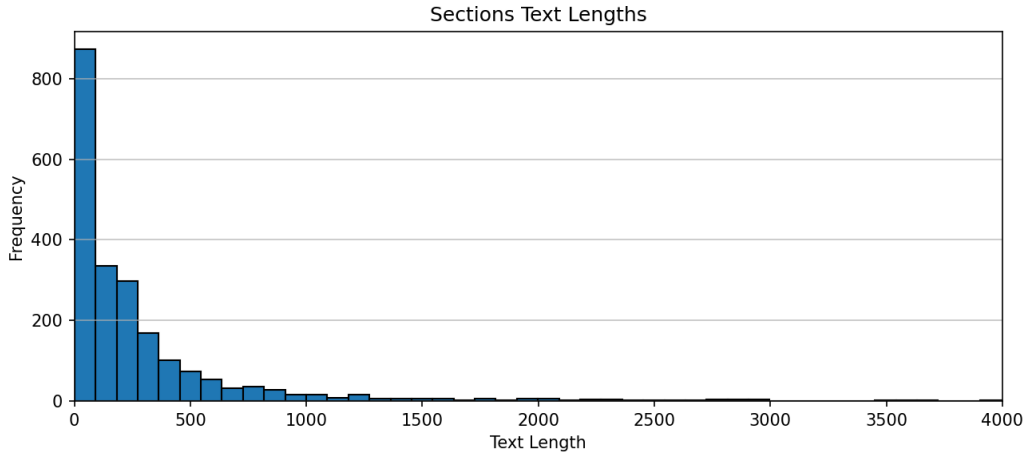


Figure 4.5: Histogram of section lengths (Number of words)

4.2 Data preprocessing

Effective preprocessing is essential for ensuring that data is in the appropriate format for evaluation and integration into the system, which in turn facilitates accurate model assessment and improvement. By transforming and structuring the raw data, the quality and usability of the data is enhanced, allowing for more efficient processing and more reliable outcomes in the analysis.

4.2.1 LLM Eval

Since the LLM Eval dataset was specifically developed at Omnium AI, it was already tailored to meet the requirements of the human feedback platform. The only preprocessing step needed was to convert the dataset from an Excel format to a [JSON](#) format. This conversion facilitated easier manipulation and integration within the system, ensuring compatibility with the existing data handling protocols.

4.2.2 PubMedQA

The PubMedQA dataset was largely in the correct format for evaluation purposes, requiring minimal preprocessing. The primary task involved processing the "context" field, which contained a [JSON](#)-encoded string with a list of contexts and labels. The preprocessing step involved extracting this [JSON](#) string and converting it into a simple list of contexts. This transformation was necessary to streamline the evaluation process and ensure the data was easily accessible for model input.

4.2.3 CFR

The [CFR](#) dataset required more extensive preprocessing due to its original format as a continuous text document. To store this dataset in a vector database, it was necessary to divide the text into manageable chunks. An algorithm was developed to achieve this segmentation, prioritizing the preservation of positional and contextual information.

The algorithm operated in several stages:

- **Markdown Sections:** Initially, the text was divided by markdown sections, ensuring that logical divisions within the text were maintained;
- **Line Divisions:** If markdown sections were too large, the text was further divided by lines;
- **Word Divisions:** As a final step, if necessary, the text was segmented by words to achieve the desired chunk length, with each chunk having a maximum length of 512 characters and a minimum length of 128 characters.

This multi-step approach ensured that the textual integrity and contextual relevance were preserved as much as possible, enabling efficient storage and retrieval within the vector database.

4.3 Cost Estimation

Accurately estimating the costs involved in developing an effective conversational agent is crucial for planning and resource allocation. Blindly running experiments and automatic evaluation systems without a general idea of the cost would lead to unsustainable expenses and incomplete experiments. The ideal experimental scenario would allow to test all possible combinations of components. However, testing all possible combinations is financially impractical. Specifically, evaluating an extensive dataset of 200,000 questions with an average input prompt length of 2,000 tokens and an average output of 200 tokens would cost approximately \$489,510.00 if both [GPT-4 Turbo](#) and [GPT-3 Turbo](#) were tested.

To address this challenge, the costs of two testing scenarios were simulated, as seen in Table 4.4, by varying the number of test questions (200,000 and 5,000) and by evaluating either all possible testing combinations or only a subset.

Exhaustive Evaluation

In this idealized scenario, all possible combinations of [LLMs](#) (both [GPT-3 Turbo](#) and [GPT-4 Turbo](#)), embedding models, rerankers, and agent types are tested. This means that each [LLM](#) would have to answer to all questions and all evaluation prompts for each possible combination of embedding models, rerankers and agent types.

The total cost **C** of such an exhaustive evaluation can be calculated using the following formula:

$$C = N_{questions} \times N_{combinations} \times N_{metrics} \times (N_{input_tokens} \times P_{input_token} + N_{output_tokens} \times P_{output_token}) \quad (4.1)$$

Where:

- $N_{questions}$ is the number of questions in the evaluation dataset.

- $N_{combinations}$ is the total number of possible combinations of embedding models, rerankers, and agent types.
- $N_{metrics}$ is the number of metrics being evaluated.
- N_{input_tokens} is the average number of tokens in the input prompt.
- P_{input_token} is the price per input token for the LLM being used.
- N_{output_tokens} is the average number of tokens in the output response.
- P_{output_token} is the price per output token for the LLM being used.

Applying this formula to the full PubMedQA dataset (section 4.1.2) and all combinations of LLMs and embedding models produces the results seen in exhaustive scenario of Table 4.4. The costs of this approach are unmanageable and increase exponentially with each new variable that is tested, making it unrealistic for comprehensive testing due to the high expenses involved.

Staged Evaluation (Chosen)

This scenario involves a staged evaluation approach, as illustrated in Figure 4.6. This approach tries to identify which parts of the system can be evaluated independently of the other components without affecting the end result. One such component is the evaluation process of the embedding models. Since the RAGAS framework evaluates the output of retrieval independently of the rest of the system, there is no need to repeat the process more than one time for each embedding model. This improvement alone decreases the evaluation cost by n times, being n the number of candidate embedding models. Although significant, this does not yet reduce the evaluation cost to an acceptable value. To bring the price further down, a compromise had to be taken, the LLM had to be chosen independently. This way the evaluation of the various context filtering levels (rerankers and agent types) is performed with only one LLM, which significantly decrease the number of API calls to the models at the cost of possibly falling short of the best possible combination.

The staged evaluation is therefore divided into three stages to streamline the evaluation:

- **Choosing the LLM:** Utilizes the LLM Eval dataset and the Argilla platform to have humans rank the responses from different LLMs. These rankings are then used to compute the Elo scores for each model, selecting the best-performing one.
- **Evaluating Embedding Models:** Assess the embedding models using the PubMedQA dataset (4.1.2), focusing on three key RAGAS metrics: context precision, context recall, and context relevancy. The best embedding model is then selected based on these evaluations.

- **Selecting Filters:** With the chosen LLM and embedding model, the reranker and agent type combinations are evaluated by running the entire system end-to-end. Appropriate RAGAS metrics (faithfulness and answer relevancy) are used to determine the optimal combinations.

This staged approach, is expected to significantly reduce evaluation costs and aid in identifying very effective configurations for the conversational agent.

Embedding Model	GPT Version	Number of Questions	Price (\$)	Scenario
One	GPT-3.5 Turbo	5k	316.5	Staged
One	GPT-3.5 Turbo and GPT-4 Turbo	5k	6,646.5	-
One	GPT-3.5 Turbo	200k	12,660.0	-
One	GPT-3.5 Turbo and GPT-4 Turbo	200k	265,860.0	-
All	GPT-3.5 Turbo	5k	582.75	-
All	GPT-3.5 Turbo and GPT-4 Turbo	5k	12,236.75	-
All	GPT-3.5 Turbo	200k	23,310.0	-
All	GPT-3.5 Turbo and GPT-4 Turbo	200k	489,510.0	Exhaustive

Table 4.4: Price comparison for different combinations of embedding models, GPT versions, and dataset sizes.

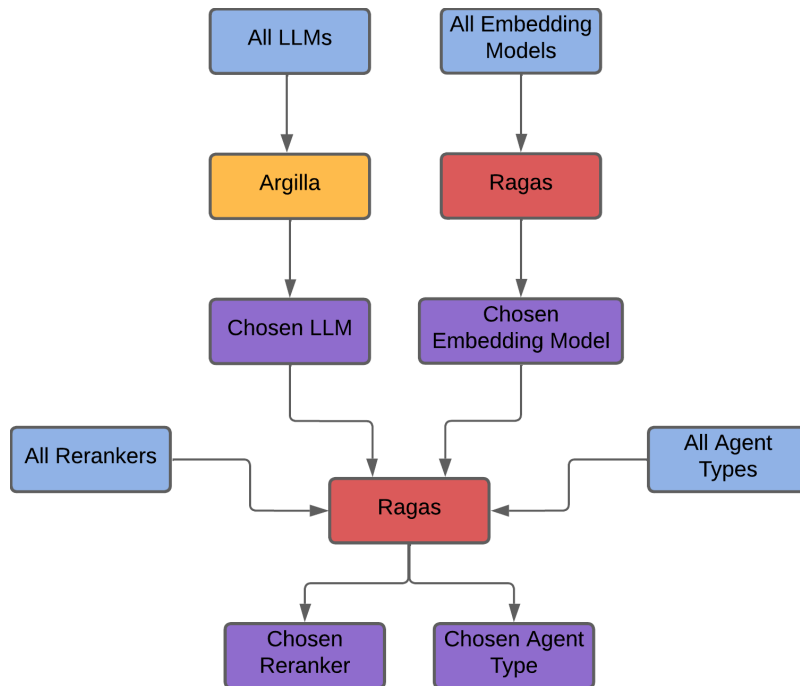


Figure 4.6: Staged Evaluation process

4.4 Evaluation Method

As described in the previous section, the staged evaluation method is far more cost effective than an exhaustive search. As such, it was chosen as the method to perform a system wide evaluation and understand what is the best component combination to create a retrieval augmented conversational system.

The process is divided in three stages: LLM evaluation; embedding model evaluation and filters evaluation. Each stage produces a JSON file with the evaluation results associated with each component and also persistently stores intermediate results that are used as a safety net in case the servers go down or the process fails unexpectedly.

4.4.1 LLM Evaluation

The LLM evaluation stage is responsible for determining which of the candidate models better aligns with human preferences when answering food engineering related questions and it can be subdivided into three fundamental phases.

- **Response generation:** This first phase of the process makes use the ResponseGen module (section 3.2.1) from the GumGum package to generate answers from each candidate model to the questions in the LLM Eval dataset.
- **Response ranking:** The generated answers and corresponding questions are then uploaded to the argilla platform hosted on azure with the help of the ArgillaManager module (section 3.2.1) where they can be reviewed and comparatively ranked as seen in Figure 4.7.
- **Elo scores calculation:** Once all answers have been comparatively ranked they are downloaded with the ArgillaManager module and the EloRating module (section 3.2.1) is used to generate the final Elo scores for each model. The highest scoring one is then chosen to be used in the latter stages of evaluation.

All the candidate models for this process, seen in Table 4.5 were chosen either because they are open source projects and/or because of their performance in the OpenLM Chatbot Arena Leaderboard [106].

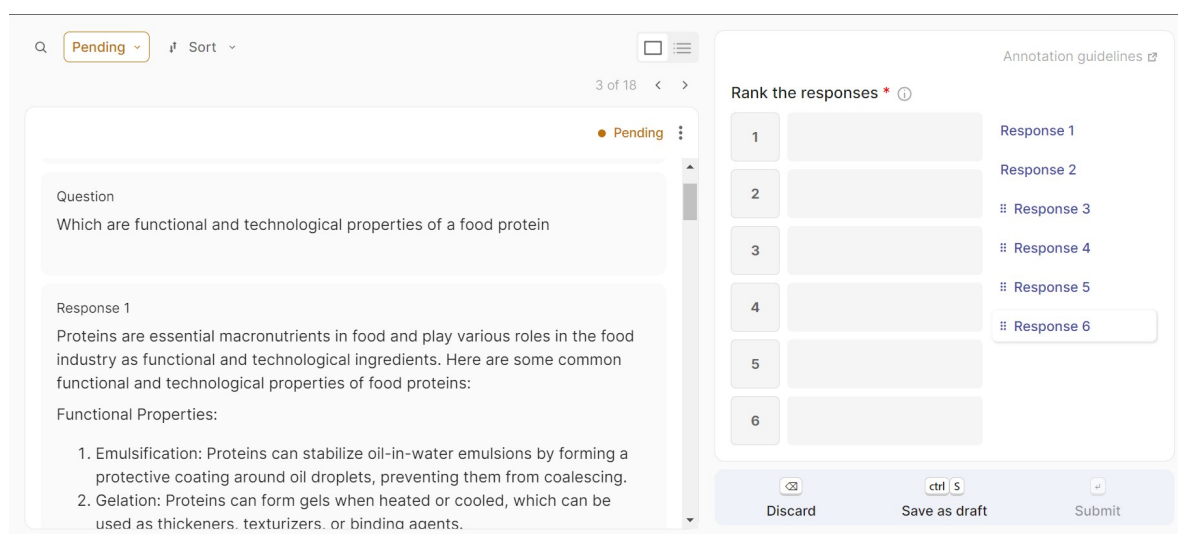


Figure 4.7: Argilla human feedback platform

Model	Arena Elo	Organization	License
GPT-4-1106-preview	1251	OpenAI	Proprietary
GPT-3.5-Turbo-0613	1117	OpenAI	Proprietary
Mistral-7B-Instruct-v0.2	1072	Mistral	Apache-2.0
zephyr-7b-beta	1053	HuggingFace	MIT
openchat-3.5-0106	1091	OpenChat	Apache-2.0
gemma-7b-it	1037	Google	Gemma license

Table 4.5: OpenLM Chatbot Arena Leaderboard [106]

4.4.2 Embedding Model Evaluation

This stage aims to select the best embedding model from a pool of candidates, using the PubMedQA dataset for evaluation. The models, seen in Table 4.6, were chosen from the HuggingFace platform for easy integration and availability of tools and were chosen by their overall performance and popularity among the community. The evaluation process involves two sequential steps for each candidate model: embedding generation and model evaluation, both described in section 4.4.2.

Model	Parameter Count	Sequence Length	Embed Size
jinaai/jina-embeddings-v2-base-en	137M	8k	768
intfloat/multilingual-e5-large	560M	512	1024
BAAI/bge-large-en-v1.5	335M	512	1024

Table 4.6: Candidate embedding models

Embedding generation

In the first step, embeddings are generated. This requires downloading and instantiating the embedding model. This instantiation may necessitate a query instruction and/or an embed instruction, which are used to prepend to the model prompt when generating embeddings for the query or the context, respectively. Subsequently, the vector store is generated using the FAISS library, which was integrated in the GumGum package. This vector store transforms the contexts into embeddings using the embedding model. Afterward, a retriever is created with the vector store, responsible for receiving a text query, embedding it, and performing a vector search to find the top k contexts most likely to answer the question. The top k parameter is set to 10, as a higher value typically improves retrieval performance, but it is essential to consider that increasing top k also raises the input token count per prompt, thereby increasing the API inference cost. The search index used by the retriever is a flat cosine similarity search, since the amount of data is considerably small and does not cause a noticeable search delay and does not sacrifice any accuracy during search.

Once the retriever is created, the dataset is iterated, and the retriever is called with each question, generating a list of top k contexts, which is added to the dataset. This new dataset, now with the retrieved contexts, is persistently stored in a JSON file in case the process fails. This approach allows the evaluation process to recover from a failure without the need to regenerate all the embeddings.

Model evaluation

The second step involves evaluating the retrieved contexts using the RAGAS framework, integrated into the GumGum package. This is an automatic evaluation process, performed on selected metrics by an LLM. By default, RAGAS use GPT-4 Turbo to perform the evaluation, as it is the most accurate model to date. However, preliminary tests with GPT-4 Turbo as the evaluator have shown that the model frequently responds in an incorrect output format, which is JSON. Coupled with the cost estimation study detailed in section 4.3, this justifies the use of the GPT-3.5 Turbo model, which is the second best and more cost-effective.

Regarding the evaluation metrics, three were chosen to assess the quality of the retrieved content:

- **Context precision:** This metric evaluates whether all ground-truth relevant items present in the contexts are ranked higher. Ideally, all relevant chunks must appear at the top ranks. This metric returns a result from 0 to 1 and is computed in Equation 4.2.

$$\text{Context Precision@K} = \frac{\sum_{k=1}^K (\text{Precision@k} \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}} \quad (4.2)$$

$$\text{Precision@k} = \frac{\text{true positives@k}}{(\text{true positives@k} + \text{false positives@k})} \quad (4.3)$$

where K is the total number of chunks in contexts and $v_k \in \{0, 1\}$ is the relevance indicator at rank k . In this metric the LLM is responsible to evaluate a given context as useful or not useful to answer a question with the prompt in Listing 4.1.

```
1 Given question, answer and context verify if the context was useful in
   ↳ arriving at the given answer. Give verdict as "1" if useful
   ↳ and "0" if not with json output.
```

Listing 4.1: Context precision metric prompt

- **Context recall:** This metric measures the extent to which the retrieved context aligns with the annotated answer. It returns a value between 0 and 1 as computed in Equation 4.4.

$$\text{Context Recall} = \frac{|\text{Ground truth sentences that can be attributed to context}|}{|\text{Number of sentences in ground truth}|} \quad (4.4)$$

In this metric, the role of the LLM is to identify and score each sentence in an answer as being or not part of a context. The utilized prompt is shown in Listing 4.2.

```
1 Given a context, and an answer, analyze each sentence in the answer
   ↳ and classify if the sentence can be attributed to the given
   ↳ context or not. Use only "Yes" (1) or "No" (0) as a binary
   ↳ classification. Output json with reason.
```

Listing 4.2: Context recall metric prompt

- **Context relevancy:** This metric measures the relevancy of the retrieved context, calculated based on both the question and contexts. The values range from 0 to 1, with higher values indicating better relevancy. It is computed as shown Equation 4.5.

$$\text{Context Relevancy} = \frac{|\text{Number of relevant sentences in context for answering question}|}{|\text{Total number of sentences in retrieved context}|} \quad (4.5)$$

The role of the LLM in this metric is to extract relevant sentences from a context that can answer a question, receiving as input the prompt shown in Listing 4.3.

```

1  Please extract relevant sentences from the provided context that are
    ↳ absolutely required to answer the following question. If no
    ↳ relevant sentences are found, or if you believe the question
    ↳ cannot be answered from the given context, return the phrase "
    ↳ Insufficient Information". While extracting candidate sentences
    ↳ you're not allowed to make any changes to sentences from given
    ↳ context.

```

Listing 4.3: Context relevancy metric prompt

The dataset is evaluated by GPT-3.5 Turbo across all three metrics, with the workload divided among 16 workers to maximize available compute resources and minimize evaluation time. The evaluation results for each embedding model are stored in a JSON file, containing the final value of each metric.

4.4.3 Filters Evaluation

Once the LLM and embedding model are selected from the pool of candidate models, it is time to evaluate the performance and the need for different kinds of filtering mechanisms. The filtering mechanisms chosen are rerankers and a corrective stage by the LLM that is added in the encapsulating system agent behaviour. Since this layer operates on top of the retrieval mechanism, it is dependent on it. So the retrieval system used during this evaluation is the best one selected in the previous stage. This guarantees consistency and similar performance can be expected in the final system. Given that these filters are optional, it is important to evaluate not just the performance of each one but also the performance of the system without them. The candidate rerankers were selected from the HuggingFace platform for their benchmark claims and popularity in the community and can be seen in Table 4.7. The corrective system is applied using the LLM selected in the LLM evaluation stage (section 4.4.1). This evaluation process does also need to be subdivided in two sequential stages for each candidate reranker-agent combination: answer generation and combination evaluation (section 4.4.3).

Model	Parameter Count
BAAI/bge-reranker-v2-m3	568M
intfloat/simlm-msmarco-reranker	~ 100M
None	0

Table 4.7: Candidate reranker models

Answer generation

To evaluate the quality of the generated answers, it is essential to first generate these answers. This process begins with downloading the necessary dataset and instantiating the chosen [LLM](#) and vector store retriever using the selected embedding model. Following this setup, all combinations of reranker and agent type pairs are iterated to generate the answers. The process starts by instantiating the reranker being evaluated, leveraging LangChain classes such as `HuggingFaceCrossEncoder`. The `top_k` parameter for each reranker is set to 5 to ensure a manageable number of top contexts are considered. Depending on the agent type under evaluation—whether [RAG](#) or [CRAG](#)—the respective classes are instantiated using the appropriate retriever template. For the [RAG](#) agent, the instantiation involves creating a retriever that interacts with the [LLM](#) to provide the most relevant contexts for each query. The [CRAG](#) agent, on the other hand, combines the retriever with a reranking mechanism to further refine the selection of contexts before generating the final answer. This refinement process aims to improve the relevance and accuracy of the answers by ensuring that the most pertinent information is considered. Once the system variant is created, the dataset is iterated, and the agent answers each question one by one. To safeguard against data loss and to avoid the need to regenerate answers in case of process failure, all answers and retrieved contexts are persistently stored in a [JSON](#) file.

Combination evaluation

The second step utilizes the RAGAS framework to quantitatively evaluate the quality of the final answers generated by the system. This evaluation is performed automatically with the RAGAS module (section 3.2.1) integrated in the GumGum package. For reasons similar to those discussed in section 4.4.2, [GPT-3.5 Turbo](#) is chosen as the [LLM](#) for evaluating the answers. This evaluation employs two RAGAS metrics specifically designed to assess the quality of the final answers:

- **Faithfulness:** This metric measures whether all claims made in the generated answer can be inferred from the given context. To calculate this, a set of claims from the generated answer is first identified using the prompt listed in Listing 4.4. Each of these claims is then cross-checked with the given context to determine if it can be inferred, using the prompt listed in Listing 4.5. The faithfulness score ranges from 0 to 1 and is computed as shown in Equation 4.6.

$$\text{Faithfulness Score} = \frac{|\text{Claims in the generated answer that can be inferred from given context}|}{|\text{Total number of claims in the generated answer}|} \quad (4.6)$$

1 Create one or more statements from each sentence in the given answer.

Listing 4.4: Faithfulness metric long form answer prompt

1 Natural language inference. Use only 'Yes' (1), 'No' (0)

Listing 4.5: Faithfulness metric natural language inference prompt

- **Answer relevancy:** This metric evaluates how relevant the generated answer is to the given prompt, considering the question, context, and answer. Answer relevancy is determined by calculating the mean cosine similarity between the original question and a set of artificial questions generated (reverse engineered) from the answer using the prompt in Listing 4.6. The formula for answer relevancy can be computed using cosine similarity as in Equation 4.7 or with a dot product as in Equation 4.8.

$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^N \cos(E_{g_i}, E_o) \quad (4.7)$$

$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^N \frac{E_{g_i} \cdot E_o}{\|E_{g_i}\| \|E_o\|} \quad (4.8)$$

where:

- E_{g_i} is the embedding of the generated question i .
- E_o is the embedding of the original question.
- N is the number of generated questions, which is 3 default.

1 Generate a question for the given answer and Identify if answer is
 → noncommittal. Give noncommittal as 1 if the answer is
 → noncommittal and 0 if the answer is committal. A noncommittal
 → answer is one that is evasive, vague, or ambiguous. For example,
 → "I don't know" or "I'm not sure" are noncommittal answers.

Listing 4.6: Answer relevancy metric prompt

Results

The main objective of this chapter is to present and analyze the results obtained from the various experimental settings conducted during this study. This chapter is structured to provide a comprehensive overview of the findings, detailing the performance and effectiveness of different components and techniques employed. Firstly, the results from the LLMs evaluation are presented, discussing the performance metrics and ultimately identifying the LLM selected for further experimentation. This section includes a comparative analysis of different LLMs and justifies the choice of the final model based on the evaluation criteria. Next, the performance of each embedding model is shown, examining the results for each of the selected metrics. This analysis highlights the strengths and weaknesses of the various embedding models and their impact on the retrieval process. Lastly the results of the different filtering techniques applied are presented. This section evaluates the effectiveness of each filtering method with respect to the chosen metrics, illustrating how these techniques influence the quality and relevance of the generated answers.

5.1 LLM Evaluation

This section presents the Elo scores calculated for each candidate LLM on the LLM Eval dataset (section 4.1.1), as shown in Table 5.1. This section aims to provide a detailed analysis of the performance of each LLM, highlighting key observations and implications.

First, it is important to discuss the notably poor results observed for the Gemma 7B Instruct model. There is a known issue within the community that the Gemma 7B class of models often produces gibberish, repeats words, and generates nonsensical answers, as can be seen in Figure 5.1. This problem may be attributed to some misconfiguration during model deployment. Since Google has not published the full model inference configuration, including critical parameters such as temperature, repeat penalty, and other penalties, the default configurations used in popular frameworks like HuggingFace and LlamaCPP might be adversely affecting the model's performance. Consequently, these misconfigurations could be

undermining the potential of the Gemma 7B model, leading to its subpar performance.

Beyond the specific case of the Gemma model, it is also evident that all the 7B models have lower Elo scores compared to the paid API models, such as GPT-3.5 Turbo and GPT-4 Turbo. These models, which have orders of magnitude more parameters, consistently outperform the smaller 7B models. This observation underscores the significant impact that model size and the number of parameters have on performance, with larger models generally providing more accurate and coherent responses.

The best performing model in this evaluation is GPT-4 Turbo, achieving the highest Elo score. This result was anticipated, given that GPT-4 Turbo is at least an order of magnitude larger than some of the other models evaluated. However, as detailed in the Cost Estimation section (4.3), conducting large-scale evaluations with GPT-4 Turbo would be prohibitively expensive. The high operational costs associated with using GPT-4 Turbo make it impractical for extensive evaluations and deployment in cost-sensitive applications. As such, the selected model for further use is GPT-3.5 Turbo. This model offers the best cost-performance ratio, making it a viable option for large-scale automatic evaluations and the deployment of a demo web application. GPT-3.5 Turbo provides performant results while maintaining significantly lower costs compared to GPT-4 Turbo, thereby enabling more sustainable and scalable solutions.

Model	Elo Score
google/gemma-7b-it	743
mistralai/Mistral-7B-Instruct-v0.2	849
HuggingFaceH4/zephyr-7b-beta	950
openchat/openchat-3.5-0106	1049
gpt-3.5-turbo-instruct	1151
gpt-4-turbo	1259

Table 5.1: LLM Elo score on LLM Eval dataset

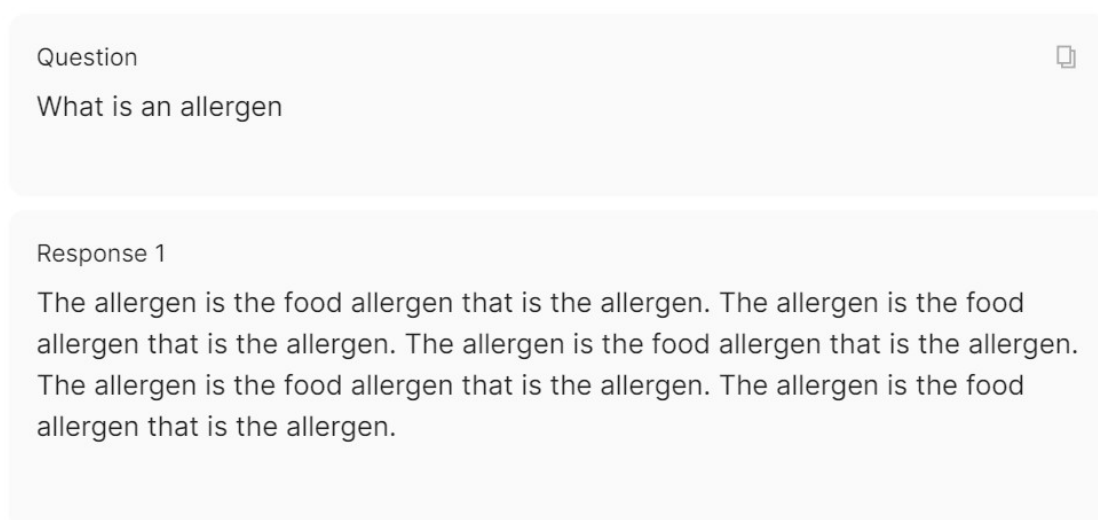


Figure 5.1: Example of answer generated by Gemma 7B Instruct

It is also important to note that the LLM Eval dataset only contains 18 questions at the moment of this evaluation, which can lead to statistically insignificant results. Another critical aspect to mention is that the

evaluation conducted on the Argilla platform was performed by a food scientist at Omnium AI. This human involvement may inevitably introduce small biases into the evaluation process, which must be taken into account when analyzing the results.

5.2 Embedding Model Evaluation

This section presents the results of all the candidate embedding models, as seen in Table 5.2, relative to the selected metrics: context precision, context recall, and context relevancy.

The embeddings with the lowest performance are the jina-embeddings. This was somewhat anticipated, as this model has the smallest parameter count (137M) and the smallest embedding size (768). A smaller embedding size can result in a reduced search space, thereby increasing the likelihood of different contexts having similar embeddings, which negatively impacts performance.

Models with significantly better results include the intfloat/multilingual-e5-large model and the BAAI/bge-large-en-v1.5 model. Both have an embedding size of 1024, with 560M and 335M parameters respectively. Despite the intfloat/multilingual-e5-large model's larger size, the BAAI/bge-large-en-v1.5 model performs markedly better across all evaluated metrics. This superior performance may be attributed to differences in training data, such as the content and context size, making the BAAI/bge-large-en-v1.5 model a more suitable choice for the selected dataset.

Model	Context Precision	Context Recall	Context Relevancy	Average
jinaai/jina-embeddings-v2-base-en	0.3076	0.3508	6.055e-5	0.2195
intfloat/multilingual-e5-large	0.8826	0.8189	0.1336	0.6117
BAAI/bge-large-en-v1.5	0.9611	0.8286	0.1830	0.6576

Table 5.2: Embedding models evaluation results

Given these results, a comparative analysis can now be made. As shown in Figure 5.2, it is evident that the BAAI/bge-large-en-v1.5 model is the best choice, not only based on evaluation metrics but also due to its relatively small size, which makes it more efficient and an excellent candidate for the final system.

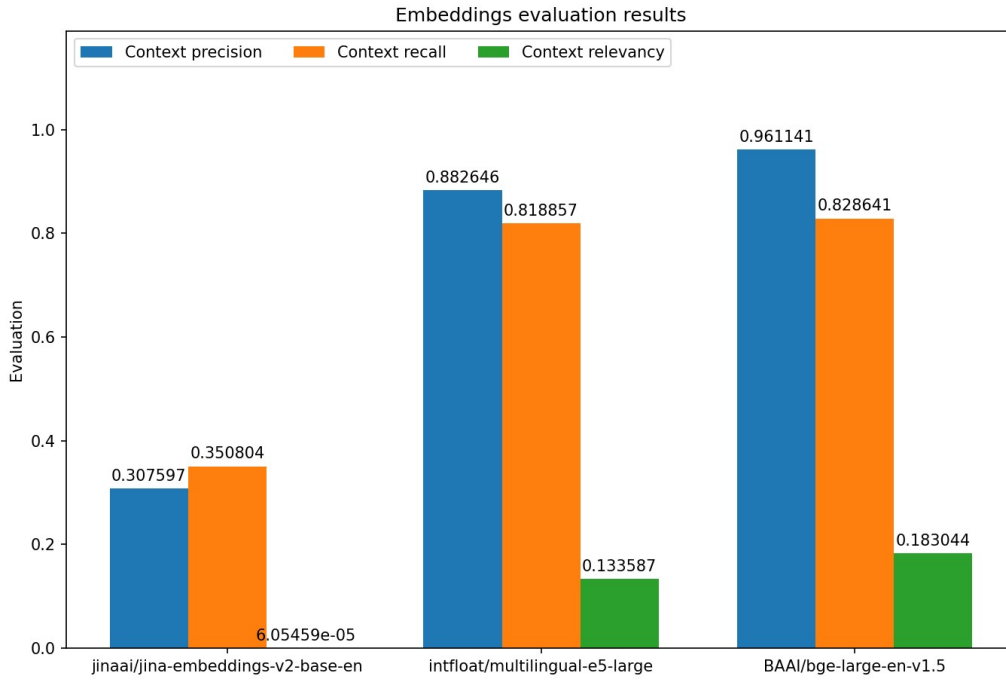


Figure 5.2: Comparison of embedding models results

5.3 Filters Evaluation

This section presents the results of the filter evaluation stage of the evaluation pipeline. In this stage the answers were evaluated using the faithfulness metric and the answer relevancy metric. Looking at the Table 5.3, the first conclusion that can be made is that the results with the CRAG agent behaviour have very poor results. These results show that the selected model (GPT-3.5 Turbo) is not effective at discriminating the selected contexts as relevant or not relevant to answer a question. This lead to many of the questions not having any retrieved contexts at all, making it very difficult for the LLM to give an answer at a later stage. The agents that do not use the CRAG corrective stage perform much better. In particular the use of the BAAI/bge-reranker-v2-m3 reranker gives the best result in answer relevancy since it helps to eliminate irrelevant contexts and therefore irrelevant information from the answer. However when it comes to faithfulness the best combination doesn't involve any filters, this happens because faithfulness is calculated as the number of claims that can be inferred from the contexts over total number of claims. As such having access to more context due to the lack of filters could only either maintain or increase this value.

The comparative analysis made in Figure 5.3 shows that the filtering combination with the best overall evaluation results is the RAG agent with no reranker model, with an average metric score of 0.9139. Even though the BAAI/bge-reranker-v2-m3 is also a good choice, is better not to use a reranker since it comes with increased computational costs and decreased generation efficiency.

Reranker	Agent Type	Faithfulness	Answer Relevancy	Average
BAAI/bge-reranker-v2-m3	CRAG	0.2374	0.0728	0.1551
intfloat/simlm-msmarco-reranker	CRAG	0.2372	0.0668	0.1520
None	CRAG	0.2905	0.0821	0.1863
BAAI/bge-reranker-v2-m3	RAG	0.9415	0.8741	0.9078
intfloat/simlm-msmarco-reranker	RAG	0.9381	0.8636	0.9009
None	RAG	0.9581	0.8696	0.9139

Table 5.3: Filtering mechanisms evaluation results

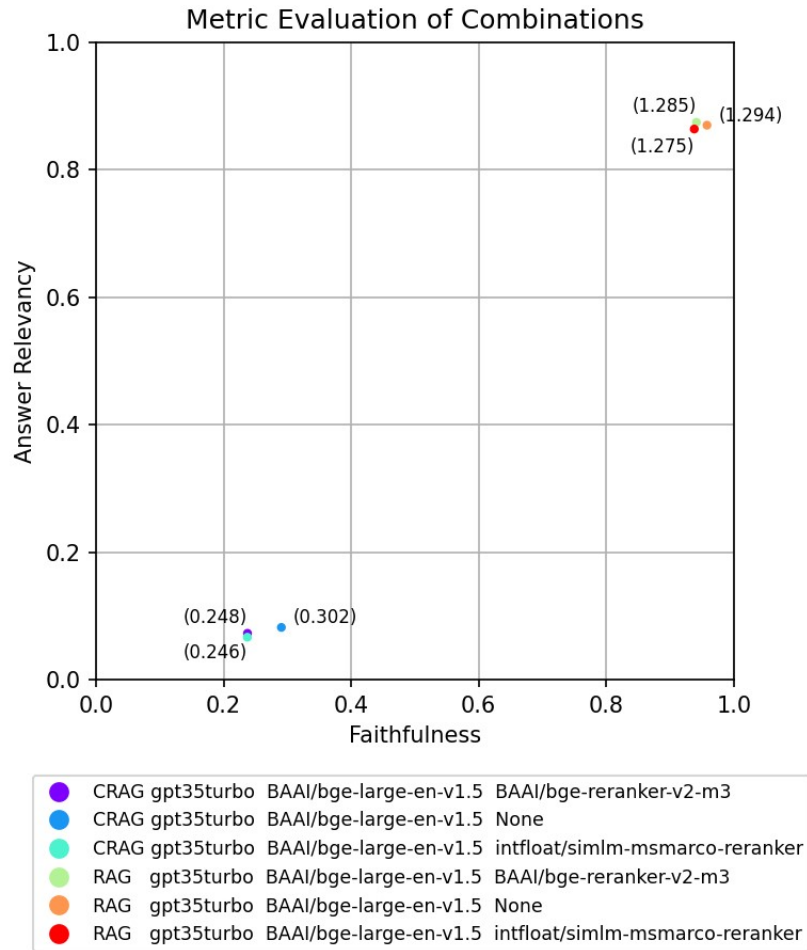


Figure 5.3: Comparison of filtering combinations results (values represent distance to the origin)

5.4 Demo App

This section elaborates on the development and deployment of the demo Chainlit application, highlighting the design decisions and implementation steps. The demo application serves dual purposes: demonstrating the system's performance and functionalities, and collecting valuable user interaction data to refine the agent's capabilities.

Upon initiating a conversation, users are prompted to select a vector database, choosing between the PubMedQA contexts database and the CFR database, both of which utilize datasets detailed in sections

4.1.2 (PubMedQA) and 4.1.3 (CFR). Following the database selection, users choose an LLM, with GPT-3.5 Turbo and GPT-4 Turbo available as options, based on their performance in the LLM Evaluation (section 4.4.1).

Next, users select a reranker, either the BAAI/bge-reranker-v2-m3 or opting for no reranker, reflecting the best-scoring models in terms of answer relevancy and faithfulness as evaluated in the Filters Evaluation (section 4.4.3). Finally, users choose an agent type, RAG or CRAG, despite CRAG's relatively poor performance, it was included to allow for user experimentation. Each response provided by the system is accompanied by the context selected through the retrieval and filtering process, offering transparency and justification for the answers with real data.

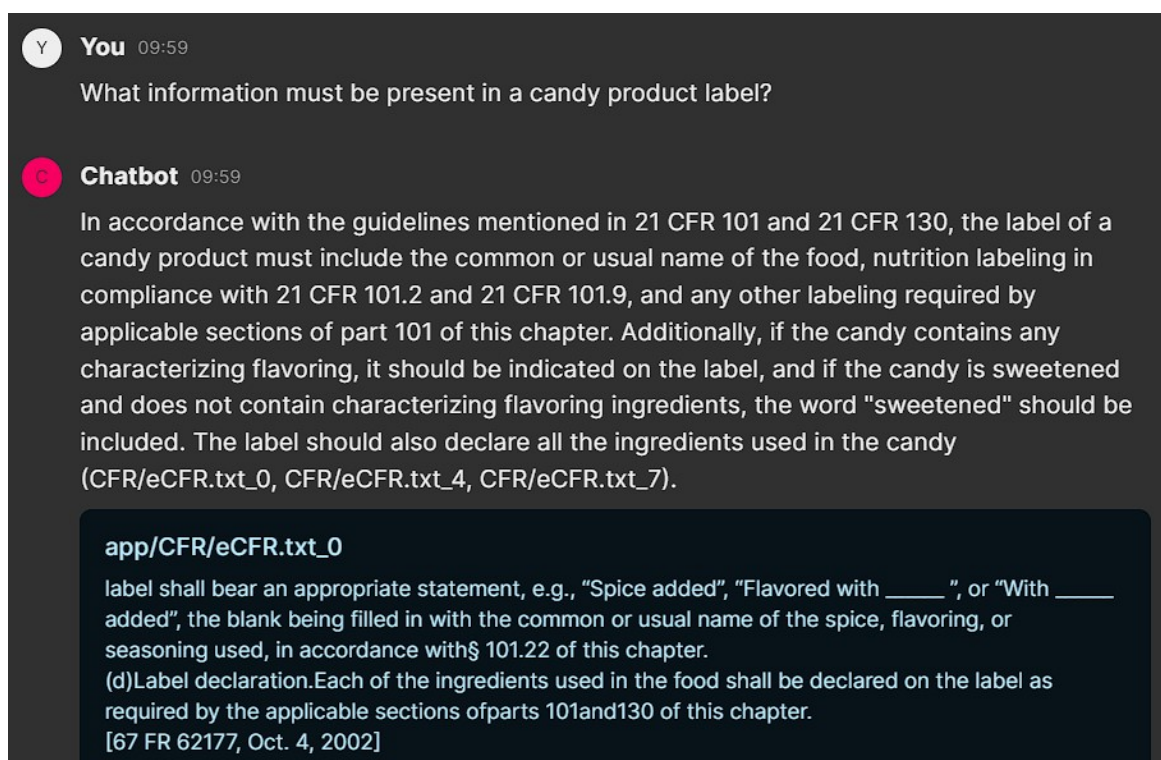


Figure 5.4: RAG demo answer example. CFR contexts database with GPT-3.5 Turbo and no reranker.

Implementation Details

Thanks to the development of the GumGum package detailed in chapter 3.2, the practical implementation of the system is significantly simplified and can be seen in Figure 5.5.

- **LLM Hosting:** The instances of both LLMs (GPT-3.5 Turbo and GPT-4 Turbo) are hosted on the Azure cloud platform. Due to the integration of LangChain and OpenAI in GumGum, utilizing these models is straightforward, as demonstrated in the following code snippet:

```
1 from gumgum.llms import AzureChatOpenAI
2
3 llm = AzureChatOpenAI(
```

```

4     azure_deployment = f"cibus-llm-{selected_llm}-dev-001",
5     azure_endpoint = envs["AZURE_ENDPOINT"],
6     name = "model_name",
7     api_key = envs["AZURE_KEY"],
8     api_version = "2023-06-01-preview",
9 )

```

- **Embedding Model:** The embedding model, running locally, is available on the Huggingface platform. Huggingface provides an [API](#) that is integrated into GumGum, simplifying the process of generating a vector retriever:

```

1     from gumgum.embeddings import HuggingFaceBgeEmbeddings
2     from gumgum.stores import FAISS
3
4     emb = HuggingFaceBgeEmbeddings(model_name="BAAI/bge-large-en-v1.5")
5     ret = FAISS.load_local(
6         selected_vdb, emb, allow_dangerous_deserialization = True
7     ).as_retriever(search_kwargs={"k": int(envs["TOP_K"])})

```

- **Reranker:** The reranker, also running locally and available via Huggingface, can be loaded with the integrated LangChain and Huggingface [APIs](#):

```

1     from gumgum.rerankers import (
2         ContextualCompressionRetriever,
3         CrossEncoderReranker,
4         HuggingFaceCrossEncoder,
5     )
6
7     reranker = HuggingFaceCrossEncoder(model_name=selected_reranker)
8     compressor = CrossEncoderReranker(model=reranker, top_n=5)
9     ret = ContextualCompressionRetriever(
10         base_compressor = compressor,
11         base_retriever = ret
12     )

```

- **RAG/CRAG Agent:** The [RAG](#) or [CRAG](#) agent developed in this work can be instantiated as follows:

```

1     from gumgum.rag import CRAG, RAG
2
3     rag = RAG(llm, ret, retriever_template=retriever_template)

```

```
4 crag = CRAG(llm, ret, retriever_template=retriever_template)
```

A dockerfile was then created to load a python 3.10 environment and install GumGum and all the application requirements. To ensure that it can be accessed from anywhere, port 8000 was exposed to the exterior before running the chainlit execution command. This file was then used to compile the application into a docker image that can be pushed to an azure virtual machine where it can be installed and executed.

To facilitate the storage and subsequent analysis of all conversation data, the Databricks platform was selected by Omnium AI. An [HTTPS](#) endpoint was integrated into the Chainlit web application to return a structured list of conversational turns, capturing essential information such as:

- **conversation_id**: A unique identifier for each conversation between a user and the system.
- **user**: The question posed by the user.
- **assistant**: The final answer generated by the system.
- **contexts**: A list of strings containing the retrieved and filtered contexts used by the [LLM](#) to generate the answer.
- **timestamp**: The epoch time indicating when the question was answered.

On the Databricks side, a dedicated notebook was developed to access the Chainlit endpoint, retrieve the conversation history data, preprocess it into a Pandas DataFrame, and utilize Spark to write the data into an data lake warehouse. To ensure the periodic collection of conversation histories, the notebook was scheduled to run daily at 5 AM, automating the data ingestion process and enabling continuous improvement of the system based on real user interactions.

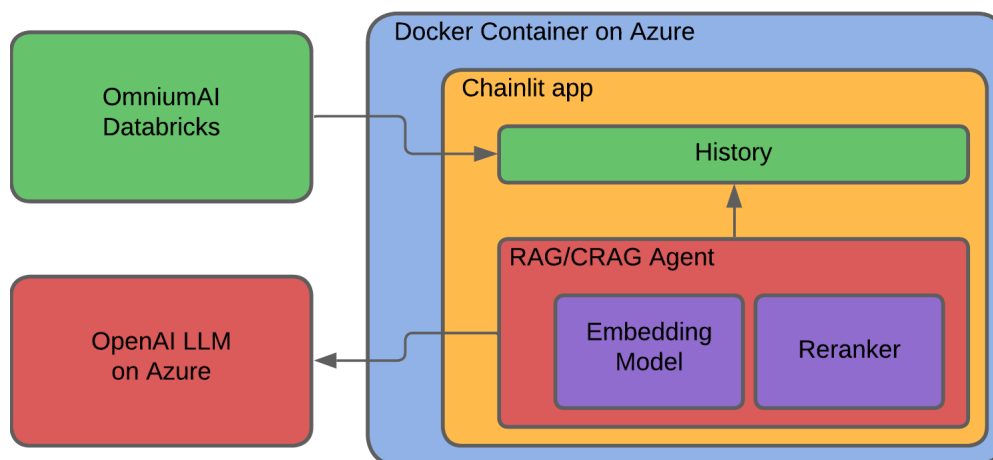


Figure 5.5: RAG demo deployment diagram

Conclusions and Future Work

The integration of [AI](#) technologies into the field of food engineering has the potential to drive significant advancements in both quality and efficiency. This is particularly evident in product reformulation efforts, whether aimed at enhancing taste or improving health benefits. [AI](#)-powered conversational systems can play a pivotal role in this transformation by providing food engineers with a more accessible and user-friendly interface to interact with technology. These systems empower engineers with rapid, conversational access to necessary information, facilitating more informed decision-making processes.

The primary objective of this work was to develop a platform that simplifies the creation, testing, and deployment of [LLM](#)-based systems. The resulting framework, GumGum (section [3.2](#)), serves as a robust foundation for engineers at Omnium AI to build versatile and powerful conversational agents. These agents can help clients gain deeper insights and access information that would otherwise be time-consuming to obtain and interpret. GumGum's design emphasizes speed, simplicity, and efficiency, ensuring that even those with limited technical expertise can harness the full potential of [LLM](#) technologies.

In addition to developing the GumGum package, this work also focused on building and deploying a [RAG](#) system to evaluate existing solutions and assess their strengths and limitations. This comprehensive evaluation provided several key insights. Firstly, it highlighted the potential costs associated with deploying various types of agents using different [LLMs](#). Understanding these cost implications is crucial for making informed decisions about the practical deployment of [AI](#) solutions. Secondly, the study offered valuable perspectives on what constitutes a successful agent solution. This includes insights into the choice of [LLMs](#), embedding models, and the potential benefits of incorporating rerankers to enhance performance.

While this work has laid a solid foundation, there are several avenues for future research and development that could further optimize the system for food engineering applications. One significant area of improvement involves the mass collection of data and text corpora relevant to the field. Expanding the information sources available to the system can significantly enhance its knowledge base and accuracy. This effort would complement the datasets currently in use, such as the PubMedQA dataset (section [4.1.2](#))

and the [CFR](#) dataset (section [4.1.3](#)).

This work shows that the integration of [AI](#) technologies into food engineering holds immense promise for enhancing the work of food engineers. The development of the GumGum framework and the deployment of the [RAG](#) system represent significant steps towards achieving this vision. By continuing to refine and expand these tools, food engineers can be empowered to make better-informed decisions, ultimately leading to innovations that benefit both industry and consumers.

Bibliography

- [1] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, W.-t. Yih, and F. Ai. “Dense Passage Retrieval for Open-Domain Question Answering.” In: (n.d.). url: <https://arxiv.org/abs/2004.04906>.
- [2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, and C. Hesse. “Language Models are Few-Shot Learners.” In: (2020). url: <https://arxiv.org/abs/2005.14165>.
- [3] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. “Improving Language Understanding by Generative Pre-Training.” In: (2018). url: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.
- [4] K. Lv, Y. Yang, T. Liu, Q. Gao, Q. Guo, and X. Qiu. “Full Parameter Fine-tuning for Large Language Models with Limited Resources.” In: (2023). url: <https://arxiv.org/abs/2306.09782>.
- [5] W. H. Organization. *Healthy Diet*. World Health Organisation, 2020. url: <https://www.who.int/news-room/fact-sheets/detail/healthy-diet>.
- [6] *Reformulation of Food and Beverage Products for Healthier Diets: Policy Brief*. n.d. World Health Organization. url: <https://www.who.int/publications/i/item/9789240039919>.
- [7] J. Fanzo, R. McLaren, A. Bellows, and B. Carducci. “Challenges and opportunities for increasing the effectiveness of food reformulation and fortification to improve dietary and nutrition outcomes.” In: *Food Policy* 119 (2023), p. 102515. doi: [10.1016/j.foodpol.2023.102515](https://doi.org/10.1016/j.foodpol.2023.102515). url: <https://doi.org/10.1016/j.foodpol.2023.102515>.
- [8] Food and D. Administration. *HACCP Principles & Application Guidelines*. U.S. Food and Drug Administration. 2022. url: <https://www.fda.gov/food/hazard-analysis-critical-control-point-haccp/haccp-principles-application-guidelines>.
- [9] FDA. *Generally Recognized as Safe (GRAS)*. 2019. url: <https://www.fda.gov/food/food-ingredients-packaging/generally-recognized-safe-gras>.
- [10] P. Ongsulee. “Artificial Intelligence, Machine Learning and Deep Learning.” In: *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)* (2017).

- [11] G. Barbastathis, A. Ozcan, and G. Situ. "On the Use of Deep Learning for Computational Imaging." In: *Optica* 6.8 (2019), p. 921. doi: [10.1364/optica.6.000921](https://doi.org/10.1364/optica.6.000921). url: <https://doi.org/10.1364/optica.6.000921>.
- [12] A. Singhal, P. Sinha, and R. Pant. "Use of Deep Learning in Modern Recommendation System: A Summary of Recent Works." In: *International Journal of Computer Applications* 180.7 (2017), pp. 17–22. doi: [10.5120/ijca2017916055](https://doi.org/10.5120/ijca2017916055). url: <https://doi.org/10.5120/ijca2017916055>.
- [13] S. D. Holcomb, W. K. Porter, S. V. Ault, G. Mao, and J. Wang. "Overview on DeepMind and Its AlphaGo Zero AI." In: *Proceedings of the 2018 International Conference on Big Data and Education - ICBDE '18*. 2018. doi: [10.1145/3206157.3206174](https://doi.org/10.1145/3206157.3206174). url: <https://doi.org/10.1145/3206157.3206174>.
- [14] J. X. Chen. "The Evolution of Computing: AlphaGo." In: *Computing in Science & Engineering* 18.4 (2016), pp. 4–7. doi: [10.1109/mcse.2016.74](https://doi.org/10.1109/mcse.2016.74). url: <https://doi.org/10.1109/mcse.2016.74>.
- [15] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy Eleutherai. "The Pile: An 800GB Dataset of Diverse Text for Language Modeling." In: (2020). url: <https://arxiv.org/pdf/2101.00027.pdf>.
- [16] L. N. Kanal. *Perceptron*. In: *Encyclopedia of Computer Science*. John Wiley and Sons Ltd., 2003.
- [17] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete. "A survey on modern trainable activation functions." In: *Neural Networks* 138 (2021), pp. 14–32. doi: [10.1016/j.neunet.2021.01.026](https://doi.org/10.1016/j.neunet.2021.01.026).
- [18] L. Almeida. *C1.2 Multilayer Perceptrons*. 1997. url: <http://www.lx.it.pt/~lbalmeida/papers/AlmeidaHNC.pdf>.
- [19] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Muller. "Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications." In: *Proceedings of the IEEE* 109.3 (2021), pp. 247–278. doi: [10.1109/jproc.2021.3060483](https://doi.org/10.1109/jproc.2021.3060483). url: <https://doi.org/10.1109/jproc.2021.3060483>.
- [20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Representations by Back-Propagating Errors." In: *Nature* 323.6088 (1986), pp. 533–536. doi: [10.1038/323533a0](https://doi.org/10.1038/323533a0).
- [21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [22] S. Ruder. "An Overview of Gradient Descent Optimization Algorithms." In: (2017). url: <https://arxiv.org/pdf/1609.04747.pdf>.
- [23] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015. url: <https://arxiv.org/pdf/1512.03385.pdf>.

-
- [24] Y. Bohra. *Vanishing and Exploding Gradients in Deep Neural Networks*. Analytics Vidhya. 2021. url: <https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/>.
 - [25] S. Hochreiter. "Recurrent Neural Net Learning and Vanishing Gradient." In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.2 (1998), pp. 107–116. url: <https://www.bioinf.jku.at/publications/older/2904.pdf>.
 - [26] M. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. n.d. url: <https://arxiv.org/pdf/1212.5701.pdf>.
 - [27] P. Kunchala. *A brief explanation of Overfitting, Underfitting, Variance and Bias*. Analytics Vidhya. 2021. url: <https://medium.com/analytics-vidhya/a-brief-explanation-of-overfitting-underfitting-variance-and-bias-d407c4c63bc8>.
 - [28] Built In. *L1 and L2 Regularization Methods, Explained | Built In*. Retrieved from <https://builtin.com/data-science/l2-regularization>. n.d. url: <https://builtin.com/data-science/l2-regularization>.
 - [29] S. J. Jain. *An Overview of Regularization Techniques in Deep Learning (with Python code)*. Analytics Vidhya. 2018. url: <https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/#h-data-augmentation>.
 - [30] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. url: <https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>.
 - [31] E. Sezerer and S. Tekir. *A Survey On Neural Word Embeddings*. Retrieved November 13, 2023. n.d. url: <https://arxiv.org/pdf/2110.01804.pdf>.
 - [32] A. Goldberg. "Alfred Goldberg." In: *Current Biology* 24.17 (2014), R780–R782. doi: [10.1016/j.cub.2014.08.014](https://doi.org/10.1016/j.cub.2014.08.014).
 - [33] J. Pennington, R. Socher, and C. Manning. "GloVe: Global Vectors for Word Representation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. 2014, pp. 1532–1543. url: <https://aclanthology.org/D14-1162.pdf>.
 - [34] B. Athiwaratkun, A. Wilson, and A. Anandkumar. "Probabilistic FastText for Multi-Sense Word Embeddings." In: (n.d.). url: <https://arxiv.org/pdf/1806.02901.pdf>.
 - [35] N. Altman and M. Krzywinski. "The curse(s) of dimensionality." In: *Nature Methods* 15.6 (2018), pp. 399–400. doi: [10.1038/s41592-018-0019-x](https://doi.org/10.1038/s41592-018-0019-x).
 - [36] C. Kozyrkov. "The Curse of Dimensionality... minus the curse of jargon." In: *Medium* (2022). url: <https://towardsdatascience.com/the-curse-of-dimensionality-minus-the-curse-of-jargon-520da109fc87>.

- [37] X. Zhang, C. Xu, M. Shen, X. He, and W. Du. "Survey of Convolutional Neural Network." In: *Proceedings of the 2018 International Conference on Network, Communication, Computer Engineering (NCCE 2018)*. Atlantis Press, 2018, pp. 93–97. isbn: 978-94-6252-517-7. doi: [10.2991/ncce-18.2018.16](https://doi.org/10.2991/ncce-18.2018.16). url: <https://doi.org/10.2991/ncce-18.2018.16>.
- [38] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga. "A Comprehensive Survey of Deep Learning for Image Captioning." In: *ACM Comput. Surv.* 51.6 (2019). doi: [10.1145/3295748](https://doi.org/10.1145/3295748). url: <https://doi.org/10.1145/3295748>.
- [39] D. Tang, B. Qin, X. Feng, and T. Liu. "Effective LSTMs for Target-Dependent Sentiment Classification." In: (2016). url: <https://arxiv.org/pdf/1512.01100.pdf>.
- [40] I. Google, V. Oriol, Google, and Q. Le Google. "Sequence to Sequence Learning with Neural Networks." In: (2014). url: https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.
- [41] K. O'Shea and R. Nash. "An Introduction to Convolutional Neural Networks." In: (2015). url: <https://arxiv.org/pdf/1511.08458.pdf>.
- [42] S. Albawi, T. A. Mohammed, and S. Al-Zawi. "Understanding of a Convolutional Neural Network." In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. doi: [10.1109/icengtechnol.2017.8308186](https://doi.org/10.1109/icengtechnol.2017.8308186).
- [43] V. Dumoulin and F. Visin. "A Guide to Convolution Arithmetic for Deep Learning." In: (2018). url: <https://arxiv.org/pdf/1603.07285.pdf>.
- [44] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. "How to Construct Deep Recurrent Neural Networks." In: (2014). url: <https://arxiv.org/pdf/1312.6026.pdf>.
- [45] R. Staudemeyer and E. Morris. "Understanding LSTM - a tutorial into Long Short-Term Memory Recurrent Neural Networks." In: (2019). url: <https://arxiv.org/pdf/1909.09586.pdf>.
- [46] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory." In: *Neural Computation* 9.8 (1997), pp. 1735–1780. doi: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735).
- [47] A. Vaswani, G. Brain, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, . Kaiser, and I. Polosukhin. "Attention Is All You Need." In: 2017. url: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- [48] Z. Niu, G. Zhong, and H. Yu. "A review on the attention mechanism of deep learning." In: *Neurocomputing* 452 (2021), pp. 48–62. doi: [10.1016/j.neucom.2021.03.091](https://doi.org/10.1016/j.neucom.2021.03.091).
- [49] S. Zhao and Z. Zhang. "Attention-via-attention neural machine translation." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2018, pp. 563–570.
- [50] D. Britz, A. Goldie, M.-T. Luong, and Q. V. Le. "Massive Exploration of Neural Machine Translation Architectures." In: *CoRR* abs/1703.03906 (2017). url: <http://arxiv.org/abs/1703.03906>.

-
- [51] T. Luong, H. Pham, and C. Manning. “Effective Approaches to Attention-Based Neural Machine Translation.” In: *EMNLP*. 2015, pp. 1412–1421.
 - [52] A. Graves, G. Wayne, and I. Danihelka. “Neural Turing Machines.” In: *CoRR* abs/1410.5401 (2014). url: <http://arxiv.org/abs/1410.5401>.
 - [53] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. “Convolutional Sequence to Sequence Learning.” In: *Proceedings of Machine Learning Research*. Vol. 70. PMLR, 2017. url: <https://proceedings.mlr.press/v70/gehring17a.html?ref=https://githubhelp.com>.
 - [54] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. “RoFormer: Enhanced transformer with Rotary Position Embedding.” In: *Neurocomputing* 568 (2024), p. 127063. doi: [10.1016/j.neucom.2023.127063](https://doi.org/10.1016/j.neucom.2023.127063). url: <https://doi.org/10.1016/j.neucom.2023.127063>.
 - [55] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: (2019). url: <https://arxiv.org/pdf/1810.04805.pdf>.
 - [56] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. Le. “XLNet: Generalized Autoregressive Pretraining for Language Understanding.” In: (n.d.). url: https://proceedings.neurips.cc/paper_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.
 - [57] *Notion – The all-in-one workspace for your notes, tasks, wikis, and databases*. Retrieved November 23, 2023. n.d. url: <https://coatue-external.notion.site/AI-2022-The-Explosion-e76afd140f824f2eb6b049c5b85a7877>.
 - [58] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. De, L. Casas, L. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. Van Den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, and E. Elsen. *Training Compute-Optimal Large Language Models*. n.d. url: <https://arxiv.org/pdf/2203.15556.pdf>.
 - [59] S. Gunasekar, Y. Zhang, J. Aneja, C. César, T. Mendes, A. Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. De, R. Olli, S. Adil, S. Shital, S. Harkirat, S. Behl, X. Wang, S. Bubeck, R. Eldan, A. Tauman, and K. Yin. *Textbooks Are All You Need*. n.d. url: <https://arxiv.org/pdf/2306.11644.pdf> (visited on 11/23/2023).
 - [60] V. Singh. *A Guide to Controlling LLM Model Output: Exploring Top-k, Top-p, and Temperature Parameters*. <https://ivibudh.medium.com/a-guide-to-controlling-llm-model-output-exploring-top-k-top-p-and-temperature-parameters-ed6a31313910>. Medium. 2023.
 - [61] W. Zhu, H. Hao, and R. Wang. *Penalty Decoding: Well Suppress the Self-Reinforcement Effect in Open-Ended Text Generation*. <https://arxiv.org/pdf/2310.14971>. Retrieved July 30, 2024. 2024.

-
- [62] K. W. Church, Z. Chen, and Y. Ma. “Emerging Trends: A Gentle Introduction to Fine-Tuning.” In: *Natural Language Engineering* 27.6 (2021), pp. 763–778. doi: [10.1017/s1351324921000322](https://doi.org/10.1017/s1351324921000322). url: <https://doi.org/10.1017/s1351324921000322>.
 - [63] *Training vs. Fine-tuning: What is the Difference?* Retrieved January 4, 2024. url: <https://encord.com/blog/training-vs-fine-tuning/#h3>.
 - [64] S. Zhang, D. Linfeng, X. Li, S. Zhang, S. Xiaofei Wang, J. Li, R. Hu, T. Zhang, F. Wu, and G. Wang. *Instruction Tuning for Large Language Models: A Survey*. n.d. url: <https://arxiv.org/pdf/2308.10792.pdf> (visited on 01/07/2024).
 - [65] *Learning from Human Preferences*. url: <https://openai.com/research/learning-from-human-preferences>.
 - [66] Z. Li, Z. Yang, and M. Wang. *Reinforcement Learning with Human Feedback: Learning Dynamic Choices via Pessimism*. 2023. url: <https://arxiv.org/pdf/2305.18438.pdf>.
 - [67] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. *Training Language Models to Follow Instructions with Human Feedback*. n.d. url: https://cdn.openai.com/papers/Training_language_models_to_follow_instructions_with_human_feedback.pdf.
 - [68] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. OpenAI. *Proximal Policy Optimization Algorithms*. 2017. url: <https://arxiv.org/pdf/1707.06347.pdf>.
 - [69] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. Hinton, and J. Dean. “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer.” In: *arXiv preprint arXiv:1701.06538* (2017). url: <https://arxiv.org/pdf/1701.06538>.
 - [70] W. Fedus, B. Zoph, N. Shazeer, and A. Clark. “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity.” In: *Journal of Machine Learning Research* 23 (2022), pp. 1–40. url: <https://arxiv.org/pdf/2101.03961>.
 - [71] Q. Ye, M. Axmed, R. Pryzant, and F. Khani. *PROMPT ENGINEERING A PROMPT ENGINEER*. Retrieved January 8, 2024. n.d. url: <https://arxiv.org/pdf/2311.05661.pdf>.
 - [72] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi Quoc, V. Le, and D. Zhou. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models Chain-of-Thought Prompting*. arXiv preprint. 2023. url: <https://arxiv.org/pdf/2201.11903.pdf>.
 - [73] S. Yao, D. Yu, G. Deepmind, J. Zhao, T. Griffiths, Y. Cao, and K. Narasimhan. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. Manuscript submitted for publication. 2023. url: <https://arxiv.org/pdf/2305.10601.pdf>.
 - [74] S. Chen, D. Beeferman, and R. Rosenfeld. *EVALUATION METRICS FOR LANGUAGE MODELS*. n.d. url: <https://www.cs.cmu.edu/~roni/papers/eval-metrics-bntuw-9802.pdf>.

-
- [75] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. "BLEU: a Method for Automatic Evaluation of Machine Translation." In: (2002). url: <https://aclanthology.org/P02-1040.pdf>.
 - [76] K. Ganesan. *ROUGE 2.0: Updated and Improved Measures for Evaluation of Summarization Tasks*. n.d. url: <https://arxiv.org/pdf/1803.01937.pdf>.
 - [77] S. Banerjee and A. Lavie. "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments." In: (2005). url: <https://aclanthology.org/W05-0909.pdf>.
 - [78] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. n.d. url: <https://arxiv.org/pdf/1804.07461.pdf>.
 - [79] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*. n.d. url: <https://arxiv.org/pdf/1905.00537.pdf>.
 - [80] Papers with Code. *Papers with Code - SQuAD2.0 Benchmark (Question Answering)*. <https://paperswithcode.com/sota/question-answering-on-squad20>. n.d.
 - [81] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, Y. Choi, and P. Allen. *HellaSwag: Can a Machine Really Finish Your Sentence?* n.d. url: <https://arxiv.org/pdf/1905.07830.pdf>.
 - [82] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Facebook, and A. Research. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. n.d. url: <https://arxiv.org/pdf/2005.11401.pdf>.
 - [83] 12 Retrieval Augmented Generation (RAG) Tools / Software in '23. *Research*. Retrieved December 18, 2023. n.d. url: <https://research.aimultiple.com/retrieval-augmented-generation/>.
 - [84] *What is a Vector Index? An Introduction to Vector Indexing*. <https://www.datastax.com/guides/what-is-a-vector-index>. DataStax. n.d. (Visited on 12/18/2023).
 - [85] O. Jafari. *A Survey on Locality Sensitive Hashing Algorithms and their Applications*. 2021. url: <https://arxiv.org/pdf/2102.08942.pdf>.
 - [86] P. Chang. *Similarity Search with IVFPQ*. Medium. 2022. url: <https://towardsdatascience.com/similarity-search-with-ivfpq-9c6348fd4db3>.
 - [87] Y. A. Malkov and D. A. Yashunin. "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4 (2020), pp. 824–836. doi: [10.1109/tpami.2018.2889473](https://doi.org/10.1109/tpami.2018.2889473). url: <https://doi.org/10.1109/tpami.2018.2889473>.
 - [88] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan. "Cosine similarity to determine similarity measure: Study case in online essay assessment." In: *IEEE Xplore* (2016). doi: [10.1109/CITSM.2016.7577578](https://doi.org/10.1109/CITSM.2016.7577578). url: <https://doi.org/10.1109/CITSM.2016.7577578>.

-
- [89] L. Liberti, C. Lavor, N. Maculan, and A. Mucherino. “Euclidean Distance Geometry and Applications.” In: *SIAM Review* 56 (May 2012). doi: [10.1137/120875909](https://doi.org/10.1137/120875909).
 - [90] R Suwanda, Z. Syahputra, and E. Zamzami. “Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K.” In: *Journal of Physics: Conference Series* 1566 (June 2020), p. 012058. doi: [10.1088/1742-6596/1566/1/012058](https://doi.org/10.1088/1742-6596/1566/1/012058).
 - [91] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu. “Using of Jaccard Coefficient for Keywords Similarity.” In: Mar. 2013.
 - [92] M. Glass, G. Rossiello, F. Chowdhury, A. Rajaram Naik, P. Cai, and A. Gliozzo. *Title of the Work*. Retrieved January 7, 2024. n.d. url: <https://arxiv.org/pdf/2207.06300.pdf>.
 - [93] Pinecone. *Rerankers and Two-Stage Retrieval* | Pinecone. Website. n.d. url: <https://www.pinecone.io/learn/series/rag/rerankers/>.
 - [94] S.-Q. Yan, J.-C. Gu, Y. Zhu, and Z.-H. Ling. *Corrective Retrieval Augmented Generation*. Accessed: 2024-07-30. 2024. url: <https://arxiv.org/pdf/2401.15884>.
 - [95] L. Monigatti. *Evaluating RAG Applications with RAGAs*. 2023. url: <https://towardsdatascience.com/evaluating-rag-applications-with-ragas-81d67b0ee31a>.
 - [96] Weaviate - Vector Database. *An Overview on RAG Evaluation*. Weaviate.io. 2023. url: <https://weaviate.io/blog/rag-evaluation#generation-metrics>.
 - [97] Ragas. *Metrics*. Retrieved January 8, 2024, from <https://docs.ragas.io/en/latest/concepts/metrics/index.html>. n.d. url: <https://docs.ragas.io/en/latest/concepts/metrics/index.html>.
 - [98] K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, S. Pfohl, P. Payne, M. Seneviratne, P. Gamble, C. Kelly, A. Babiker, N. Schärli, A. Chowdhery, P. Mansfield, D. Demner-Fushman, and Blaise. “Large language models encode clinical knowledge.” In: *Nature* (2023). Advance online publication. doi: [10.1038/s41586-023-06291-2](https://doi.org/10.1038/s41586-023-06291-2). url: <https://doi.org/10.1038/s41586-023-06291-2>.
 - [99] Y. Gu, R. Tinn, H. Cheng, M. Lucas, N. Usuyama, X. Liu, T. Naumann, J. Gao, and H. Poon. *Domain-Specific Language Model Pretraining for Biomedical Natural Language Processing*. arXiv preprint arXiv:2007.15779. n.d. url: <https://arxiv.org/pdf/2007.15779.pdf>.
 - [100] PAPERQA. *PAPERQA: Retrieval-Augmented Generative Agent for Scientific Research*. n.d. url: <https://openreview.net/pdf?id=cLU5xWyItb> (visited on 01/24/2024).
 - [101] K. a. C. A. Beltagy Iz and Lo. *SCIBERT: A Pretrained Language Model for Scientific Text*. arXiv preprint. 2019. url: <https://arxiv.org/pdf/1903.10676.pdf>.
 - [102] R. Luo, L. Sun, Y. Xia, T. Qin, S. Zhang, H. Poon, and T.-Y. Liu. “BioGPT: Generative Pre-trained Transformer for Biomedical Text Generation and Mining.” In: *Briefings in Bioinformatics* (2022). doi: [10.1093/bib/bbac409](https://doi.org/10.1093/bib/bbac409). url: <https://doi.org/10.1093/bib/bbac409>.

- [103] T. Xie, Y. Wan, W. Huang, Z. Yin, Y. Liu, S. Wang, Q. Linghu, C. Kit, C. Grazian, W. Zhang, I. Razzak, and B. Hoex. *DARWIN SERIES: DOMAIN SPECIFIC LARGE LANGUAGE MODELS FOR NATURAL SCIENCE A PREPRINT*. Preprint. 2023. url: <https://arxiv.org/pdf/2308.13565.pdf>.
- [104] D. Soong, S. Sridhar, H. Si, J.-S. Wagner, A. Sá, C. Yu, K. Karagoz, M. Guan, H. Hamadeh, and B. Higgs. *Improving accuracy of GPT-3/4 results on biomedical data using a retrieval-augmented language model*. Retrieved January 24, 2024. n.d. url: <https://arxiv.org/pdf/2305.17116.pdf>.
- [105] W. Hiesinger, C. Zakka, S. Chaurasia, R. Shad, A. R. Dalal, J. Kim, M. Moor, K. M. Alexander, E. A. Ashley, J. Boyd, K. Boyd, K. G. Hirsch, C. P. Langlotz, and J. Nelson. *Almanac: Retrieval-Augmented Language Models for Clinical Medicine*. Research Square. 2023. doi: [10.21203/rs.3.rs-2883198/v1](https://doi.org/10.21203/rs.3.rs-2883198/v1).
- [106] Chatbot Arena. *Chatbot Arena*. Accessed: 2024-07-09. url: <https://openlm.ai/chatbot-arena/>.