**IPSSI**

**École supérieure d'informatique et du numérique**

**Master 2 - Dev, Data & IA**

**2022-2023**

# Evolution of ESG data processing with the advent of AI, NLP and Large Language Models

**Abstract**

Environmental, Social, and Governance (ESG) data processing has become an essential aspect of modern finance and investment decision-making. This thesis examines the role of Artificial Intelligence (AI), Natural Language Processing (NLP), and Large Language Models (LLMs) in improving ESG data processing and analysis. Traditional approaches to ESG data processing face numerous challenges, including the oligopolistic nature of the ESG data market, diverse data sources and formats, and issues related to data quality and reliability. This thesis explores the potential of AI, NLP, and LLMs to address these challenges and enhance ESG data processing.

We begin by reviewing the foundations of AI, NLP, and Semantic Search, followed by a detailed examination of the deep learning innovations that led to the development of LLMs, such as the Transformer architecture and models like BERT and GPT-3. Next, we discuss the potential applications of these technologies in ESG data processing, including automating data extraction, advanced NLP techniques for data analysis, and AI-driven ESG risk assessment and prediction. We also explore the use of ChatGPT in ESG data processing, enabling new forms of data interaction and facilitating data access and interpretation.

Finally, we present case studies of successful AI and LLM implementations in ESG data processing, discussing lessons learned, best practices, and the challenges and limitations faced. In conclusion, we outline future trends and implications of AI and LLMs in ESG data processing, highlighting potential ethical and practical challenges, and providing recommendations for further research and development.

**Keywords: ESG data processing, AI, NLP, Large Language Models (LLM), Semantic Search, Deep Learning, Transformer architecture, GPT, ESG risk assessment**

**Under the supervision of Sayf Bejaoui**

**Presented and defended by**

**Romain Jouhameau**

**IPSSI neither approves nor disapproves the opinions expressed in this dissertation: they should be considered as the author's own.**

# Contents

# 1 Introduction

Environmental, Social, and Governance (ESG) factors have become increasingly important in the financial landscape. They play a crucial role in guiding investment decisions, risk management, and regulatory compliance. As a result, there is a growing demand for efficient and reliable ESG data processing.

The advent of Artificial Intelligence (AI), Natural Language Processing (NLP), and Large Language Models (LLMs) has had a transformative impact on various fields, including ESG data processing. These advanced technologies have the potential to address many of the challenges associated with traditional ESG data processing methods, such as data fragmentation, manual data extraction, and data quality issues.

In this thesis, we seek to answer the question: to what extent do AI, LLMs, and other advanced techniques offer the potential to disrupt the ESG data processing industry, and can they bring about a significant transformation in the way ESG data is processed and analyzed?

This thesis aims to explore the impact of AI, NLP, and LLMs on ESG data processing and their implications for ESG finance. The study is structured as follows:

- Section 2 discusses traditional approaches to ESG data processing and their limitations.
- Section 3 provides an overview of AI, NLP, and LLMs, and their applications in data processing and analysis.
- Section 4 delves into AI and LLMs in ESG data processing, including automating ESG data extraction, data analysis using advanced NLP techniques, and AI-driven ESG risk assessment and prediction.
- Section 5 explores the applications of ChatGPT and other LLMs in ESG data interaction.
- Section 6 presents case studies of successful AI and LLM implementations in ESG data processing, lessons learned, and best practices.
- Section 7 discusses future trends and implications, including the role of AI and LLMs in shaping the future of ESG data processing and potential ethical and practical challenges.
- Section 8 concludes the thesis with a summary of key findings, implications for ESG data processing and the financial sector, and final remarks.

This thesis aims to provide a comprehensive overview of the existing AI and LLM landscape in ESG data processing, tracing its evolution and showcasing examples of what can be achieved through

open-source tools and affordable APIs like OpenAI. While the discussion will not delve into highly technical details or demonstrations, the thesis seeks to be as precise as possible in presenting the current state of the field and the potential advancements in the near future, without being overly technical.

# 2 ESG data processing: Traditional approaches

In this section, we'll provide brief answers to the following questions: What is ESG finance? Why has it grown rapidly since the 2010s? Then, we'll take a closer look at the main players in the ESG data production market, and the main characteristics of ESG data.

## 2.1 ESG Finance

### 2.1.1 An Overview

ESG finance, also known as sustainable or responsible finance, refers to the integration of Environmental, Social, and Governance factors into investment decision-making and financial analysis.

The disclosure of extra-financial information is essential to direct financial flows towards the most sustainable projects over the long term, and environmental (E), social (S) and governance (G) criteria must be given the necessary importance, particularly in the investment decisions of investors and managers. While ESG finance has been particularly emphasized in Europe, it is a rapidly growing phenomenon worldwide, playing a crucial role in allocating financial resources towards more sustainable activities and achieving global environmental goals (such as ecological transition).

To give a clearer idea of the different indicators or data involved in ESG rating, some of the main indicators are listed in the table below1. These data or indicators can be separated by their type, either environmental, social or governance.

These indicators can be measured using a variety of data sources, including company reports, third-party audits, and public records. The specific indicators used can vary depending on the industry and the specific ESG goals of the investor.

In the remainder of this thesis, we will focus on environmental issues, when we refer to ESG criteria. That is because non-financial information plays a critical role in climate risk management, as it enables investors, companies, and other market participants to better understand and anticipate the impacts of climate change on financial performance and market stability. It is also, as we will see, more and more, a requirement due to increasing regulation.

| Indicator | Description | Category |
|---|---|---|
| Carbon footprint | This measures the total greenhouse gas emissions caused directly and indirectly by a person, organization, event, or product. | Environmental |
| Water usage | This measures the amount of water used by the company in its operations. | Environmental |
| Waste management | This measures how the company manages its waste, including recycling efforts. | Environmental |
| Energy efficiency | This measures how efficiently a company uses energy in its operations. | Environmental |
| Biodiversity impact | This measures the impact of the company's operations on local biodiversity. | Environmental |
| Employee turnover | High turnover can indicate poor working conditions or low employee satisfaction. | Social |
| Diversity and inclusion | This measures the diversity of the workforce and the company's efforts to promote inclusion. | Social |
| Human rights | This measures the company's commitment to upholding human rights in its operations. | Social |
| Community engagement | This measures the company's efforts to engage with the local community and contribute positively to it. | Social |
| Health and safety | This measures the company's commitment to maintaining a safe and healthy working environment. | Social |
| Board diversity | This measures the diversity of the company's board of directors. | Governance |
| Executive compensation | This measures how the company compensates its executives, including whether compensation is tied to ESG performance. | Governance |
| Corruption and bribery | This measures the company's efforts to prevent corruption and bribery in its operations. | Governance |
| Shareholder rights | This measures the company's commitment to protecting the rights of its shareholders. | Governance |
| Transparency and disclosure | This measures the company's commitment to transparency and the disclosure of relevant information to stakeholders. | Governance |

Table 1: ESG Indicators

### 2.1.2 ESG Finance benefits from increasing regulation

The importance of addressing climate change and environmental degradation has been recognized globally, with initiatives such as the 2015 Paris Agreement, where the main objective is to keep "the increase in global average temperature well below 2°C above pre-industrial levels" [1]. In this context, the European Commission has taken a central role in driving ESG finance through its European Green Deal, which seeks to transform the EU into a modern, resource-efficient, and competitive economy. Key objectives include becoming the first climate-neutral continent by 2050 and reducing net greenhouse gas emissions by at least 55% by 2030 compared to 1990 levels Commission [2019]. As part of these efforts, large firms are increasingly required to disclose their environmental impacts resulting from their economic activities. In Europe, since 2014, the NFRD (Non-Financial Reporting Directive) directive requires European companies with more than 500 employees to publish reports on their policies regarding environmental protection, social responsibility and the treatment of employees, respect for human rights, the fight against corruption, and diversity on boards of directors. The NFRD will be replaced by the CSRD (Corporate Sustainability Reporting Directive) in 2025, to become Europe's leading sustainability reporting standard. This new Directive is driven by a desire to harmonize the publication of information on corporate sustainability, and bring it up to the same level of robustness as financial information, so as to provide financial companies, investors and the general public with a reliable overview of ESG performance and the risks to which companies are exposed.

As a result, the ESG finance sector has experienced significant growth, both because of the attention given to climate issues and because of the various regulations put in place at international level requiring the largest companies to disclose this information to their investors.

In the next section, we'll identify the main players in this market, try to understand how it works, and what its main characteristics are.

## 2.2 ESG market overview : an oligopolistic market

So-called extra-financial rating agencies have emerged, assessing companies according to ESG criteria, either through data providers or by directly collecting the data required for their analysis. On an international scale, the acquisition of ESG data is generating increasing expenditure, amounting to more than a billion dollars a year (Demartini [2020]).

[1] The United Nations Framework Convention on Climate Change (UNFCCC) provides a short summary of the Paris agreement: https://unfccc.int/process-and-meetings/the-paris-agreement

Demartini [2020], in her mapping of the extra-financial data supply business carried out for the Autorité des marchés financiers (AMF), distinguishes three types of activity: the supply of raw data, the supply of transformed data and the provision of related services.

The ESG data market is characterized by an oligopolistic structure, with a small number of major providers (from US or EU) dominating the market. This has several implications for the quality and diversity of ESG data available to investors and other stakeholders.


Figure 1: Overview of the main ESG rating agencies

Source : Autissier C., Note : Les acteurs et enjeux de la notation ESG, Observatoire de la responsabilité sociale des entreprises (ORSE), Février 2022

➤ ***Dominance of major ESG data providers***

Today's ESG rating market is highly concentrated. This is due to a succession of cross-border consolidation moves (mergers & acquisitions), illustrating strategic external growth choices. The entities absorbed often have the status of subsidiaries, i.e. they retain their legal personality despite the takeover, enabling the entity behind the operation to concentrate all its ESG rating activities within a specific structure.

Major ESG data providers, such as MSCI, Sustainalytics, and Refinitiv, have significant influence over the market due to their established reputations, extensive data offerings, and large customer bases.

This dominance can lead to a lack of competition, limiting the variety of ESG data products and potentially stifling innovation in the sector or conflicts of interest.

➤ ***Barriers to entry***

There are significant barriers to entry in the ESG data market, including the high costs associated with data collection, processing, and analysis, as well as the challenge of establishing credibility and trust among potential customers. This further exacerbates the dominance of major providers and limits the diversity of ESG data available. In order to exist, smaller firms are more likely to specialize in a particular segment or indicator.

Fintechs specializing in ESG data production are flourishing. Despite the small size of these companies, they are proving to be competitive, particularly by specializing in a specific market segment (business sector, climate issues, category of player or asset). For example, Iceberg Datalab, a French fintech that develops evaluation tools and provides environmental data solutions for financial institutions, stands out for its specialization in measuring the impact of financial institutions on biodiversity (Corporate Biodiversity Footprint - CBF). Thanks to this expertise, it was selected in 2020 by Axa Investment Management, BNP Paribas Asset Management, Mirova and Sycomore Asset Management to "provide investors with the first biodiversity impact measurement tool" (link). But despite the existence of these promising fintechs, the market remains highly concentrated, and this could lead to increased dependence on a limited number of players, price inflation and potential conflicts of interest.

The limited competition may result in a lack of innovation and improvements in ESG data methodologies, potentially perpetuating existing issues with data quality and reliability. Additionally, the dominance of a few providers may reduce the diversity of perspectives and approaches in ESG data processing.

In the following section, we will assess the main challenges related to ESG data.

## 2.3 Challenges in ESG data processing and analysis

The evaluation of companies and portfolios using ESG criteria is a complex task. Indeed, as environmental indicators are driven by a multitude of factors including the diversity of data sources, their lack of standardization or inconsistent reporting standards, and the intricate nature of mapping corporate activities to their environmental impact.

Hence, we will try to understand the different sources and formats of the ESG data. Then, we will evaluate the different extraction methods within the ESG firms. Then, we will see that the data sources, formats and the extraction methods lead to data quality and reliability issues.

### 2.3.1 Diversity of data sources and formats

Extra-financial rating agencies and ESG data providers employ a multitude of methodologies to assess a company's performance on ESG criteria. Some utilize qualitative approaches based on interviews and literature reviews, while others prefer quantitative approaches based on Key Performance Indicators (KPIs) and statistical models. Some even combine both approaches. ESG analysis relies on a multi-criteria approach and historical company data.

Traditional approaches to ESG data processing and analysis face numerous challenges. ESG data comes from a variety of sources and can be presented in different formats, adding to the complexity of ESG data processing. Among the different sources, we usually find :

- **Company reports and filings:** Companies provide a wealth of ESG data through their annual reports, sustainability reports, and regulatory filings. However, this information is often presented in different formats, making it difficult to compare and aggregate data across companies.
- **Government databases and publications:** These sources may vary in terms of format, data availability, and data quality, further complicating the ESG data processing landscape.
- **News articles and press releases:** They can provide valuable insights into a company's ESG performance, but the unstructured nature of this data makes it challenging to extract and analyze relevant information systematically.
- **Industry and NGO reports:** Industry associations and non-governmental organizations (NGOs) can provide sector-specific ESG data and insights. However, these sources may also vary in terms of data quality, coverage, and format, adding to the complexity of ESG data processing.

Hence, we see that ESG data comes from a variety of sources and can be presented in different formats, adding to the complexity of ESG data processing. So, the first observation concerning raw ESG data is that it is fundamentally unstructured and is often contained in reports, such as PDF documents.

### 2.3.2 Extraction methods

Traditional extraction methods for ESG data often involve manual processes and are subject to various limitations.

➤ ***Manual extraction by analysts***

ESG analysts extract data from various sources, such as company reports and news articles. Due to the unstructured nature of ESG data, this approach can be time-consuming, labor-intensive, and prone to human error and subjectivity.

➤ ***Keyword-based search and filtering***

Keyword-based search and filtering techniques can help identify relevant information within large datasets. However, this approach can be limited in its ability to capture the nuances and context of ESG data. A small difference in search terms and text content can have a big impact on search results.

➤ ***Spreadsheet-based data organization***

Spreadsheets are commonly used to organize and analyze ESG data, with analysts inputting extracted information into rows and columns for comparison and evaluation. However, this method has several drawbacks, including the potential for human error during data input, difficulty in handling large volumes of data, and limited capacity for dealing with unstructured data sources.

### 2.3.3 Data quality and reliability issues

The quality and reliability of ESG data have been long-standing concerns for investors and other stakeholders. Several factors contribute to these issues, including the lack of standardization in reporting of ESG criteria, data collection process, and quality.

ESG data is complex and multifaceted. It can be quantitative or qualitative, often aggregated from several sources, covering multiple countries and sometimes more than one continent. The sectors of activity are diverse, each with its unique characteristics. This complexity is further compounded by the indicators created from raw data. Rating agencies publish information on their methodology, but this is often incomplete, particularly concerning the variables used for calculation and the weights assigned to them.

Reliable data, which is granular, specific, and quantified, is essential. However, some data, due to their collection process, are erroneous, incomplete, or absent. Indeed, suppliers of extra-financial data use a variety of methods to gather information. These methods include questionnaires sent to companies, use of information published by the entities to which the data relates or by trusted third parties (press agencies, non-governmental organizations), and use of data produced by other suppliers in the sector through subscriptions or partnerships.

One of the significant challenges in dealing with ESG data is the lack of standardization in reporting. Without universally accepted reporting standards for ESG data, inconsistencies arise in the way

companies disclose information. This lack of standardization makes it challenging to compare and aggregate data across organizations and sectors. For instance, in the US, publicly-traded companies must report their financial performance via a 10K report as requested by the SEC. While this is still a report, its standardization makes it easier to extract automatically the information contained in those reports. For ESG reporting, such standardization is still not implemented.

Another issue is ensuring the timeliness and completeness of ESG data. Companies may report ESG information at different times and with varying levels of detail. This inconsistency can result in gaps in data and difficulties in assessing the most recent and relevant information. Therefore, it is crucial to establish a standardized timeline and criteria for ESG data reporting to ensure its reliability and usefulness.

### 2.3.4 Limitations of traditional approaches

The traditional methods of processing ESG data present several challenges that can impede the effective analysis and interpretation of the data.

Firstly, these methods are often time-consuming and labor-intensive. The manual extraction and spreadsheet-based organization of ESG data can be a slow and resource-intensive process. This not only limits the efficiency of the data analysis but also hampers its scalability. The need for a more streamlined and automated process is evident to enhance the efficiency of ESG data analysis.

Secondly, the traditional methods are prone to human error and subjectivity. The reliance on manual processes increases the risk of errors and introduces a subjective element in the data processing. This can potentially affect the accuracy and reliability of the resulting data and insights. Therefore, there is a need for more objective and error-free methods of ESG data processing.

Thirdly, traditional ESG data processing methods often struggle with handling unstructured data. Sources such as data extracted from PDF reports or news articles and press releases pose a challenge to these methods, limiting their ability to capture a comprehensive view of a company's ESG performance. The need for methods that can effectively handle and analyze unstructured data is therefore crucial.

Lastly, the divergence of extra-financial ratings presents a significant challenge. A study by Berg et al. [2022] documents the rating divergence and maps the different methodologies onto a common taxonomy of categories[2]. The divergence is decomposed into contributions of scope, measurement, and weight, with measurement contributing 56% of the divergence, scope 38%, and weight 6%. The

[2]The authors based their data from leading ESG rating agencies such as MSCI, S&P Global, Moody's ESG, Sustainalytics and Refinitiv

study further reveals a rater effect where a rater's overall view of a firm influences the measurement of specific categories. This divergence has led to difficulties in comparing data across companies, industries, and regions, and poses challenges in ensuring the accuracy and reliability of ESG data. The need for a more standardized and consistent rating system is therefore evident.

To conclude on this Section 2, we have seen that traditional approaches to ESG data processing face several challenges, such as data quality and reliability issues, limited extraction methods, and an oligopolistic market structure. These factors hinder the efficient processing and analysis of ESG data, making it difficult for investors and other stakeholders to make informed decisions. In order to overcome these limitations, it is crucial to explore new technologies that can streamline the ESG data processing pipeline and enhance the accuracy and reliability of ESG analysis. This is where artificial intelligence (AI), natural language processing (NLP), and semantic search come into play.

In Section 3, we will delve into the foundations of AI, NLP, and semantic search, exploring how these technologies can be harnessed to transform the ESG data processing landscape. By understanding the underlying concepts and methodologies, we can better appreciate how deep learning innovations, such as large language models (LLMs), can change and improve the way ESG data is processed, analyzed, and utilized.

# 3 Foundations: AI, NLP, and Semantic Search

After looking at the specifics of the ESG finance market and the characteristics of ESG data, we're going to define AI, NLP and semantic search. This will enable us to define some of the building blocks that are essential for some of the applications we'll be looking at next. The most important of these is an understanding of embeddings, which are used today in an extremely wide range of applications, as well as in Transformer models, which we'll be looking at in part 4.

## 3.1 Overview of AI and NLP

### 3.1.1 Definitions and key concepts

Artificial Intelligence (AI) refers to the field of computer science that aims to imbue machines with the capability to perform tasks that typically require human intelligence. This involves the challenge of translating informal, intuitive knowledge into a formal, computable format. As described in Goodfellow et al. [2016], AI applications often utilize probability theory in two significant ways. Firstly, it guides the design of algorithms that compute or approximate expressions derived using probability theory, essentially dictating how AI systems should reason. Secondly, probability and statistics are used to theoretically analyze the behavior of proposed AI systems. The ultimate test for AI lies in solving tasks that are easy for humans but difficult to formally describe, such as recognizing spoken words or faces in images.

Natural Language Processing (NLP) is a subfield of AI that focuses on enabling computers to understand, interpret, and generate human language. It focuses on the interaction between computers and humans through natural language. The objective of NLP is to read, decipher, understand, and make sense of the human language in a valuable way.

A Language Model (LM) is a type of artificial intelligence system that is designed to predict the next word in a sequence of words [3]. This is a fundamental concept in natural language processing (NLP). In the context of a language model, this involves the creation of statistical and probabilistic systems that can understand language based on a specific set of rules or algorithms. These models are typically trained on large amounts of text data, allowing them to learn the probability of a word given its context within a sentence. This predictive capability is what enables applications like text translation, text generation, auto-correct, and speech recognition.

[3] A model whose aim is to predict future words in a sequence based on the words that have come before it, is also called an Auto-Regressive model, taking its inspiration from Time Series models such as AR/MA/ARIMA models, which are auto-regressive models applied to time series.

### 3.1.2 Evolution and advancements in AI and NLP research

The evolution of AI, NLP, and LLM research has been marked by continuous advancements in algorithms, computational power, and the availability of large-scale datasets. From early rule-based systems to modern deep learning models, these fields have undergone significant transformations, enabling increasingly sophisticated language understanding and generation capabilities.

State-of-the-art techniques like GPT-4 and BERT have revolutionized NLP by leveraging deep learning, large-scale training data, and advanced model architectures. These models demonstrate exceptional performance in various NLP tasks, including text generation, text classification, sentiment analysis, and question answering. These models will be studied in detail in part 4.

## 3.2 Embeddings and Semantic Search

In the following sections, we'll describe the advances in NLP with more details, starting with the word2vec innovation that led to the creation of embeddings, an especially important method for transforming a word or phrase into a vector, and then describing what these embeddings can achieve via semantic search and similarity search.

### 3.2.1 Word embeddings: Word2Vec

Word embeddings are vector representations of words that capture semantic meaning, enabling efficient comparison and analysis of textual data. Advances in embedding techniques have improved the accuracy and interpretability of these representations, contributing to more effective semantic search and analysis.

The "Efficient Estimation of Word Representations in Vector Space" article, authored by Tomas Mikolov and his team at Google, introduces Word2Vec (Mikolov et al. [2013a]), which was a widely-used technique for learning continuous word embeddings in an unsupervised manner. Word2Vec represents words as high-dimensional vectors, capturing semantic and syntactic relationships between them.

The paper presents two novel model architectures for learning word embeddings:

1. **Continuous Bag-of-Words (CBOW)**: This model predicts a target word based on the context words surrounding it. The context words are averaged, and the model tries to minimize the difference between the predicted target word and the actual target word.

2. **Skip-gram**: This model reverses the CBOW approach, predicting context words given a target word. It tries to maximize the probability of context words appearing around the target word.

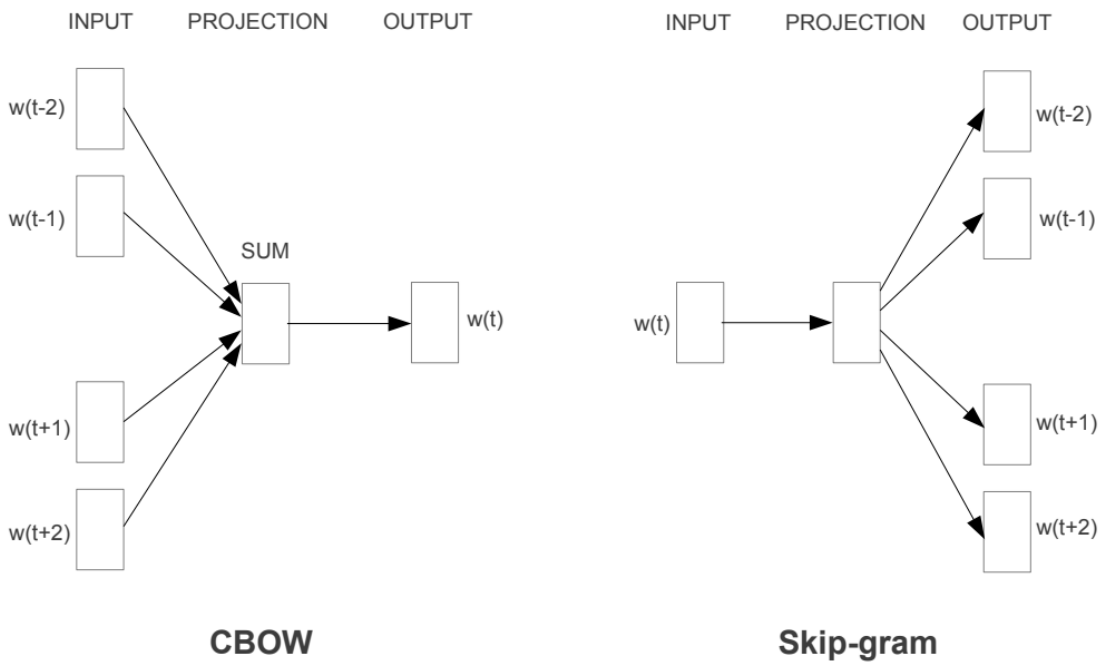



Figure 2: Graphical representation of the CBOW model and Skip-gram model. In the CBOW model, the distributed representations of context (or surrounding words) are combined to predict the word in the middle. In the Skip-gram model, the distributed representation of the input word is used to predict the context

Source : Mikolov et al. [2013b]

Using those models, we can then find a word embedding. It is a learned representation for text where words that have the same meaning have a similar representation in a vector space. It allows to capture semantic and syntactic relationships between words and address limitations of one-hot encoding and count-based methods like term frequency-inverse document frequency (TF-IDF).

More generally, the embedding method, which allows words or phrases to be represented in a vector space, has a number of advantages. These include *semantic relationships* (words with similar meanings will have a greater probability of being located close to each other in the vector space generated by the model), *transfer learning* (training a model for embeddings may be resource-intensive, but it can then be reused for other tasks), *dimension reduction* (since each word/phrase is now a vector, common machine learning techniques can be used), better management of missing words, etc. Moreover, the information contained in embedding vectors avoids the problem of sparse matrices, which contain usally many 0s and few 1s. On the contrary, embedding vectors are dense matrices. It can be said that the text information is in fact compressed into these vectors, and becomes intelligible to machines via the algorithms and models that are developed.

Today's methods for creating embeddings are more complex than the two above-mentioned archi-

tectures, but nevertheless, the advantages described remain the same. Models based on the Transformer architecture (which we'll look at in the next section) are now used to create embeddings. These models can not only create embeddings for words (like the simple method we've just discussed), but also for phrases, making it easy to transform large documents into multiple embeddings.

In practical terms, the two most straightforward methods to obtain word embeddings are through the use of OpenAI Embeddings API. This service is highly cost-effective, offering 3000 pages for a mere dollar as of the current date [4]. While it may not provide the highest quality embeddings, it compensates with its ease of access, multilingual support, and extensive context length.

The selection of an embeddings provider is a critical decision when developing a large-scale application. This is because if a substantial database of embeddings is established and the provider decides to alter their methodology - for instance, by changing the dimension of the vector space - it would necessitate the re-creation of embeddings for each data segment.

Open Source alternatives also exist, such as the Sentence Transformer library [5]. Moreover, a variety of models are accessible through Hugging Face Embeddings Leaderboard [6].

The Application below shows a simple example of how to use Embeddings using the OpenAI API. While this requires obtaining an OpenAI API key, for such queries, the cost is negligible.

Now that we have a good understanding of what an embedding is, we're going to take a brief look at semantic search, which is made possible by the transformation of textual data into a numerical representation, in the form of vectors, using embedding techniques.

[4]For more details, see: https://platform.openai.com/docs/guides/embeddings/what-are-embeddings
[5]For more details, see: https://www.sbert.net/
[6]Hugging Face Embeddings Leaderboard: https://huggingface.co/spaces/mteb/leaderboard

**Application Example using OpenAI Embeddings**

In this application, we begin by importing the necessary packages:

- Pandas, for the creation of the dataframe.
- OpenAI, to establish a connection with their API.

Subsequently, we define a function to invoke OpenAI Embeddings. A list of arbitrary words is defined, each of which is dispatched to OpenAI Embeddings API for vectorization. Consequently, each word is associated with a vector comprising 1536 dimensions.

Following this, we construct a dataframe that encapsulates the words along with their vectorized representations. This dataframe can then be utilized to calculate the distance between words, thereby enabling us to identify words that are most similar to a given word.

**OpenAI Embeddings**

```
import openai
import pandas as pd
```

```
def get_embedding(text, model="text-embedding-ada-002"):
    return openai.Embedding.create(input=[text], model=model)['data'][0]['embedding']
```

```
words = [
    "king", "queen", "male", "female",
    "apple", "banana", "car", "dog", "elephant", "fish", "guitar", "house", "icecream", "jacket",
    "kite", "lion", "mountain", "nurse", "ocean", "piano", "quilt", "river", "sun", "tree",
    "umbrella", "violin", "whale", "xylophone", "yacht", "zebra", "ant", "bear", "cat", "deer",
]

embeddings = [{"word": word, "embedding": get_embedding(word)} for word in words]
```

```
df = pd.DataFrame(embeddings)
df.head(5)
```

| | word | embedding |
|---|---|---|
| 0 | king | [0.010118617676198483, -0.003987479489296675, ... |
| 1 | queen | [-0.004535923711955547, -0.006746180821210146,... |
| 2 | male | [-0.0033002542331814766, -0.018109697848558426... |
| 3 | female | [-0.014222253113985062, -0.009118244051933289,... |
| 4 | apple | [0.0077999732457101345, -0.02301608957350254, ... |

**OpenAI Embeddings have 1536 dimensions**

Figure 3: Computing OpenAI Embeddings using Python

Since these words are now simple vectors, we can use traditional machine learning techniques. For example, we could use a dimension reduction technique, such as Principal Component Analysis (PCA), and retain only the 10 PCs that explain the highest variance in the data, making it easy to go from 1536 columns to just 10, while retaining a maximum of information. Of course, the number 10 here is arbitrary and must be adapted to the case under study.

### 3.2.2 Semantic search and Embeddings

Semantic search refers to the practice of understanding the context, intent, and semantic meaning of search phrases rather than focusing solely on keyword matching. Unlike traditional search algorithms that only focus on finding exact matches of the search terms, semantic search algorithms look to understand the searcher's intent and the contextual meaning of the terms to generate more relevant results. Semantic search is a vast concept, and many different techniques exist but we will focus on semantic search related to word embeddings.

Embeddings, like Word2Vec (Mikolov et al. [2013a]), GloVe (Pennington et al. [2014], or BERT (Devlin et al. [2019]), play a critical role in semantic search by providing a numerical representation of words, phrases, or documents that captures their meaning and relationships. By representing queries and documents in the same semantic space, it becomes possible to measure their similarity and relevance more accurately than with traditional keyword-based approaches.

Semantic search can leverage word embeddings to identify and retrieve information based on the underlying meaning of queries, rather than relying on simple keyword matching. This enables more accurate and relevant search results, particularly for complex or ambiguous queries.

In the context of vector search, similarity measures are a function that takes two embedding vectors as input and calculates a distance value between them. We use the calculated distance to judge how close or far apart two vector embeddings are. Many metrics exists to perform such operations :

➤ ***Cosine Similarity***

Cosine similarity measures the cosine of the angle between two vectors. The smaller the angle, the more similar they are. This measurement is particularly useful for embeddings because it is not affected by the magnitude of the vectors, but only their orientation in the embedding space. Cosine similarity ranges between -1 and 1, where 1 means the vectors are identical, 0 means the vectors are orthogonal (i.e., not similar), and -1 means the vectors are diametrically opposed (i.e., opposite).

$$\text{Cosine similarity}(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} = cos(\theta)$$

Where $A$ and $B$ are embedding vectors for a given word (or phrase), $||A||$ the norm of the embedding vector $A$ and $\theta$, the angle between two embedding vector.

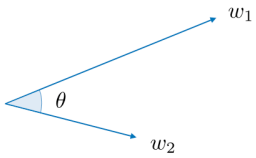



Figure 4: Cosine Similarity

➤ ***Manhattan Distance***

The Manhattan distance, also known as the L1 norm, between two vectors is the sum of the absolute differences of their coordinates. Unlike Euclidean distance, this does not involve any square or square root operations, making it computationally cheaper.

$$\text{Manhattan distance}(A, B) = \sum_{i=1}^{n} |A_i - B_i|$$

➤ ***Euclidean Distance***

The Euclidean Distance, also known as the L2 norm, between two vectors, is the sum of the root mean squared difference between two vectors. The distance can be any value between zero and infinity. If the distance is zero, the vectors are identical. The larger the distance, the farther apart the vectors are.

$$\text{Euclidean distance}(A, B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

In practice, Cosine Similarity is often the most used one when dealing with text contents, and the recommended one by OpenAI [7].

| **Application Example** |
| --- |
| Perform a similarity search directly from python notebook |

### 3.2.3 Improved retrieval and analysis of unstructured data

Sentence embeddings and semantic search techniques have significantly enhanced the ability to retrieve and analyze unstructured data, such as news articles, press releases or documents stored in

[7] See: https://platform.openai.com/docs/guides/embeddings/what-are-embeddings

PDFs. By understanding the underlying meaning of text, these technologies can identify relevant information and insights more efficiently and accurately than traditional methods.

These new information retrieval techniques have led to the development of a new category of database called the Vector Database. They have benefited greatly from the ChatGPT craze, and many firms have been able to raise substantial funds. These include Pinecone [8], Weaviate [9] and many others.

These vector databases are particularly important today, as they allow us to overcome the problem of the context that can be given to a Transformer model, which, as of today often ranges from 4,000 to 16,000 tokens. As it's impossible to give them more than this context, the most popular solution today, especially when you want to give a model information contained in a text document (PDF for example), is to chunk the document, vectorize the text to obtain embeddings, compare the similarity of the question with all the embeddings of every chunk in the document, and give the text most similar to the question to the model. In part 6, we'll show this use case and how it can be implemented.

Taking Weaviate as an example, it's easy to quickly switch vectorizers (from OpenAI Embeddings to an Open Source model for transforming text into embeddings). Those solutions also facilitate the scaling to hundreds of millions of embedding vectors. The algorithms available for performing similarity searches are more powerful, thanks in particular to the use of Hierarchical Navigable Small World (HNSW, Malkov and Yashunin [2018]) when indexing vector embeddings. [10] This makes similarity searches both fast and memory-efficient using an approximate nearest neighbors search. Plus, tools like Weaviate allows for different types of search (BM25 or hybrid search which combines semantic search and keyword search [11])

It's also important to note that these embedding databases are not limited to text, but also work with images, and text within images.

[8] See: https://sg.finance.yahoo.com/news/pinecone-drops-100m-investment-750m-151248557.html

[9] See: https://www.prnewswire.com/news-releases/weaviate-raises-50-million-series-b-funding-to-meet-soaring-demand-for-ai-native-vector-database-technology-301803296.html

[10] Both Pinecone and Weaviate use HNSW algorithm under the hood.

[11] Stackoverflow in their blog explains how they use Weaviate in its stack to perform hybrid search.

## 3.3 Recurrent Neural Networks and LSTMs

In this section we'll look at some language models, such as RNN or LSTM, which were popular models before 2017, and the beginnings of transformer models. Today, the most widely used and best-performing models are, for the most part, model which inherit from the transformers architecture. Because these models have a very large number of parameters (over 170B for ChatGPT, for example), we call them Large Language Models (LLMs). They are state-of-the-art models like GPT-4 demonstrating impressive performance in various NLP tasks.

Before the advent of model transformers, RNN and LSTM were considered the state of the art for NLP tasks. A RNN can be used as a Language Model, ie, at predicting the next word, but it can also do more. So we'll briefly describe how they work, in order to understand their strengths and weaknesses, and this should help us better understand what problems the model transformers have helped to solve.

### 3.3.1 Understanding Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a specialized type of artificial neural networks designed to handle sequential data. This could be time series data, natural language, or any data with temporal or sequential dependencies. RNNs are unique in their ability to use previous outputs as inputs while maintaining hidden states.

The "internal state" of an RNN is updated as it processes a sequence, making it particularly effective at modeling problems where the input and/or output is a sequence. This is due to its inherent ability to maintain a hidden state that can store information from previous time steps and use it as new input. RNNs form the foundation for more complex models such as Long Short-Term Memory (LSTM) networks.

In Natural Language Processing (NLP), RNNs are used for a variety of tasks including part-of-speech tagging, named entity recognition, sentiment classification, question answering, machine translation, speech recognition, summarization, and text generation.

### 3.3.2 Working of RNNs

An RNN is composed of three primary components: an input layer, a hidden layer, and an output layer. The hidden layer maintains a hidden state (a vector), which is updated at each time step in the sequence.

RNNs are trained using a variant of backpropagation known as Backpropagation through Time

(BPTT). BPTT unfolds the RNN for a specific number of time steps and calculates the gradients for each time step with respect to the loss function. These gradients are then used to update the weight matrices ($W_xh$, $W_hh$, $W_hy$) and bias terms ($b_h$, $b_y$) using an optimization algorithm like stochastic gradient descent (SGD) or Adam.

RNNs are advantageous as they can process input sequences of any length without increasing the model size as the input sequence length increases. They can also use past information to predict future elements in the sequence.

### 3.3.3 Challenges with RNNs

Despite their advantages, RNNs face several challenges. They can be slow to train and infer because the data is processed *sequentially*, preventing parallelization. While they can theoretically capture long-term dependencies, in reality, due to the *vanishing* or *exploding gradients problem* they struggle with *long-term dependencies* and thus, have difficulty understanding context.

While basic RNNs only consider past words to predict the next word, other variants like Bidirectional RNNs consider both past and future words. However, these models are more memory-intensive as they require twice the number of hidden layers.

### 3.3.4 Introduction to LSTMs

We saw that Recurrent Neural Networks (RNNs) are a type of artificial neural network designed to recognize patterns in sequences of data, such as text, genomes, handwriting, or the spoken word. They are extremely powerful and versatile, but they also have some limitations. One of the main issues with RNNs is the problem of long-term dependencies, where the network needs to remember information for a long period of time in order to make accurate predictions. This is where Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) come into play.

LSTM is a type of RNN architecture that was designed to overcome the limitations of traditional RNNs. LSTM introduces a new structure called a *memory cell* which essentially acts as an information highway, carrying the relevant information across long distances thereby mitigating the problem of vanishing gradients, a common issue with standard RNNs. This memory cell is controlled by various gates, which are responsible for the flow of information inside the LSTM unit. These gates decide what information should be kept or discarded at each time step, allowing the model to learn long-term dependencies.

On the other hand, GRU is a variation of LSTM designed to be more efficient. It combines the forget and input gates of an LSTM into a single "update gate" and also merges the cell state and hidden state.

This results in a simpler and more efficient model that often performs comparably to LSTM on certain tasks, but with less computational complexity.

In 2016, the de facto strategy in Natural Language Processing (NLP) was to encode sentences with a bidirectional Long Short-Term Memory (LSTM). This approach was particularly effective in tasks such as translating a source sentence into a target language. The bidirectional LSTM, a type of recurrent neural network, was designed to effectively capture the contextual information from both the past (backward direction) and the future (forward direction), making it a powerful tool for encoding sentences, as it could understand the semantic meaning of a sentence in a more comprehensive way than its unidirectional counterpart.

The LSTM's ability to remember and retrieve information over long periods made it an ideal choice for sequence generation tasks. It could effectively model the sequential dependencies in the data, making it capable of generating coherent and contextually relevant sequences.

One way to improve RNN and LSTM was via the use of *Attention Mechanisms*. Hence, we will delve on the topic, as it is crucial in explaining the rise of Large Language model.

### 3.4 Attention mechanisms

The attention mechanism was first elaborated in "Neural Machine Translation by Jointly Learning to Align and Translate" by Bahdanau et al. [2014]. This mechanism has had a major impact in the field of translation and, more broadly, in various NLP tasks, and more generally for sequence-to-sequence models, as it forms the core of model transformers that we will see in the next part. This mechanism allows models to focus on specific parts of the input when producing an output, much like how humans pay attention to certain details when understanding a concept or performing a task.

In the context of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, the attention mechanism is used to weigh the importance of different inputs at each time step. Traditional RNNs and LSTMs process inputs sequentially, which can lead to issues with long sequences due to the vanishing or exploding gradient problem. The attention mechanism helps to mitigate this by allowing the model to focus on the most relevant parts of the input sequence instead of having to process the entire sequence. This is particularly useful in tasks such as machine translation, where the relevance of a word in the source sentence can vary depending on the context.

The attention mechanism works by computing a set of attention weights for each input. These weights determine how much attention should be paid to each input when producing the output. The weights are computed using a function of the inputs and the current state of the model. The output

is then a weighted sum of the inputs, with the weights determined by the attention mechanism.

Attention allowed the model to have flexible access to its memory, enabling it to focus on the most relevant parts of the input sequence when generating the output sequence. This was particularly useful in tasks such as machine translation, where certain words in the source sentence had a stronger influence on the translation than others. By allowing the model to pay 'attention' to these influential words, the quality of the generated translations was significantly improved.

In summary, while RNNs are a powerful tool for processing sequential data, they struggle with long-term dependencies due to the vanishing gradient problem. LSTM and GRU are both variations of RNNs that address this issue, with LSTM using a more complex structure with multiple gates and a separate memory cell to control the flow of information, and GRU simplifying this structure for increased efficiency [12]. RNN and LSTM models can also be used with Attention mechanisms to improve the performance of the models [13]. However, their difficulty in learning long-term dependencies and their sequential processing of texts prevented the parallelization of tasks. As we'll see in the next section, Transformer models, which is the first model to based all its architecture on the Attention mechanism, will help solve these problems, leading to substantial improvements in these models and becoming state-of-the-art NLP models.

[12] For more informations about RNN, GRU and LSTM, see : Stanford Course I and Stanford Course II

[13] To better understand how attention mechanism is used in conjunction of RNN and LSTM, see: Stanford Course III

# 4 Deep Learning Innovations: The Path to LLMs

The impact of Transformers models in the field of Deep Learning and more specifically in the NLP field has been extremely significant. We can clearly say that there was a before and an after to the release of the article, *Attention is all you need* (Vaswani et al. [2017]), which first developed this new architecture.

This architecture is now widely used, and is responsible for improved performance and the state of the art performance in tasks like machine translation, text summarization, sentiment analysis, question-answering and, more broadly, for the development of what is now known as generative AI.

OpenAI models, such as ChatGPT, are based on this architecture. Therefore, it is essential to take a closer look at how the Transformer architecture works, in order to better understand how these models are able to achieve feats that previously seemed unattainable, but also to understand their limitations.

## 4.1 The Transformer Architecture

A Transformer model is a deep learning architecture introduced by Vaswani et al. [2017] that focuses on attention mechanisms to handle sequence-to-sequence tasks, such as natural language processing, more effectively. It replaces recurrent and convolutional layers (RNN or CNN) with self-attention and positional encoding mechanisms to process input data in *parallel* rather than *sequentially*, which reduces computation time, allows for longer-range dependencies and for larger training datasets.

The main idea behind this attention mechanism is to enable the model, when it wants to predict the next word, to choose from the input sequence where to focus its attention. This mechanism allows the model to handle long sequences and capture long-distance dependencies, overcoming the limitations of the traditional encoder-decoder architecture. As we saw previously, the attention mechanism can also used in RNN, but the main contribution of Transformer models is precisely to based the whole architecture model on attention mechanism.

### 4.1.1 Reasons for transformer success

Before we delve deeper into the Transform model, and the infrastructure that makes it so special, let's start by highlighting the key points that have enabled it to outperform all previous models.

➤ ***Self-Supervised***

A language model usually does not need labels for its pretraining. The pretraining is usually self-supervised, which means the labels are created automatically from the inputs (like predicting the next word or filling in some masked words). This allows to train the models on very large datasets.

This is important, because it theoretically allows to train a model on an extremely large amount of data, potentially the whole Internet. In the case of Meta's LLama 2[14], a Large Language Model, the models have been trained on over 2 trillion tokens [15].

➤ ***Handling long-range dependencies***

Traditional sequential models, like RNNs and LSTMs, have difficulties in capturing long-range dependencies due to the vanishing gradient problem. Transformer models, with their attention mechanisms, can process long-range dependencies more effectively, allowing them to learn complex patterns and contextual information.

➤ ***Parallelization***

The parallelization of tasks is essential in the success of these models because it is this that ultimately allows training these models on huge amounts of data, while not taking an infinite amount of time. Parallelization also allows the use of a very large number of graphics cards during the pre-training phase. For example, a cluster of more than 2000 graphical cards A100 80 GB from Nvidia [16] were used to train the LLama 2 model.

➤ ***Transfer learning***

Transformer models, like BERT and GPT, have popularized transfer learning and unsupervised pre-training in the AI field. By pre-training models on massive amounts of data, they can learn general language representations that can be fine-tuned for specific tasks with smaller labeled datasets. This approach has improved performance in various tasks and enabled the development of AI solutions for low-resource languages and domains.

Models are generally trained for general tasks, such as simply predicting the next word. They are then fine-tuned to perform more precise tasks. The Llama 2 model developed by Meta is an interesting example of how a foundation model can be trained to perform more precise tasks, in this case for a chatbot. The model was first trained on 2 trillion tokens, then fine-tuned on almost 100,000 chat examples and 1,000,000 million human comments. This then enables it to be an effective chatbot.

[14]Llama stands for: Large Language Model Meta AI

[15]On July 18, 2023, Llama 2 was released. This pretrained models are trained on 2 trillion tokens, and have double the context length than Llama 1. For the full report, see: Link

[16]Each A100 GPU with 80 GB of VRAM cost roughly $15 000, which cost approximately 30 million dollars.

### 4.1.2 Introduction to the Transformer architecture

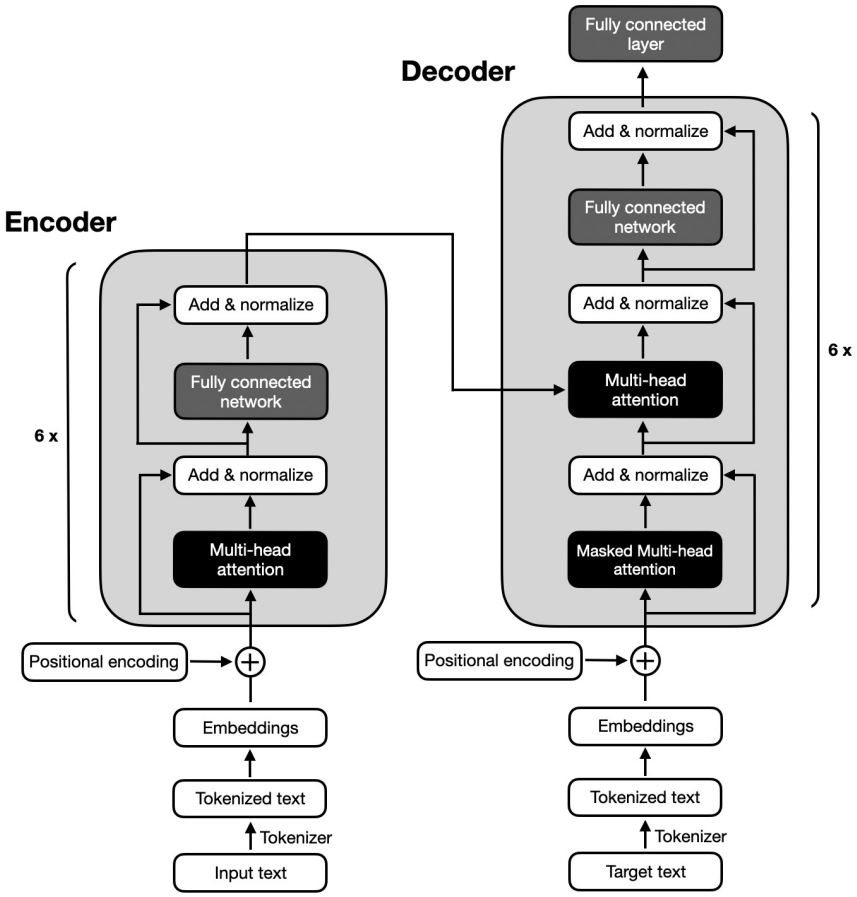



Figure 5: Transformer architecture for the traditional encoder-decoder model from "Attention is all you need". The encoder part is on the left, while the decoder is on the right.

Source : Vaswani et al. [2017]

In this section, we'll look at the main components of the Transformer architecture. We won't go into all the details, and certain parts essential to its implementation will not be mentioned, for the sake of simplicity and to focus on the main elements that may be useful in understanding certain problems that will be raised later on. Indeed, it will allow us to understand the different optimization than can be done to improve the model.

Here are the main stages in a model transformer :

1. **Tokenization**: This is the first step in the Transformer model. The process involves breaking down the input text into smaller units, called tokens. These tokens can be as small as individual characters, or as large as words. The choice of token size depends on the specific application and the language of the input text.

2. **Embedding**: Once the text is tokenized, each token is then mapped to a vector in a high-dimensional space. This process is called embedding. These vectors capture the semantic meaning of the tokens. The essential thing to note here, is that the embedding for a given token, at this moment, is non-contextual. Meaning, the embedding is independent of the context. The goal will be to develop strong context via Positional encoding and the transformer block.

3. **Positional encoding**: Adds order to the words in the text. It is designed to help the model learn some notion of sequences and relative positioning of tokens. A same word thus, can have a slightly different embedding, depending on its position in the sequence. Without positional encoding, a transformer model would not be able to distinguish between "the oven cooked the bread" ("le four a cuit le pain") and "the bread cooked the oven" ("le pain a cuit le four").

4. **Transformer block**: The core of the Transformer model is the Transformer block, which consists of two main components: the self-attention mechanism and the feed-forward neural network.

   (a) **Self-Attention Mechanism**: It is the main architectural component of the model. The goal here is to add context to the embedding vector output by enabling the model to make use of the surrounding words to learn contextual information for predicting the next token. This mechanism allows the model to weigh the importance of each token in the sequence when processing a particular token. In other words, it allows the model to focus on different parts of the input sequence when processing each token, hence the term "attention".

   (b) **Feed-Forward Neural Network**: After the self-attention mechanism, each token's output is then passed through a feed-forward neural network, which is the same for each position. This network has two layers and is used to transform the output of the self-attention mechanism. The feed-forward neural network then processes each token independently, allowing for parallel computation

5. **Output**: The output of each Transformer block is then used as the input to the next block. This process is repeated for as many blocks as there are in the model. The final output of the last Transformer block is then used for the task at hand, whether it be classification, translation, or some other task. The output of the Transformer model is a sequence of vectors, each corresponding to a token in the input sequence. These vectors are then passed through a final linear layer followed by a softmax function, which converts them into probabilities. The token with the highest probability is chosen as the prediction for the next token in the sequence.

6. **Temperature**: The temperature is a parameter in the softmax function that controls the ran-

domness of the model's predictions. Language models do not output an actual token. Rather, they output a probability distribution over all possible tokens that are known by the model. A higher temperature will make the probabilities more uniform, leading to more random outputs. A lower temperature will make the probabilities more extreme, leading to more deterministic outputs. Thus, the temperature simply modifies the distribution according to which we sample tokens

7. Technical stages :

   (a) **Layer Normalization**: Layer normalization is a step that is performed at two stages within each Transformer block: once after the self-attention mechanism and once after the feed-forward neural network. This process helps stabilize the model's learning process and allows it to train more effectively.

   (b) **Residual Connections**: Another important feature of the Transformer architecture is the use of residual connections. These are direct shortcuts from the input of each Transformer block to its output, which are added to the output of each sub-layer (self-attention and feed-forward neural network). This helps to mitigate the problem of vanishing gradients, which can occur in deep neural networks.

In the original paper, the encoder (left part) is repeated 6 times, and the output is then given to the decoder, which itself is also repeated 6 times, and at each repetition, uses the final encoder output.

#### 4.1.3 Self-Attention

The Attention mechanism is the main architectural component of a transformer model. It allows the model to make use of contextual information contained within the input sequence.

However, in Transformer models, the attention mechanism is used in a slightly different way than what we saw previously with the RNN and LSTM. Instead of processing inputs sequentially, Transformer models process all inputs in parallel. This is achieved through the use of *self-attention*, a variant of the attention mechanism that allows the model to consider all parts of the input when producing each part of the output. This makes Transformer models particularly effective at tasks that require understanding the context of a sentence, such as language translation and text summarization.

In a Transformer model, the self-attention mechanism computes a set of attention weights for each input in relation to every other input. This allows the model to consider the context of each word in the sentence when producing the output. The weights are computed using a function of the inputs, and the output is a weighted sum of the inputs, with the weights determined by the self-attention

mechanism. This allows the model to focus on the most relevant parts of the input when producing each part of the output, leading to more accurate and contextually aware results.

There are many forms of self-attention. The most famous one is called key-query-value self-attention [17]. In practice, self attention is not used, but instead, multi-head self-attention [18].

## 4.2 Large Language Models (LLMs)

So far, we've studied the Transformer model as described in the original paper. However, one of the strengths of the Transformer architecture is that it allows a wide variety of different models. Yang et al. [2023a] highlight the existence of three main categories of LLM :

1. Encoder-Decoder
2. Encoder only (most popular after the release of "Attention is all you need)
3. Decoder only (the most popular since the introduction of GPT3)

The graph below shows the main models that have been developed since 2018. On the left, in pink, are models that only use the encoder in the transformer architecture. Its main representative is the BERT model developed by Google, which has spawned a large number of variations. In green, in the middle, are encoder-decoders. Finally, on the right, we find the decoder-only models. As we see, this method was first developed by OpenAI via its GPT model and is now, by far, the most popular and most known as of today.

[17] The LLama 2 model uses grouped-query attention developed by Google researchers (Ainslie et al. [2023]), which allows for faster inference.

[18] For more information about self-attention, multi-head-attention and layer normalization, we can refer to these Stanford course (Link)

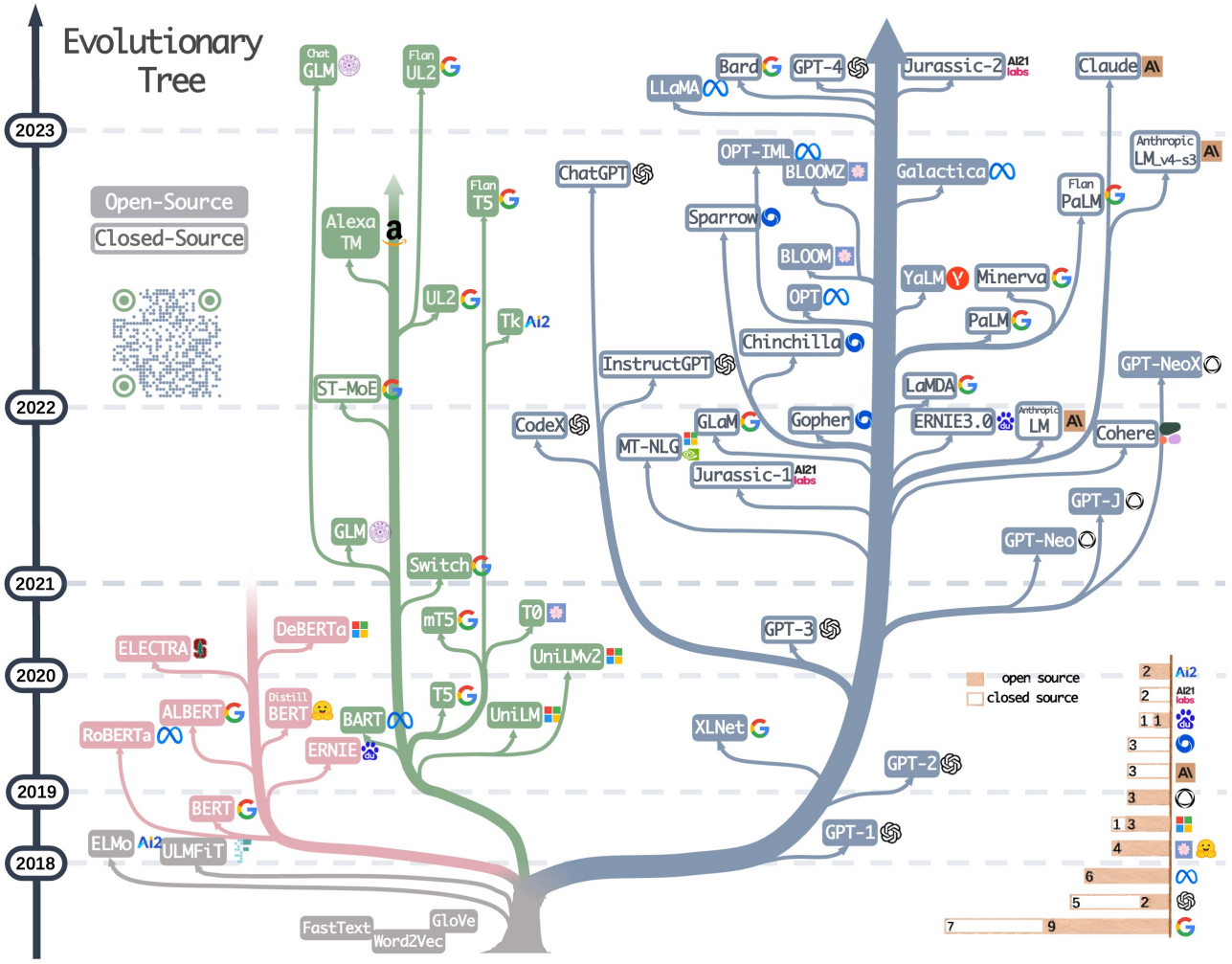



Figure 6: The different model tranformers according to their architecture

Source : Yang et al. [2023a]

First, we'll briefly explain the main differences between encoder and decoder only architecture. Then, we will introduce the BERT model, emblematic of encoder-based models. Next, we'll look at the different GPT models. We'll then see that the decoder models based on this infrastructure are the most powerful today, but that other factors are essential to improving the performance of these models, notably the data passed to the model. This will enable us to put forward another approach, more focused on improving the data, rather only trying to improve the model per se. The power of today's graphics cards is also an important factor in the success of these models, enabling them to grow in size. We'll be looking at the benefits of using a GPU versus a CPU in the context of deep learning and LLMs in particular. Finally, to conclude this section, we will explore the limitations of LLMs.

### 4.2.1 Differences between Encoder and Decoder

At their core, both encoder and decoder architectures utilize self-attention layers to encode word tokens. Encoders are designed to learn embeddings for predictive modeling tasks like classification, while decoders generate new texts, such as answering queries. Decoders are autoregressive model. They generate output sequences one token at a time, conditioning each token on the previously generated tokens.

In the original Transformer architecture (an encoder-decoder model), the decoder mirrors the encoder's structure but differs in inputs and outputs. The encoder processes the input text, extracts relevant information, and outputs an embedding of the input, which the decoder uses to generate the translated text. The decoder's multi-head self-attention mechanism mirrors the encoder's but includes *masking* to prevent the model from attending to future positions. This ensures predictions for the token at the position $i$ depends only on known outputs at positions that are before this token, maintaining the transformer model's autoregressive property during training and inference.

### 4.2.2 BERT: Bidirectional Encoder Representations from Transformers

BERT is a pre-trained Transformer model introduced by Google in 2018 that is designed for natural language understanding tasks. It is based on the Transformer architecture, and use only the *encoder* part. The model is trained on a large corpus of text using unsupervised learning with the goal of learning a general language representation that can be fine-tuned for specific tasks.

There are two key techniques used in BERT:

1. **Bidirectional Context**: Unlike some other models that only process text in one direction (left-to-right or right-to-left), BERT processes text bidirectionally, allowing it to capture a more accurate understanding of the context of words in a sentence.

2. **Masked Language Model (MLM)**: During pre-training, BERT uses a technique called MLM, where it randomly masks some words in a sentence and then tries to predict those masked words based on the surrounding context. This helps BERT learn better representations of words and their meanings.

After pre-training, BERT can be fine-tuned on smaller labeled datasets for specific tasks, such as sentiment analysis, question-answering, or NER (Named Entity Recognition). The fine-tuning process involves training additional layers on top of the pre-trained BERT model to adapt it to the specific task.

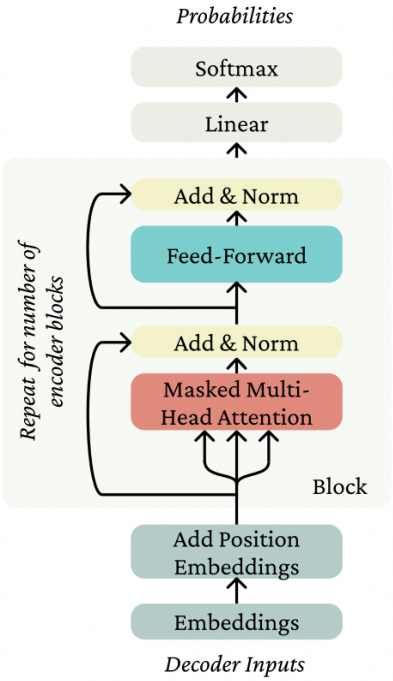
BERT has achieved state-of-the-art performance on various benchmark datasets and has had a

significant impact on the field of natural language processing. It has also inspired the development of numerous variations and adaptations of the original model, such as RoBERTa, DistilBERT, and ALBERT. Nevertheless, decoder-based model successes like GPT models are by far the most popular today.

### 4.2.3 Understanding GPT models: The Evolution of Generative Pre-trained Transformers

The distinction between Transformer Decoders and Encoders lies in the application of future masking in the former during each self-attention process. This ensures an informational constraint, preventing the model from predicting future data.

Generative Pre-trained Transformer (GPT) models are notable for their emergent properties, abilities developed through next-word prediction pretraining. Despite being trained only to predict the next word, these models can perform tasks like text summarization, translation, question answering, and classification. They can also perform new tasks without updating model parameters via *in-context learning*.




*Transformer Decoder*

Figure 7: Transformer Decoder Architecture

We'll quickly go over what's new in each evolution of the GPT models, so as to fully grasp the important developments that have made these models so powerful.

➤ ***GPT 1***

The significance of this paper (Radford et al. [2018]) lies in its demonstration of how unsupervised learning, i.e. training on very large unlabeled dataset, can enhance the performance of language models, making them more precise and versatile for a myriad of tasks. The methodology introduced forms the foundation for the widely used GPT model series.

Moreover, it showed that transfer learning was effective via fine-tuning once the pre-training phase was over. At that time, models were still very small. GPT 1 only had 117 million parameters.

➤ ***GPT 2***

The GPT-2 model (Radford et al. [2019] ) is built on the premise that a model adept at predicting the next word in a sentence is also learning crucial information about language structure, context, and even world facts. This training approach has enabled GPT-2 to perform multiple tasks, such as translation, summarization, and question-answering, by merely altering the input format.

GPT-2 model also improved upon its predecessor by increasing the model size (from 117 million parameters to 1.5 billion parameters) and training on a larger dataset.

➤ ***GPT 3***

The GPT-3 model introduces the concept of "few-shot learning," where the model learns to perform new tasks with only a few examples. Instead of fine-tuning the model for each task, GPT-3 can understand and perform tasks by providing a small number of examples in the input, demonstrating how the task should be done. This is made possible due to the extensive text data training of GPT-3, enabling it to learn about language, context, and various tasks. This research is crucial as it highlights the potential of large-scale language models to learn and adapt to new tasks quickly and with minimal additional training.

The GPT-3 model, often referred to as a monolithic entity, is in reality a collection of several models, with the "davinci" model being the most extensive, boasting 175 billion parameters [19]. This model architecture served as the foundation upon which OpenAI developed its renowned Chat GPT application. However, it is crucial to note that despite GPT-3's initial release in 2020, ChatGPT was not launched until 2022, underscoring the extensive development required to significantly enhance the model's capabilities.

A pivotal advancement that substantially improved GPT-3's capabilities was introduced in "Instruct GPT" (Ouyang et al. [2022]) [20]. This paper elucidates the Reinforcement Learning from Hu-

[19] For a comprehensive list of GPT-3 models, we can refer to: https://en.wikipedia.org/wiki/GPT-3

[20] For a brief overview of the topic, refer to: https://openai.com/research/instruction-following

man Feedback (RLHF) method, which enhances the model's instruction-following ability through a reward system that fine-tunes the model. This method has also contributed to the improved safety and reliability of these models, while simultaneously reducing hallucinations.

Decoder-only models, such as GPT, have witnessed a significant surge in model parameters, with a tenfold increase from GPT-1 to GPT-2, and a hundredfold increase to GPT-3. The number of parameters for GPT-4, however, remains undisclosed. Despite this, these models, trained on larger and higher-quality datasets, have demonstrated remarkable performance across a diverse range of tasks, including language translation, text summarization, and question-answering.

In the context of GPT models, it is evident that augmenting the size of the models results in a significant enhancement in performance. However, each new iteration of the model is accompanied by a substantial increase in the volume of data used for training, as well as an improvement in data quality.

In the following section, we will delve into the pivotal role of data, underscoring its significance in the performance of these models.

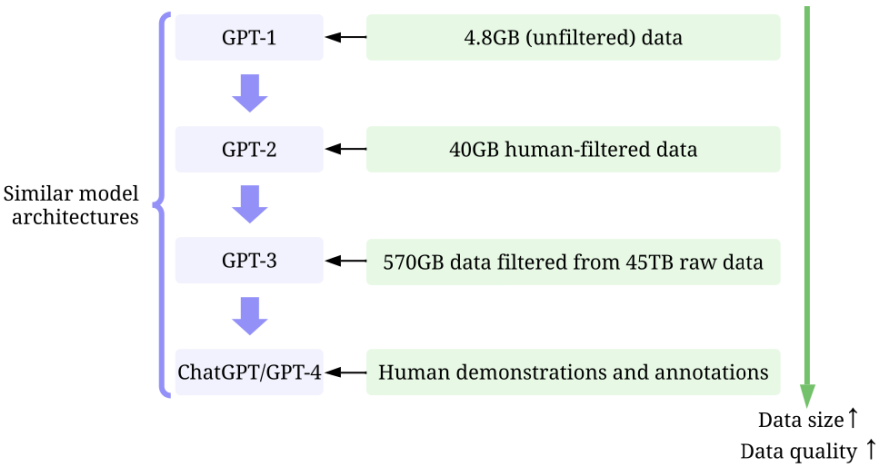



Figure 8: The improved performance of OpenAI models and LLMs in general has gone hand in hand with a very sharp increase in their size (number of model parameters) and in the quantity of data on which they have been trained.

Source : Zha et al. [2023a]

### 4.2.4 Critical Role of Data in LLMs training

The significance of data in reducing model errors is underscored by Hestness et al. [2017]. They empirically demonstrate that in the case of Deep Learning models, error decreases linearly with dataset size when plotted on a log error versus log data size graph. This insight appears to have been val-

idated by the facts, particularly by the Transformers architecture, which has facilitated the use of increasingly larger datasets, thereby yielding superior results.

A data scaling law is a simple formula that maps dataset size to error. The curve commences in the small data region, where models grapple with learning from a limited number of training samples. In this region, models can only perform as well as "best" or "random" guessing. The middle portion of learning curves is the power-law region, where each additional training sample provides information that assists models in enhancing predictions on previously unseen samples. The power-law exponent determines the steepness of this curve, or the slope when viewed on a log-log scale.

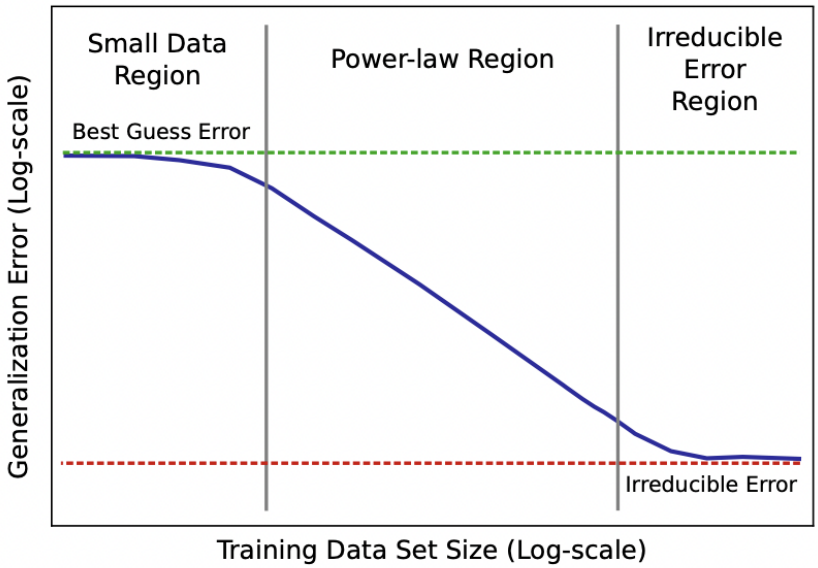



Figure 9: Data scaling law represensation

Source : Hestness et al. [2017]

This exponent is an indicator of the difficulty models face in representing the data generating function. The lower bound of this curve includes Bayes error, which is the information theoretic lower bound based on the data generating function, and a combination of other factors that result in imperfect generalization. For instance, mislabeled samples in the training or validation datasets are likely to cause irreducible error. This is referred to as the irreducible error region.

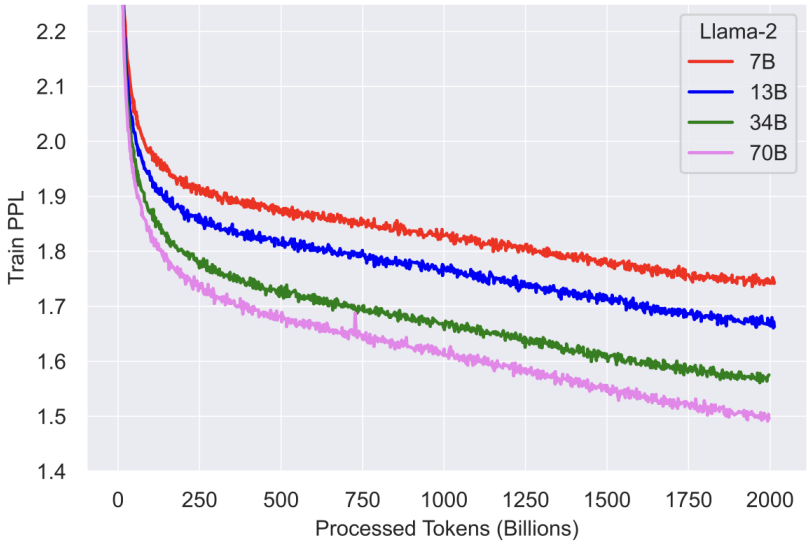



Figure 10: The Training Loss for Llama 2 models. We can observe that after pretraining on 2T Tokens, the models were still learning.

Source : Touvron et al. [2023a]

It's interesting to look at this graph taken from the LLama 2 model publication. It shows the error of the models (7, 13, 34 and 70 billion parameters) as a function of the number of tokens on which the model has been trained. As expected, the error decreases as the model is trained on a larger data set. But what's worth noting is that the different models, and in particular the largest, continue to reduce their errors. So, it seems we're still in the phase the authors (Hestness et al. [2017]) call the Power-law Region, and that we're still not near the "Irreducible Error Region".

To explore further the crucial issue of data in LLM training, we're going to take a look at a new concept that has been developing recently, called Data-centric AI.

➤ ***Data-centric AI***

Data-centric AI, as defined by Andrew Ng [21], is the discipline of systematically engineering the data used to build an AI system. This approach contrasts with model-centric AI, where the primary objective is to iterate on the models to enhance their performance. The main issue with the model-centric approach is that it places excessive trust in the data, which can be flawed or biased. Improving data quality is likely to enhance performance.

[21] Ng is a former Stanford professor, and is also well known for his Machine Learning and Deep Learning courses, which have played a major role in the democratization of these disciplines.

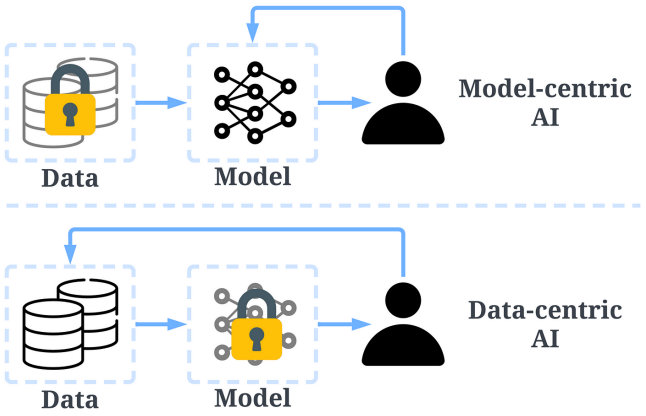



Figure 11: Model-centric AI as opposed to Data-centric AI

Source : Zha et al. [2023b]

Various issues may arise with the data, including inaccurate labels, duplicates, and biases. Overfitting a dataset may not necessarily lead to improved model behaviors. High-quality data is crucial, but data collection is time-consuming and it is often more efficient to leverage existing datasets. Data labeling is costly, as evidenced by the crowdsourcing efforts of OpenAI.

Data preparation involves cleaning and transforming data to remove noise (duplicates, outliers, missing values, etc.), feature engineering, and transformation (normalization). Data reduction aims to simplify the dataset and make it more understandable, while data augmentation strategies, such as image flipping, increase data diversity to enhance the performance and accuracy of the model Zha et al. [2023a].

The importance of data for training cannot be overstated. Large Language Models (LLMs) are capable of ingesting vast amounts of data, which is why they perform so well on a wide array of tasks. However, data collection is a challenging task and remains critical for achieving good performance on downstream tasks.

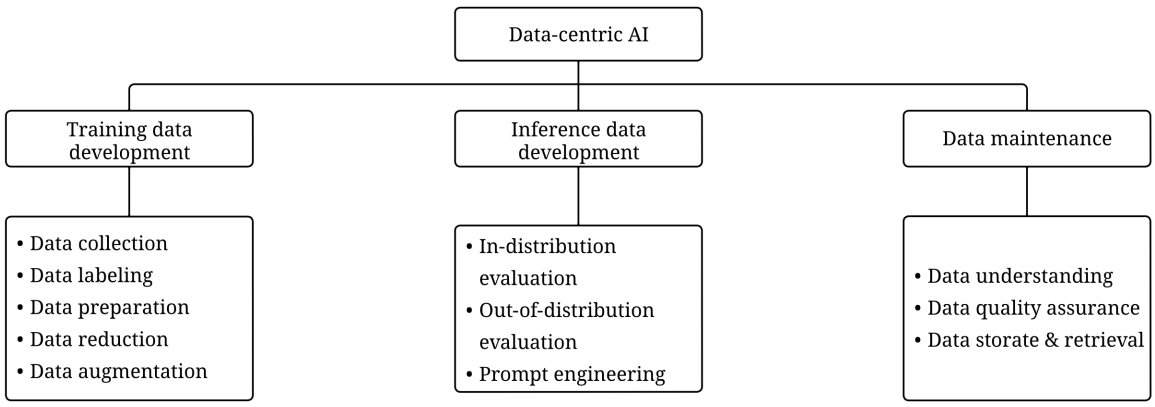



Figure 12: The different stages of data processing for LLM in production

Source : Zha et al. [2023b]

Now that we've seen the importance of data for LLM training, and the importance of improving a model not through its number of parameters, or just through its architecture, but also through the quality of the data it receives, we're going to take a look at graphical cards in Deep Learning. In fact, they have played a major role in the development of LLMs and deep learning in general, delivering extremely significant performance gains.

### 4.2.5 GPU over CPU

Today's models have reached such large sizes (up to 1T parameters) thanks to advances in graphics card manufacturing. It therefore seems important to understand how these gpu's have made the existence of these models possible and enabled their improvement (both in size and inference).

In deep learning, we often work with very large amounts of data and complex mathematical operations. These operations are often matrix operations, which are highly parallelizable tasks. This is where GPUs (Graphics Processing Units) come in, as they are specifically designed to handle such tasks. Here are some specific reasons why we use GPUs instead of CPUs for deep learning:

➤ ***Memory Bandwidth***

GPUs have much higher memory bandwidth than CPUs, meaning they can read and write data to their memory much faster. This is important for deep learning, where we often need to process large amounts of data, especially, during the pre-training phase.

➤ ***Parallel Processing***

GPUs are designed to perform many operations simultaneously. They have thousands of cores, compared to the dozens at most in CPUs. This makes GPUs particularly well-suited for tasks that can be broken down into many small, independent tasks, like the matrix and vector operations that are common in deep learning.

➤ ***Higher Computational Power***

The architecture of a GPU is such that it can execute thousands of threads concurrently. This enables them to perform a large number of simple computations rapidly and at the same time. This is crucial for deep learning algorithms, which involve a large amount of matrix and vector operations.

➤ ***Hardware Optimization for AI***

Many recent GPUs are specifically optimized for AI and deep learning tasks, with features that make these tasks even faster and more efficient. These include specialized tensor cores for improved matrix operations and hardware support for reduced-precision arithmetic, which can significantly speed up deep learning without significantly impacting accuracy.

➤ *Software Ecosystem*

Tools and libraries for deep learning have been extensively developed with GPU support in mind. For example, TensorFlow, PyTorch, and CUDA have been optimized for GPU utilization.

Overall, while CPUs are excellent for tasks that require complex logic and cannot be easily parallelized, GPUs are far superior for the highly parallelizable and computationally intensive tasks found in deep learning. As such, using a GPU can result in a significant speedup in training time, which is crucial given the large datasets and complex models often used in this field.

It's crucial to note that GPUs are often extremely expensive parts, and to train an LLM from scratch, requires access to several hundred or even thousands of GPUs to train these very large models. When asked about the cost of training the GPT 4 model, OpenAI's CEO estimated the cost at over $100 million (including only hardware and electricity).

For resources concerning the use of GPUs in LLM training and inference, the GPU Benchmark by Lambda (a GPU cloud provider) and this blog are good introduction.

Before delving into the limitations of these models, it is crucial to examine the Open Source environment, a significant catalyst for innovation and the proliferation of ideas in the field of Language Model Learning (LLM).

### 4.2.6 Open Source LLM

The last decade has witnessed major strides in deep learning, primarily driven by the research centers of leading American tech firms such as Google, Facebook, and Microsoft. These companies' access to vast datasets, advanced computational resources, and top-tier talent has enabled them to spearhead the development of transformative models. For instance, the paper behind the development of the Transformer model was authored by Google researchers, while the Llama model, which we will discuss next, was developed by researchers at Facebook. These models were made open source, fostering significant development within the community and leading to a surge in the number of models available.

To evaluate the various models, the primary resource is the Hugging Face Open LLM Leaderboard,

which tracks, ranks, and evaluates LLMs and chatbots as they are released [22]. This leaderboard employs four different benchmarks using the *Language Model Evaluation Harness* repository. The tests assess a model's ability to answer grade-school science questions, make commonsense inferences, demonstrate multitask accuracy on subjects like elementary mathematics, US history, computer science, law, and more, and evaluate the model's propensity to reproduce online falsehoods.

Another benchmark, *FLASK*, tests for logical robustness, correctness, and efficiency, commonsense understanding, factuality, metacognition, insightfulness, completeness, comprehension, conciseness, readability, and harmlessness. These benchmarks provide a comprehensive evaluation of a model's capabilities and limitations.

Open Source models, often referred to as foundation models, include Llama (Touvron et al. [2023]), Llama 2 (Touvron et al. [2023a]), Falcon, and MPT. These models serve as the basis for further development and fine-tuning. Fine-tuned models, such as Alpaca (et al. [2023b]), Vicuna, and Open Assistant, are derived from these foundation models and enhanced through techniques like fine tunings (using QLora for example) or Reinforcement Learning from Human Feedback (RLHF).

One of the main benefits of these open source models, in addition to fostering healthy competition, is to avoid dependence on a single player (OpenAI) that could have a monopoly on the sector. What's more, because some data may be sensitive, a player may not want to send it to a third party. This is why an Open Source model can be a valuable tool, enabling users to benefit from this technology while retaining greater control over their data. We will delve deeper into this issue in the case studies section. Before that, we'll take a look at the LLama 2 model, which is today's best-performing open source model, and for some, with results close to those of Chat GPT 3.5.

#### 4.2.7 Llama 2

At the time of writing, Llama 2, released in July 2023 [23], is by far the best Open Source model. Trained by Meta and exist on 3 different versions : 7B, 13B and 70B parameters.

It was trained on 2 trillion of tokens, and with a context length of 4096 token. This is a foundation model. It has then been fine tuned for chat purposes with 100 000 supervised training and more than 1 000 000 human preferences via RHLF.

The LLaMA-2 paper (et al. [2023a]) presents a new large language model (LLM) developed by Facebook (Meta), which competes with models like GPT-3 and GPT-4. The success of LLaMA-2 is largely attributed to the use of reinforcement learning from human feedback (RLHF), a process that

[22] If many models are present in this ranking, they are in fact mostly fine tunings of foundation models such as Llama 1, Llama 2 (et al. [2023a]) or Falcon.

[23] For the blog post about its release, see: https://ai.meta.com/llama/

involves collecting data, training a reward model, and fine-tuning with reinforcement learning. This allows to further align model behavior with human preferences and instruction following.

In the data collection phase, human annotators write a prompt based on an alignment property (e.g., helpfulness or safety), and the model generates two responses. The annotator then labels the preferred response. This human preference data is used to train a reward model, which shares the same architecture and weights as the LLM but includes a regression head. The reward model is fine-tuned to predict human preference when given a prompt and response pair.

The reward model is then used to automate or predict human feedback for fine-tuning, optimizing the LLM to maximize human preference using reinforcement learning algorithms. LLaMA-2 uses both Proximal Policy Optimization (PPO) and rejection sampling fine-tuning in sequence during alignment.

While most open-source models focus on supervised fine-tuning (SFT), the authors of LLaMA-2 found that RLHF was more effective, particularly in terms of cost and time. They found that RLHF fosters a synergy between humans and LLMs during the annotation process.

The authors also noted that RLHF is less noisy and easier to accomplish than SFT, as it only requires humans to provide binary preference feedback. Furthermore, the responses generated by the LLM during RLHF can surpass the writing abilities of human annotators, which is not possible with SFT.

In conclusion, the LLaMA-2 model's emphasis on both SFT and RLHF has led to its success, as RLHF suppresses low-quality outputs over time and improves the model's behavior with respect to desired alignment criteria, such as helpfulness and safety.

To conclude this section on LLMs, it's important to point out the limitations of these models, because despite their great capabilities, these limitation can have a major impact on their use.

### 4.2.8 Capabilities and limitations of LLMs

Large Language Models (LLMs) exhibit both remarkable capabilities but also significant limitations. This section is inspired by the insights of Yann Le Cun [24], a leading AI researcher at Meta, and a famous researcher awarded with the Turing Award. It is clear that while LLMs can generate fluent and plausible responses, they often struggle with factual consistency, leading to hallucinations - predictions that seem plausible but are factually incorrect. This is compounded by their limited understanding of the world, particularly regarding information that has emerged post-training. LLMs lack common sense and the ability to plan their responses, which can lead to logical errors and inconsistencies.

[24] For more information on the subject, please refer to this talk with Yann Le Cun: (Link)

The challenge of ensuring appropriate behavior in LLMs is another significant hurdle. Current methods, such as Reinforcement Learning from Human Feedback (RLHF), are imperfect solutions, as they can be circumvented through specific prompts, potentially leading to toxic outputs. Attempts to augment LLMs with "tools" such as search engines, advanced calculations, or database queries have also proven to be largely imperfect.

The fluency of LLMs can be misleading, often giving the impression of correctness when the output may be fundamentally flawed. This is due to their limited reasoning abilities and their lack of understanding of how the world operates. A study by Dziri et al. [2023] suggests that these limitations may be inherent to the autoregressive nature of LLMs, where errors in token prediction can propagate and limit the model's ability to solve complex tasks or recognize novel patterns. This issue becomes exponentially more likely as the problem or response grows larger.

LeCun further emphasizes that LLMs are not controllable and cannot be made inherently factual or non-toxic. He identifies three remaining challenges for AI and ML:

- Learning to create representations and predictive models of the world.
- Learning to reason. At the moment LLM makes like subconscious task, without real planing, making reasoning compatible with learning.
- Learning to plan complex task to satisfy objectives through hierarchical reasoning.

Having explored the deep learning innovations that led to the development of large language models (LLMs) in Section 4, we now have a solid foundation to understand their potential applications in the ESG data processing domain. LLMs have demonstrated remarkable capabilities in understanding and generating human-like text, which opens up new avenues for addressing the challenges faced by traditional ESG data processing approaches.

In Section 5, we will discuss various ways AI and LLMs can be utilized to enhance ESG data processing. We will explore how these technologies can automate ESG data extraction, enable advanced NLP-based analysis, and facilitate data access and interpretation, among other potential applications. By examining these applications, we will gain insights into how AI and LLMs can impact ESG data processing and create new opportunities for investors and other stakeholders.

# 5 Potential Applications of AI and LLMs in ESG Data Processing

The advent of Artificial Intelligence (AI) and Large Language Models (LLMs) has opened up new avenues for processing and analyzing Environmental, Social, and Governance (ESG) data. These advanced technologies has the potential to automate the extraction of ESG data, manage unstructured data, enhance data quality and accuracy, and facilitate data access and interpretation. This section delves into the potential applications of AI and LLMs in ESG data processing, highlighting how they can revolutionize the way organizations handle and interpret ESG data.

## 5.1 Automating ESG Data Extraction with LLMs

### 5.1.1 Managing Unstructured Data

As previously discussed, ESG data is predominantly unstructured, making it a complex and challenging dataset to process. However, Language Model Learning (LLMs), with their inherent capability to handle unstructured and textual data, emerge as ideal candidates for addressing these challenges.

The primary concern is whether LLMs, despite their known issues with hallucination, can reliably extract pertinent information from such data.

Artificial Intelligence (AI), Natural Language Processing (NLP), and LLMs are instrumental in automating data extraction and analysis tasks. By doing so, they possibly can significantly reduce the time and labor required for ESG data processing, thereby enhancing the efficiency and scalability of these processes.

LLMs, in particular, facilitate efficient text mining and information extraction from vast volumes of textual data and they could automate the process of identifying and extracting relevant information for ESG data analysis.

The following points highlight the key roles of LLMs in ESG data extraction:

1. Transforming unstructured data into structured formats: LLMs can convert unstructured text into structured data, making it easier to analyze and interpret.

2. Handling diverse data sources and formats: Everything that is text can be process by a LLM. Thus, data from various sources and in different formats, such as PDF and HTML, can enhance the comprehensiveness of the analysis.

3. Minimizing manual intervention: By automating the data extraction process, LLMs could reduce the need for manual intervention, thereby saving time and resources.

We have just discussed the ability of LLMs to process unstructured textual data in order to extract information. In the next sub-section, we'll look at how vector databases can help us avoid one of the weaknesses of LLMs, namely their limited context length.

## 5.2 Leveraging Large Language Models and Vector Databases for ESG Data Extraction

The potential of Large Language Models (LLMs) and vector databases in the realm of Environmental, Social, and Governance (ESG) data extraction is immense. This section delves into the ways these technologies can be harnessed to enhance ESG data extraction and processing, enabling more efficient and accurate semantic searches.

In an ideal scenario, the integration of LLMs and vector databases would enable users question-based searches about companies, sectors, years, and other parameters, and receive relevant answers and information in return. The accurate information would be retrieved from the vector database and the answer would be generated by the LLM. It should be stressed once again that much of the information contained in these reports is in text or in an unstructured form. This could facilitate comparisons of ESG performance indicators for a company against its competitors or its own historical data or allow to make it easier to extract this unstructured data and convert it into structured data, which could then be more easily queried using existing classic tools. Moreover, this database could serve as a source of truth, since, to answer the questions, the documents and sections used to answer the questions would be returned.

The creation of a comprehensive ESG database would involve the collection of vast amounts of data from ESG reports, complete with metadata. This metadata, which could include specific information about companies, sectors, types of reports, years, and more, would serve as a filter to narrow down the documents to be searched. A semantic search could then be performed within this subset of documents to identify the most relevant ones.

Vector databases, such as Weaviate or FAISS, are instrumental in performing similarity searches. These databases are populated with the embeddings of the documents, enabling similarity search over the documents, even for very large databases. This is particularly useful given the limited context window of LLMs, which can only use a finite number of documents to generate text. By retrieving the most relevant documents (using similarity or hybrid search [25]), vector databases can provide the necessary context to the LLMs for text generation.

By providing the LLM with the necessary context to answer a query, this approach can significantly improve language modeling performance, mitigate the problem of factually inaccurate text

[25] A hybrid search approach is used by Stackoverflow. They describe their implementation in this blog: Link

generation, and avoid hallucinations.

Retrieving the relevant documents is a crucial step in this process. This can be achieved through keyword retrieval, embedding retrieval, or a hybrid approach. The order in which these documents are retrieved also plays a significant role in the quality of the context provided to the LLM. As underlined by the article "Lost in the Middle" (Liu et al. [2023]), it is best to have the most relevant documents at the beginning and the end of the retrieved documents. This is because LLMs tend to focus their attention mainly on what they see at the beginning and the end.

The integration of LLMs and vector databases presents a promising approach to ESG data processing. By leveraging these technologies, we can create a robust ESG database that enables efficient and accurate semantic search and querying. In the case studies, we'll quickly present an example of such an application and show that it's relatively simple to prototype such an application with the tools available today. This, in turn, can facilitate more informed decision-making in the realm of ESG.

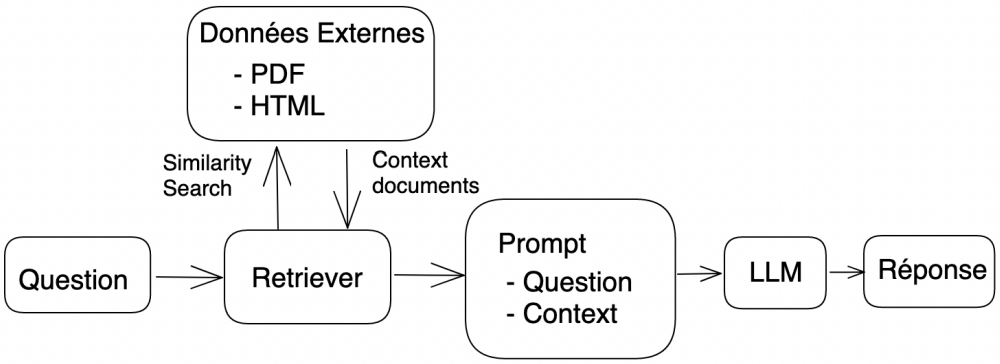



Figure 13: Schema for extracting external data from a query, then providing LLM with context

## 5.3 Applications of Chat model in ESG data processing

### 5.3.1 Summarization and Question Answering over ESG Reports

The role of an ESG analyst is multifaceted, encompassing the evaluation of a wide range of environmental, social, and governance (ESG) factors that can impact a company's performance and investment potential. This evaluation often involves the analysis of extensive ESG reports, which can be time-consuming and complex. However, the advent of Language Learning Models (LLMs) can significantly streamline this process, offering transformative potential for ESG analysis.

LLMs can be leveraged to summarise and answer questions over ESG reports, thereby enhancing the efficiency and effectiveness of ESG analysis. These models are capable of processing large volumes of text, extracting key information, and presenting it in a concise and comprehensible manner. This can greatly reduce the time and effort required to review and understand these reports, allowing

analysts to focus more on strategic decision-making.

Moreover, LLMs can be trained to answer specific questions about the content of ESG reports. For instance, an analyst could ask the model to identify the main environmental risks faced by a particular company, or to summarise the company's governance structure. The model would then scan the report and provide a precise and accurate response, based on the information contained within the document.

This application of LLMs can also support more nuanced and sophisticated ESG analysis. By automating the initial stages of report review, analysts can dedicate more time to interpreting and contextualising the information provided by the model. This could lead to deeper insights and more informed investment decisions.

Furthermore, the use of LLMs for summarisation and question answering can also enhance the consistency and objectivity of ESG analysis. By relying on machine learning algorithms to extract and summarise information, analysts can mitigate the risk of human bias and ensure a more standardised approach to report evaluation.

In conclusion, the application of LLMs for summarisation and question answering over ESG reports presents a promising avenue for enhancing the efficiency, depth, and objectivity of ESG analysis. However, the model has not been specifically trained to meet these needs, which are specific to the ESG sector. To potentially improve the model's ability to answer these questions factually and accurately, one possible solution is fine-tuning. We'll look at this in the next section.

### 5.4 Domain Specialization via Fine-tuning and RLHF LLMs for ESG-specific tasks

Training an LLM is a resource-intensive process, both in terms of time and environmental impact. Therefore, pretrained LLMs, often referred to as foundation models, are widely used as they possess general capabilities, a broad knowledge base and they can be adapted for various downstream tasks.

However, for specific tasks such as Environmental, Social, and Governance (ESG) data processing and data analysis, a more specialized domain knowledge is desirable because laws and regulations can change rapidly. Moreover, as we saw earlier, the tasks of an ESG Analyst are specific, and we'd like to be able to enhance the skills of the model to answer these questions properly. This can be achieved through instruction fine-tuning and reinforcement learning through human feedback (RLHF). Instruction fine-tuning involves training LLMs on a collection of tasks described via instructions, which significantly improves zero-shot performance on unseen tasks (Wei et al. [2022]). Despite the cost-effectiveness of fine-tuning, this requires being able to manage the model "in-house", which can be potentially restrictive, costly in terms of infrastructure and requires more in-house skills than just

calling an API.

Hence, in the next section, we will study in depth Fine tuning methods as they're a powerful tool to improve our LLM.

### 5.4.1 Fine-tuning Methods

Fine-tuning used to be a challenging process, especially for large models. However, advancements in fine-tuning methods have made it more accessible and affordable. Recent advances in the field of fine tuning LLM have led to the development of a number of techniques that we now call Parameter-Efficient Fine-Tuning (PEFT).

Lialin et al. [2023] describe Parameter-Efficient Fine-Tuning (PEFT) as a method that aimed at addressing the challenges of training large language models by only training a small set of parameters. These parameters could be a subset of the existing model parameters or a set of newly added parameters. The efficiency of these methods varies in terms of parameter efficiency, memory efficiency, training speed, the final quality of the model, and additional inference costs.

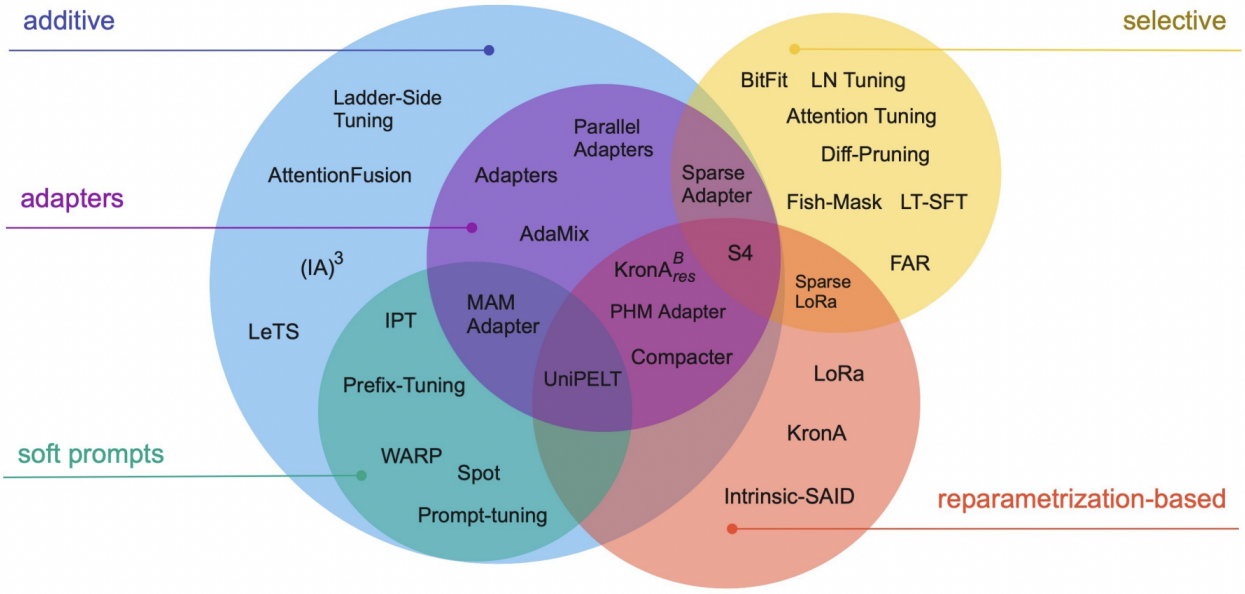



Figure 14: PEFT taxonomy, from Lialin et al. [2023]

PEFT methods can be classified based on their underlying approach or conceptual framework. They can either introduce new parameters to the model or fine-tune a small subset of existing parameters. They can also be categorized according to their primary objective, whether it is to minimize memory footprint or storage efficiency.

➤ ***Additive Methods***

Additive methods augment the existing pre-trained model with extra parameters or layers and train only the newly added parameters. This is the most explored category of PEFT methods. Within this category, two significant subcategories have emerged: Adapter-like methods and soft prompts.

*Adapters* are a type of additive parameter-efficient fine-tuning method that introduces small fully connected networks after Transformer sub-layers. The original parameters of the model remain fixed during the training process, and only the parameters of the adapters are updated. Adapters are used when we want to preserve the original model parameters and avoid catastrophic forgetting. They are also beneficial when we want to apply the model to multiple tasks simultaneously, as different adapters can be used for different tasks. Various variations of Adapters have been proposed, including modifying the placement of adapters, pruning, and using reparametrization to reduce the number of trainable parameters.

*Soft Prompts* aim to control the behavior of a language model by modifying the input text. However, these methods are difficult to optimize and are inherently limited in the number of training examples by the maximum model input length. To address these drawbacks, the concept of "soft" prompts was introduced. Soft prompts are particularly useful when we want to guide the model towards certain types of responses or when we want to incorporate additional information that is not present in the input data.

➤ ***Selective Methods***

Selective PEFT methods fine-tune only a few top layers of a network. Modern approaches are usually based on the type of the layer or the internal structure, such as tuning only model biases. An extreme version of selective methods is sparse update methods which can completely ignore the structure of the model, and select parameters individually. However, sparse parameter updates present multiple engineering and efficiency challenges.

➤ ***Reparametrization-based Methods***

Reparametrization-based PEFT methods leverage low-rank representations to minimize the number of trainable parameters. The most well-known reparametrization-based method is Low-Rank Adaptation or LoRa, which employs a simple low-rank matrix decomposition to parametrize the weight update matrix. This approach is straightforward to implement and has been evaluated on models with up to 175 billion parameters with great success.

When comparing PEFT methods, it is important to consider five dimensions: storage efficiency, memory efficiency, computation efficiency, accuracy, and inference overhead. While these dimen-

sions are not completely independent from one another, improvements in one do not necessarily translate into improvements in others. Achieving parameter efficiency alone does not necessarily lead to reduced RAM usage.

For the fine tuning, we can see that many methods exists. At the moment of writing, the most popular method is the LoRA method. Hence, we will focus on this methods and try to understand in which case it could be useful for us.

LoRA (Hu et al. [2021]) has revolutionized the training of LLMs, making it more accessible and cost-effective. The introduction of low rank adaptation matrices has significantly reduced memory requirements. This is achieved by keeping the pretrained weights fixed and appending these low rank adaptation matrices to the attention layers, as well as the feedforward matrices. These matrices, being much smaller, are the ones trained.

The advantages of LoRA are manifold and significant. Firstly, it has brought about a marked improvement in efficiency by reducing the number of trainable parameters, thus making fine-tuning more efficient. Secondly, the ability to keep pretrained weights frozen has opened up the possibility of creating multiple lightweight and portable LoRA models that can be used for a variety of tasks.

Thirdly, LoRA's compatibility with other parameter-efficient methods adds to its versatility. This means that it can be combined with other methods to achieve even greater efficiency. Fourthly, despite these efficiencies, there is no compromise on performance. Fine-tuned LoRA models have been found to perform on par with fully fine-tuned models.

Lastly, one of the most significant advantages of LoRA is that it does not add any latency. The adapter weights can be merged with the base model without causing any inference delays. This is a critical advantage in real-time applications where delays can have significant implications.

In essence, the Low Rank Adaptation (LoRA) method has not only made the training of LLMs more efficient but also more versatile and effective. One of the problems that remained with the LoRa method was that it required the entire model to be loaded into memory, which could required a large number of GPUs. However, a new method build on LoRa has been developped to solve this issue.

The introduction of Quantized LoRA (QLoRA) has further improved the efficiency of fine-tuning by reducing the GPU memory requirement. Dettmers et al. [2023] show that they were able to reduce the GPU memory requirement, when training a 65B model parameters, from 780 GB to 48 GB of GPU VRAM, making it possible to train large models on a single GPU. The Parameter-Efficient Fine-Tuning (PEFT) library from Hugging Face offers an efficient way to adapt pre-trained language models to various downstream applications without fine-tuning all the model's parameters.

### 5.4.2 Fine-tuning for specific tasks

In the realm of ESG data processing, fine-tuning a language model for specific tasks is a critical step towards creating a tool that can effectively assist ESG analysts. One such task could be portfolio comparison or notation, where the model is fine-tuned to assess whether a company fulfills certain ESG criteria. This could involve analyzing a variety of data sources, including company reports, news articles, and regulatory filings.

Another potential task is enabling the model to call an API, as suggested by Qin et al. [2023]. With the increasing amount of data being made available via APIs, having a language model that can effectively interact with these APIs is central to the development of this technology. This could involve fine-tuning the model to understand and generate appropriate API calls, as well as to process and analyze the data returned by the API.

Additional tasks for fine-tuning could include ESG risk assessment, regulatory compliance,, and reporting. Each of these tasks would involve fine-tuning the model to analyze and interpret relevant data in a way that is useful for the specific task at hand. For reporting, the model could be fine-tuned to analyze and summarize relevant data in a way that is easy to understand and communicate.

In conclusion, the fine-tuning of a language model for specific tasks is a crucial step in the development of a useful tool for ESG data processing. The specific tasks chosen for fine-tuning will depend on the needs and goals of the ESG analysts using the tool.

To conclude on the main applications of LLMs in the field of ESG data, it is worth mentioning the possibilities they offer for facilitating access to ESG data and its interpretation.

## 5.5 Facilitating data access and interpretation

LLMs have made it possible to facilitate data access and interpretation in a more efficient and user-friendly manner. One of the ways this is achieved is through the development of conversational interfaces for ESG data. These interfaces, powered by AI and NLP, can understand and respond to natural language queries, making it easier for users to access and interpret data. For instance, an agent designed to interact with SQL databases can process natural language queries, convert them into SQL queries, and return the results in a user-friendly format. This not only simplifies data access but also makes it possible for non-experts to interact with the data.

Similarly, a Pandas agent can be used to interact with data stored in Pandas dataframes. This agent can understand natural language queries, perform the corresponding operations on the dataframe, and return the results. This can be particularly useful for data exploration and analysis, as it allows

users to quickly and easily manipulate and analyze data [26].

Another way to facilitate data access and interpretation is through the use of model tooling. This involves the use of tools that can call RestAPIs, enabling users to interact with models and data in real-time. This can be particularly useful for real-time ESG analytics and feedback, as it allows users to get immediate insights into the data.

AI, NLP, and LLMs can also be used to personalize ESG information delivery. By understanding the user's preferences and needs, these technologies can deliver tailored information to the user, making it easier for them to understand and interpret the data.

Furthermore, these technologies can be used to visualize ESG data in intuitive ways. By presenting data in a visual format, users can gain a more comprehensive understanding of the data, making it easier for them to identify trends and patterns. This can facilitate better decision-making and risk management.

In conclusion, AI, NLP, and LLMs offer new ways to interact with and interpret financial data, making ESG data more accessible to non-experts and enabling more efficient data exploration and analysis. By wrapping a model around an agent, which takes in user input and returns a response corresponding to an action and a corresponding action input, these technologies can facilitate data access and interpretation, opening up new possibilities for discovering previously hidden insights and trends within ESG information.

After examining the potential applications of AI and LLMs in ESG data processing in Section 5, we will present various case studies that showcase real-world case studies. This will allow us to evaluate the practicality and effectiveness of these AI-driven solutions, as well as identify potential challenges and limitations that may arise during their implementation.

[26]Such example already exist with GPT-4 Code Interpreter.

# 6 Case studies

In this section, we will showcase some practical cases demonstrating some of the possibilities offered by LLMs, particularly thanks to the existing libraries in the Python ecosystem. This will allow us to develop applications that enable the extraction of unstructured data from a PDF, and how it becomes possible to transform this data into a structured format for storage in relational databases. We will also see the importance of model hosting for better model control. Finally, we will conclude on the main lessons learned while working with these models.

## 6.1 Working with LLMs

When working with LLMs, we need to take into account several factors which can greatly affect the final output. Hence, it is essential to have those ideas in mind:

- **Prompts**: A prompt for a language model is a set of instructions or input provided by a user to guide the model's response, helping it understand the context and generate relevant and coherent language-based output, such as answering questions, completing sentences, or engaging in a conversation. With the two most popular models (GPT and Lllama) at the moment, there's two main part in a prompt, a system prompt which allows to modify the behavior of the LLM and a user prompt where the question of the user is passed.
- **A Language Model**: can be a simple LLM or a fine tuned one for chat purposes
- **Output parsers**: LLMs typically output text. When chating, this can be fine, but what if we want more structured output ? Hence, we need a parser, something that will take some string as input and that will allow us to have a final format that we would want. The prompt can be used for that, but in the case we want strict output result, it would likely not be enough.

In this section, we will demonstrate how to interact with the OpenAI API using Python and the Langchain library, a Python-based tool. This library will serve as our interface to programmatically access a wide range of LLM-related functions. In addition, it will facilitate the integration of LLMs into real-world applications and enables connections to diverse data sources such as PDF or any other format from which can extract textual data. Moreover, using a library such as Langchain is useful as it allow us to think about how we interact with a LLM instead of thinking about how to interact with the OpenAI API. All the examples we're showing here, which are using GPT models could be used with other models, giving they are powerful enough to answer correctly to our questions (hence, we could replace the GPT models by Llama 2 models for example).

First, as an example, we can see that it's very easy to start a conversation with an LLM using the Langchain library, in this case GPT-4 model.

```
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(temperature=0.2, model="gpt-4")
llm.predict("Hello world")
```

```
'Hello! How can I assist you today?'
```

Figure 15: Hello world example using GPT-4 with the Langchain library

We can modify the system prompt so that the model is aware of what we are expecting from him. Here, we simply give a name to the chatbot, Barbatus (which is a bird).

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage, SystemMessage

llm = ChatOpenAI(temperature=0.2, model="gpt-4")
ai_response = llm([
    SystemMessage(content="Hello, I am a chatbot called Barbatus. I am here to help you with your questions."),
    HumanMessage(content="What is your name?"),
])
ai_response.content
```

```
'My name is Barbatus.'
```

Figure 16: Change LLM behavior by defining a "System Message"

The prompt is an important element when working with an LLM, as it enables us to define precisely what is expected, and avoid ambiguities that could mislead the model. We've shown above that it's very simple to modify the system prompt, but it's also possible to modify the prompt to accept parameters, a context for example, that come from external documents, such as PDFs. This is what we'll show next.

## 6.2 Effective Implementations of AI and LLM in ESG Data Processing

This section explores the successful application of Large Language Models (LLMs) in the processing ESG data. The primary focus is on the extraction of financial and environmental values from PDF documents, such as revenues and greenhouse gas (GHG) emissions.

The Langchain library, is used in this context. It simplifies the combination of the tools available when working with LLM.

Below are the main steps we will follow for this application:

1. A PDF file is loaded, and the content is divided into manageable chunks, each containing a

maximum of 4000 tokens (approximately, 3000 words).

2. Each chunk is then vectorized, resulting in an associated embedding vector. Both the vector and the corresponding text are stored in a vector database, Weaviate in our case.

3. Semantic search is performed on the PDF (e.g., "What are the revenue of each sector for 2021?"), specifying the number of chunks desired. The query is vectorized into a embedding, and we compute a similarity search between this query and all other embeddings in our vector database. This process returns the $k$ most similar chunks to the query.

4. The retrieved context is then provided to a model to answer a specific question.

Since we're using Weaviate, we need to start a Weaviate server that will be able to respond to our requests.

```
import weaviate

client = weaviate.Client(embedded_options=weaviate.embedded.EmbeddedOptions())
```

```
Started /Users/romainjouhameau/.cache/weaviate-embedded: process ID 43659
```

Figure 17: Start a simple Weaviate server to store our embeddings, associated text and the metadata from our external documents, here a PDF file

The Figure 18 shows that extracting content from a PDF is very simple. For this example, we will use the Apple Environmental Progress Report from 2022 which can be found at this address: Link. In this PDF document, there are 128 pages. Each document will have at most, 4000 tokens, meaning, for a single PDF page, we can have more than 1 document. Hence, instead of having 128 documents, we now have 152 documents. A document contains the page content, the filename the source document and the page from which the page content has been extracted. Of course, we could add other metadata to this document to suit our needs.

```
from langchain.document_loaders import PyPDFLoader

loader = PyPDFLoader("Apple_Environmental_Progress_Report_2022.pdf")
documents = loader.load_and_split()

print("Instead of the 128 pages of the PDF, we will split text within each page so that the number of tokens is less than 4000.")
print(f"Number of documents: {len(documents)}")
```

```
Instead of the 128 pages of the PDF, we will split text within each page so that the number of tokens is less than 4000.
Number of documents: 152
```

Figure 18: Extract the text content from a PDF. Each document will have at most, 4000 tokens, meaning, for a single PDF page, we can have more than 1 document.

To add the documents to our Weaviate vector database, we simply need to run the code below. Note that OpenAI Embeddings has been chosen for the creation of embeddings. We could also use

an Open Source embedding model, but here it's much simpler to simply call the OpenAI API. Each document is now vectorized (meaning each document has its own embedding), and stored in the vector database.

```
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import Weaviate

db = Weaviate.from_documents(documents, OpenAIEmbeddings(), weaviate_url="http://127.0.0.1:6666", by_text=False)
```

Figure 19: The 152 documents and their embeddings are stored into the Weaviate vector database.

Now that our documents are vectorized, we can search them directly, using a similarity search, which in this case corresponds to a cosine similarity. All we need to do is to make a query. This character string will also be transformed into an embedding, via the API OpenAI Embeddings endpoint.

As a result, we will get the four documents which are the most similar to our question, and thus the document which are the most likely to answer our question.

```
query = "What is the Energy use for each facility 2021? (in MWh or GW)"
docs = db.similarity_search(query)

for doc in docs:
    print("Source: ", doc.metadata['source'], "Page: ", doc.metadata['page'])
    print(doc.page_content[:100],  "\n")
```

```
Source:  Apple_Environmental_Progress_Report_2022.pdf Page:  85
Fiscal year
Unit 2021 2020 2019 2018 2017
Corporate facilities energy use
Electricity Total MWh 2,85

Source:  Apple_Environmental_Progress_Report_2022.pdf Page:  18
and light. As each site becomes operational, we monitor how
well we're performing and make needed a

Source:  Apple_Environmental_Progress_Report_2022.pdf Page:  19
In fiscal year 2021, our energy efficiency program avoided an
additional 15.7 million kilowatt-hour

Source:  Apple_Environmental_Progress_Report_2022.pdf Page:  91
A focus on data centers
We used over 1.96 billion kWh of electricity in fiscal year 2021
to power o
```

Figure 20: Run a cosine similarity search against the document in the vector database, given a question. This return the four most similar documents

We can now run everything with a single chunk of code. We will use a new class from Langchain called "*RetrievalQAWithSourcesChain*". This will create a "Chain". This will allow to chain the different components we described before. Hence, we define the following components:

- Our LLM: GPT-4
- Our chain type: "stuff". This tell the chain how we want the retrieved documents to be shown to the LLM. Here, we simply want to concatenate each document content (this strategy is called "stuff" in Langchain) and give it to the LLM as context. Other strategies exists and can be seen there.
- Our retriever: "db", this is our Weaviate database.

We also asked for the source documents to be returned.

Then, we can run our query. Hence, with this setup, we see that the query does in fact two things. It is used to retrieve the documents which are the most similar to the query, and then it is passed as a question, to the LLM.

```
from langchain.chains import RetrievalQAWithSourcesChain

query = """What is the Energy use for each facility 2021? (in MWh or GW)
If possible, decompose the result for the different regions and the different energy types and sub-types"""

chain = RetrievalQAWithSourcesChain.from_chain_type(
    ChatOpenAI(temperature=0.2, model="gpt-4"),
    chain_type="stuff",
    retriever=db.as_retriever(),
    return_source_documents=True,
)

response = chain({"question": query})

print(response['answer'])
print("Sources:")
for doc in response['source_documents']:
    print(doc.metadata)
```

```
The energy use for each facility in 2021 is as follows:

- Total electricity use for corporate facilities was 2,854,000 MWh, with 2,377,000 MWh used in the U.S. and 477,000 MWh
- Total fuel use for corporate facilities was 467,280 MWh, broken down as follows:
  - Natural gas: 203,010 MWh
  - Biogas: 208,620 MWh
  - Propane liquid: 40 MWh
  - Gasoline: 34,880 MWh
  - Diesel (other): 9,780 MWh
  - Diesel (mobile combustion): 10,950 MWh
  - Other Steam, heating, and cooling: 22,480 MWh
- The Denmark data center sourced 15 million kWh of renewable energy.
```

Figure 21: Query a PDF document and give the LLM with the right context to answer the user question

We can print the sources from which the context comes from. Note that the page numbering start at 0. Thus, the most similar document to the query is the document extracted from the page 86. From looking at the table located in that page, the LLM doesn't appear to have hallucinated. However, we need to be extremely careful with the output of an LLM as it can sometimes hallucinate.

```
Sources:
{'page': 85, 'source': 'Apple_Environmental_Progress_Report_2022.pdf'}
{'page': 18, 'source': 'Apple_Environmental_Progress_Report_2022.pdf'}
{'page': 88, 'source': 'Apple_Environmental_Progress_Report_2022.pdf'}
{'page': 96, 'source': 'Apple_Environmental_Progress_Report_2022.pdf'}
```

Figure 22: The four documents which are the most similar to the query when using cosine similarity search

➤ ***Lessons learned from this application***

This approach hinges on two key concepts: *document retrieval* via semantic search and the use of *LLMs* to interpret the context and accurately answer the question. This allows for natural language queries and responses.

However, evaluating the effectiveness of this setup, particularly when it fails to provide an appropriate answer, requires discerning the cause. The failure could be due to the LLM not receiving the correct context (indicating a failure in semantic search) or due to a failure in LLM reasoning or hallucination of results.

Also, a common error when supplying context to the LLM concerns context length. As we saw before, a LLM has a limited context length, meaning that the input and output tokens count combined must be below that limit. Hence, we may want to compress the information, summarize it or reduce the context size.

If we give the model a prompt that's too long, it will reply, telling us how many tokens we're passing in our sequence, and that we should respect the limit defined for this model. It's important to note that each model can have its own context length. For example, the GPT-3.5 model is available with context lengths of 4,000 and 16,000 tokens (although the model accepting 16,000 tokens is twice as expensive to use), while the GPT-4 model accepts up to 8,000 tokens.

In this relatively simple example, we were able to show that we can extract relevant information from a large document (over 100 pages in this case) using simple questions.

But if this example is interesting for obtaining an unstructured answer, and simply text, to obtain a structured answer that could allow us to extract data and easily store them in a relational database, we need to adapt our way of approaching the problem. That's what we'll do in the next example.

### 6.3 Processing of Unstructured Data

Building on the previous example, the aim is to not only interpret unstructured data but also extract raw data in a manner that allows for structured data storage in a database with appropriate types.

To achieve this, we can leverage on OpenAI function, a tool provided by OpenAI, will be used

under the hood [27]. It enables calls to fine-tuned GPT-3.5 and GPT-4 models and provides strict format answers, which can be beneficial for obtaining precise results. OpenAI function was made possible through the fine-tuning of GPT models.

To extract this data efficiently and accurately, we'll need to be more verbose with the code and use more of the tools provided by Langchain and its various classes.

Firstly, we will define two classes using Pydantic. This popular Python library allows us to easily create classes to store objects, instead of using the json formatting which is normally used by the OpenAI functions. The first class, *EnergyUsage* will be used to store single observation (the type of energy, the concerned region, the year of observation, the quantity, and its unit). Then, the second class, *EnergyUsageByYearAndRegion*, will store these observations in a list that will contain objects of the previous class, *EnergyUsage*.

This approach allows us to define the names of our variables, their type, and store them in a list. This will facilitate their subsequent use for storage in a database.

```python
from langchain.chains import RetrievalQAWithSourcesChain
from langchain.prompts import PromptTemplate
from langchain.prompts.chat import ChatPromptTemplate, HumanMessagePromptTemplate
from langchain.chat_models import ChatOpenAI
from langchain.chains import create_qa_with_sources_chain, create_qa_with_structure_chain
from langchain.chains.combine_documents.stuff import StuffDocumentsChain
from langchain.chains import RetrievalQA
from pydantic import BaseModel, Field
from typing import List
from langchain.schema import Document

class EnergyUsage(BaseModel):
    """Energy Usage schema."""
    type: str = Field(..., description="Energy type")
    region: str = Field(..., description="Region")
    year: int = Field(..., description="Year")
    quantity: float = Field(..., description="Energy consumption in MWh or GW")
    unit: str = Field(..., description="Unit")

class EnergyUsageByYearAndRegion(BaseModel):
    """Energy Usage for each type, sector and region."""
    data: List[EnergyUsage] = Field(..., description="Energy Usage for each type, sector and region")
```

Figure 23: Define two class which will be used to store the extracted data from the provided context

Next, we define our prompts to specify to the model how we want the context to be provided.

[27] OpenAI announcement and Langchain implementation

```
prompt_messages = [
    SystemMessage(
        content=(
            "Use the following pieces of context to answer the question at the end."
            "If you don't know the answer, just say that you don't know, don't try"
            "to make up an answer.")),
    HumanMessagePromptTemplate.from_template("{context}"),
    HumanMessagePromptTemplate.from_template("Question: {question}"),
]

chain_prompt = ChatPromptTemplate(messages=prompt_messages)

doc_prompt = PromptTemplate(
    template="Content: {page_content}\nSource: {source}",
    input_variables=["page_content", "source"],
)
```

Figure 24: Define the prompts and the expected variables within the prompts (context, question, page_content and source)

The most important function used is called *create_qa_with_structure_chain*. It will allow us to create a chain that will contain our context, our question, and the format of the answer we expect. This uses the OpenAI functions we discussed earlier.

```
qa_chain_pydantic = create_qa_with_structure_chain(
    ChatOpenAI(temperature=0.2, model="gpt-4-0613"), EnergyUsageByYearAndRegion, output_parser="pydantic", prompt=chain_prompt
)
final_qa_chain_pydantic = StuffDocumentsChain(
    llm_chain=qa_chain_pydantic,
    document_variable_name="context",
    document_prompt=doc_prompt,
)
retrieval_qa_pydantic = RetrievalQA(
    retriever=db.as_retriever(), combine_documents_chain=final_qa_chain_pydantic, return_source_documents=True
)

query = "What is the Energy use for each facility 2021? (in MWh or GW)"
r = retrieval_qa_pydantic({"query": query})
```

Figure 25: Define the Langchain class needed to perform our query on our provided PDF document and with the specified output types

We then execute our query as we did in the previous section, and we return the model's response. Here, we notice that the response is now not a string, but a list containing the class we defined earlier.

```
r['result'].data
```

```
[EnergyUsage(type='Electricity', region='Total', year=2021, quantity=2854000.0, unit='MWh'),
 EnergyUsage(type='Electricity', region='U.S.', year=2021, quantity=2377000.0, unit='MWh'),
 EnergyUsage(type='Electricity', region='International', year=2021, quantity=477000.0, unit='MWh'),
 EnergyUsage(type='Fuel', region='Total', year=2021, quantity=467280.0, unit='MWh'),
 EnergyUsage(type='Natural gas', region='Total', year=2021, quantity=203010.0, unit='MWh'),
 EnergyUsage(type='Biogas', region='Total', year=2021, quantity=208620.0, unit='MWh'),
 EnergyUsage(type='Propane liquid', region='Total', year=2021, quantity=40.0, unit='MWh'),
 EnergyUsage(type='Gasoline', region='Total', year=2021, quantity=34880.0, unit='MWh'),
 EnergyUsage(type='Diesel (other)', region='Total', year=2021, quantity=9780.0, unit='MWh'),
 EnergyUsage(type='Diesel (mobile combustion)', region='Total', year=2021, quantity=10950.0, unit='MWh'),
 EnergyUsage(type='Other Steam, heating, and cooling', region='Total', year=2021, quantity=22480.0, unit='MWh'),
 EnergyUsage(type='Renewable electricity', region='Corporate facilities', year=2021, quantity=2854000.0, unit='MWh'),
 EnergyUsage(type='Renewable electricity capacity (operational)', region='Supply chain', year=2021, quantity=10.3, unit='GW'),
 EnergyUsage(type='Renewable electricity capacity (committed)', region='Supply chain', year=2021, quantity=15.9, unit='GW'),
 EnergyUsage(type='Renewable electricity', region='Supply chain', year=2021, quantity=18100000.0, unit='MWh'),
 EnergyUsage(type='Electricity', region='Data centers and colocation facilities', year=2021, quantity=1960000000.0, unit='kWh'),
 EnergyUsage(type='Renewable electricity', region='Maiden, North Carolina', year=2021, quantity=392000000.0, unit='kWh')]
```

Figure 26: Final output

Therefore, we can easily access the different variables of this class, and store them in a database.

This example seems extremely interesting to us because it shows that if we know what we are looking for in a document, this information can be automatically extracted and structured. We can imagine automating this process on a larger number of reports, trying to find patterns in the documents, and using them to extract the information that interests us.

### 6.4 Solving Data Privacy Issues

In the current digital age, issues surrounding data ownership are of paramount importance across various sectors. The need for stringent data control and adherence to privacy norms often precludes the use of APIs such as OpenAI, where the handling and usage of the data sent remain uncertain.

One of the significant concerns is the potential for sensitive data to transit on American soil, which could pose risks for certain companies whose data is deemed critical. This concern is not unfounded, as evidenced by the 2015 case where BNP was fined nearly 9 billion dollars by the American Department of Justice (DOJ) for non-compliance with embargo decisions. The United States, due to the use of the dollar in international transactions, considers it has the right to legislate on these issues, even if the offenses are not committed on its territory. The same principle could potentially apply to data, raising concerns about data sovereignty and privacy.

To address these data privacy issues, the use of open-source models can help mitigate data privacy concerns. As these models are publicly available, they can be scrutinized and vetted by the community for any potential data privacy issues. Furthermore, they can be run locally, providing full control over the data and eliminating the need for data to be sent over the internet.

Hence, while data privacy issues are a significant concern when using models such as GPT, they can be effectively addressed through a combination of technical measures, policy implementation,

and the use of open-source models.

In the next two sections, we'll take a closer look at the use of Open Source model to create embeddings and to chat.

### 6.4.1 Self Hosted Embeddings

Utilizing an API for creating embeddings offers several advantages, making it an appealing option. APIs provide a convenient and user-friendly interface for generating embeddings, eliminating the need to develop and maintain complex embedding models from scratch. They offer a standardized and optimized solution, saving time and effort in the development process.

Moreover, using an API allows for seamless integration with other services and applications. By leveraging an API, developers can quickly incorporate embedding functionality into their own systems without worrying about the underlying implementation details. This promotes scalability and encourages collaboration, as multiple teams or projects can utilize the same API to generate embeddings.

However, self-hosting embeddings brings its own set of benefits that make it worth considering. By hosting the embeddings locally, you eliminate reliance on an external API, ensuring consistent availability and mitigating the risk of potential service disruptions or changes in API specifications. This level of control reduces the chance of vendor lock-in, providing flexibility to adapt and modify the embedding generation process as per specific requirements.

Self-hosting also addresses concerns related to data privacy and security. Hosting embeddings locally ensures that sensitive data remains within your own infrastructure, reducing exposure to potential breaches or unauthorized access. This level of data control is particularly crucial in scenarios where confidentiality is paramount, such as handling proprietary or confidential information.

Furthermore, self-hosting embeddings allows you to explore and incorporate state-of-the-art models that may not be available through external APIs. By having the freedom to choose and update the embedding model, you can leverage the latest advancements in the field, potentially leading to improved performance and accuracy tailored to your specific needs.

In summary, while using an API for creating embeddings offers convenience and ease of integration, self-hosting provides benefits like independence from external dependencies, better data privacy and security, and the ability to leverage cutting-edge models. Ultimately, the choice between the two approaches should be driven by the specific requirements, constraints, and priorities of your project or organization.

Integrating a model to create embeddings locally can be done relatively easily [28], as most models are not very large [29], and can run not only on GPUs but also on CPUs. What's more, a vector database like Weaviate enables rapid integration [30].

### 6.4.2 Self Hosted LLM

Local LLM management requires access to GPUs. Unless we have access to GPUs, which can be very expensive as we've seen, we need to use GPU providers. The best-known are of course AWS, Azure or GCP. However, other providers do exist, such as Lambda or Runpod, which offer access to GPUs at much lower prices.

While self-hosting a large language model (LLM) can be costly, it becomes a viable option when there is sufficient demand. However, open-source model that can be used as a foundational LLM, on which we can fine-tune for specific tasks, and which matches the required parameters can be challenging. Despite these challenges, self-hosting an LLM provides the advantage of full control over the data, thereby addressing data privacy concerns. It also allows for customization of the model to suit specific needs, potentially leading to improved performance.

Unlike model embedding hosting, which can be done quite easily, and whose constraints are more to do with model specifications (context length, multi-language or not), LLM hosting can be more challenging, especially when you want to use the largest models like Llama 2 70B. This requires either the use of a large A100 80B graphics card with a quantized model (even a slightly smaller model can work) or several A100s running in a cluster. To make them easier to use, new libraries enable us to quickly launch these models from our infrastructure, so we can perform inference and serve several users at the same time. What's more, these libraries also enable us to benefit from inference optimizations, for faster responses. We'll limit ourselves here to one example, using the Vllm library, which makes it easy to deploy this type of model. Another popular library for this task is the Huggging Face library.

It is relatively straightforward to use open-source LLMs today, and their quality has significantly improved with the release of LLama 2 models. These can be used on Google Colab (specifying the use of the GPU) or with a GPU provider. In the code below, we demonstrate how to import a Llama-2 13B parameters model to generate an answer to a question. It's worth noting that the chosen model is a "quantized" model. This means that the model's weights are compressed to reduce the model's

[28] The Sentence Transformer is the recommanded way of using Open Source embedding models.
[29] The Hugging Face Leaderboards allows to easily find Open Source embedding models.
[30] Weaviate allows to easily use Open Source embeddings model from the Sentence Transformer. For more information, this link how to implement those models.

size, allowing us to use larger models while staying within our GPU's memory limits. Here, we see that we can easily test a Llama-2 13B on a GPU with 16GB of memory.

```python
from transformers import AutoTokenizer, AutoModelForCausalLM, TextStreamer

model_id = "TheBloke/Llama-2-13B-chat-GPTQ"
model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto")
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

```python
prompt = """### User:
Explain what is the CSRD and why it is important for investors ?

### Assistant:
"""
inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
streamer = TextStreamer(tokenizer, skip_prompt=True, skip_special_tokens=True)

output = model.generate(**inputs, streamer=streamer, use_cache=True, max_new_tokens=float('inf'))
output_text = tokenizer.decode(output[0], skip_special_tokens=True)
```

Figure 27: Import Open Source Quantized Llama 2 - 13B parameters

```
The CSRD, or the Corporate Sustainability Reporting Directive, is a new European Union regulation that requires l
arge companies to disclose information on their environmental, social, and governance (ESG) performance. The dire
ctive aims to improve transparency and accountability of companies in terms of their sustainability practices and
to provide investors with better information to make informed decisions.

The CSRD is important for investors for several reasons:

1. Better information: The CSRD requires companies to provide more detailed information on their ESG performance,
which can help investors assess the sustainability of a company's business model and its potential risks and oppo
rtunities.
2. Improved transparency: The CSRD promotes transparency and accountability of companies, which can help to build
trust and confidence among investors.
3. Risk management: The CSRD can help investors to identify potential risks and opportunities associated with a c
ompany's sustainability practices, which can inform their investment decisions.
4. Long-term thinking: The CSRD encourages companies to adopt a long-term approach to sustainability, which can h
elp to align their interests with those of investors and contribute to the long-term success of the company.
5. EU-wide consistency: The CSRD applies to all EU member states, which can help to ensure consistency and compar
ability of sustainability information across the EU.

Overall, the CSRD is an important development for investors as it can help to improve the transparency and accoun
tability of companies, provide better information, and support long-term thinking and sustainable investing.
```

Figure 28: Model output

Despite these models being much smaller than GPT-3 or GPT-4, they are still highly efficient and can provide interesting results for certain use cases.

However, one point to highlight is that to use these models and make them available outside the server running the model, it is necessary to create an API endpoint. A simple solution for this is to use the Vllm library [31]. This allows you to create an API in a few lines of code and deploy it on a server. Additionally, it offers inference optimization, which reduces the inference time, i.e., the token generation time, for larger models. It also allows serving multiple users simultaneously in a

[31] The documentation explains how to easily start a server optimized for inference. See: Link

distributed manner. Another significant advantage is that it enables the creation of an API that mocks that of OpenAI. This makes it possible to easily switch between an OpenAI model and an open-source model, without having to change the application code that uses the API.

We have seen that it can be relevant to want to host one's own language model, whether it is for the generation of embeddings or in the case of an LLM. We have also seen that this is ultimately quite simple due to the large number of tools available to us today. We will now conclude this section with the lessons learned from working with these models and the difficulties encountered. We will also see what all the criteria that are to consider when choosing a language model.

## 6.5 Lessons learned and best practices

In this section, we'll be asking ourselves how to choose our model, what criteria to consider and what trade-offs we might encounter when implementing these tools. This will enable us to summarize the main information needed to choose the right model and look at the prerequisites for deploying a model, and how to test, monitor and improve it. All those steps are essential to ensure production grade quality.

This process refers to MLOps. MLOps, an abbreviation for Machine Learning Operations, is a fundamental aspect of Machine Learning engineering. Its primary role is to simplify the procedure of deploying machine learning models into production, followed by their maintenance and monitoring. MLOps promotes a collaborative environment, typically involving a team of data scientists, DevOps engineers, and IT professionals. This practice ensures the efficient integration of machine learning technology into real-world applications, thereby enhancing the overall performance and reliability of these systems.

➤ ***Choosing a base model***

The choice of a base LLM is influenced by several factors and trade-offs. These include the quality of the model, the extent of knowledge required, the cost involved, the latency and response time of inference, the degree of fine-tuning the model allows, the context length or the number of tokens the model can handle in memory, concerns about data privacy, the choice between a proprietary model or an Open Source one, and licensing permissions, especially for Open Source models.

| Factor | Description |
|---|---|
| Model Quality | The level of model quality required for the task at hand |
| Knowledge Required | The extent of knowledge required |
| Cost | The cost involved |
| Latency | The latency and response time of inference |
| Fine-tuning | The degree of fine-tuning the model allows |
| Context Length | The context length, or the number of tokens the model can handle in memory |
| Data Privacy | Concerns about data privacy |
| Proprietary vs Open Source | The choice between a proprietary model or an Open Source one |
| Licensing | Licensing permissions, especially for Open Source models (ranging from permissive licenses like Apache 2.0, to restricted licenses like CC BY-SA 3.0, or non-commercial licenses like those of Facebook and the Llama model) |

Table 2: Factors to consider when choosing a base Language Model (LLM)

Currently, GPT-4 is widely regarded as the most efficient model. However, it may not be the best choice for all use cases due to its slower response times and potentially expensive API requests. Faster models might be more suitable for certain applications.

As we saw before, one of the primary reasons a company might hesitate to use this model is due to concerns about intellectual property and data privacy. The risk of private or sensitive data leakage can be a significant deterrent for companies, making the use of an external model a potential issue.

➤ ***Assess LLM 's performance***

Following Chang et al. [2023], the evaluation of LLMs can be divided into separate dimensions: what to evaluate, where to evaluate, and how to evaluate. These dimensions encompass various categories, including natural language understanding and generation, multilingual, factual, mathematics and scientific reasoning, robustness, ethics, biases, and trustworthiness, specific domain applications, generic benchmarks, specific benchmarks, automatic evaluation, and human evaluation.

| Evaluation Dimension | Category |
|---|---|
| What to evaluate? | Natural Language Understanding, Natural Language Generation, Multilingual, Factual, Mathematics and Scientific reasoning, Robustness, ethics, biases, and trustworthiness, Specific Domain Applications |
| Where to evaluate? | Generic Benchmarks, Specific Benchmarks |
| How to evaluate? | Automatic evaluation, Human evaluation |

Table 3: Evaluation of a base LLM model

➤ ***Prompt management***

The main issue with prompts is that while they often yield better results on a given task, they can lead to worse results on other tasks. Therefore, it is essential to track and monitor prompts, either using Git or a specialized tool.

➤ ***Testing LLM outputs***

In traditional ML models, it is easy to test, because we can compare the different metrics (depending on the task, regression, classification). However, testing LLM outputs can be challenging as the quality of the output generated by the model is hard to evaluate since it can depend on the context, individual preferences, or the intentions of the user. While some metrics or benchmarks can be used (BLEU, ROUGE), they're not able to grasp the human intentions.

Thus, there is no perfect quantitative metrics for generative models since they're mostly qualitative output. Therefore, it is essential to have many tests in place to check the quality of the models and users testing the model output before making changes in production.

➤ ***Build an evaluation dataset for a given task***

Creating a small dataset and getting output for your task using a model can be beneficial. LLMs can also be used at this stage to generate test cases. More data, or different data, feedback from users can be used to improve the model.

➤ ***Deployment***

Deployment can involve calling an external API or hosting a model directly, often for privacy reasons. In either case, it is likely necessary to build an API to serve the application for the LLM responses. We saw in the case studies that the Vllm library is a good fit for that.

➤ ***Monitoring***

Monitoring involves tracking various aspects such as latency, outcomes, user feedback, performance metrics if available, hallucinations, prompt injection attacks, and prompts with user query and context. It is also important to identify optimization opportunities.

➤ ***Fine Tuning***

Fine tuning may not be necessary when using GPT 4, but it can be extremely useful when using smaller models or if we have lots of data and we want better results on a specific task. Fine tuning is relatively inexpensive, but it requires access to some GPU.

This part has enabled us to see how we can interact with these LLMs. LLMs are complex objects and require a very wide range of considerations. We've seen how it's possible to use the very large number of tools available today to interact with these models. Langchain for processing all kinds of data formats, application creation and interaction with different APIs, and Vllm for inference and to create API endpoints when deploying a self hosted LLM. This has enabled us to show that it is possible to set up new automation processes for data extraction, but also for the use of Open Source models. And finally, we have seen the main points to consider when choosing a model, as well as the importance of testing and monitoring the architecture in place. Before concluding this thesis, we'd like to take a quick look at the major trends at work today, which may lead to future changes in our experience of LLMs as we know them today.

# 7 Future trends and implications

As we delve into the future of Large Language Models (LLMs) and their implications, it is crucial to understand the current landscape and the challenges that these models face. LLMs have revolutionized the field of Natural Language Processing (NLP) with their ability to generate human-like text. However, they are not without their limitations, such as the quadratic complexity of the attention layer, the lack of positional information in embeddings, high cost of inference and a strong tendency to hallucinate. This section aims to explore the future trends and implications of LLMs, focusing on their evolution, the role of quantization, and the concept of augmented LLMs.

## 7.1 LLM future evolutions

➤ ***Context Length***

The context length of LLMs is a significant factor in their performance. The quadratic complexity of the attention layer means that doubling the context length requires four times more memory and computation time. This is why LLMs are often trained on relatively low context lengths. However, advancements in hardware have allowed for an increase in context length, facilitated by GPUs with greater VRAM. Despite this, the cost of running these models also increases with the GPU cost.

Several techniques have been developed to improve context length, such as ALIBI (Press et al. [2022]), Landmark Attention (Mohtashami and Jaggi [2023]), Rotary Position Embedding (Su et al. [2022]), and Dilated attention (Ding et al. [2023]). These techniques have their strengths and weaknesses, and their effectiveness varies.

However, these techniques for increasing the context length of LLMs are sometimes called into question. For example, Liu et al. [2023] has shown that LLMs perform best when relevant information is at the beginning or end of the input context and that the performance significantly degrades when LLMs must access relevant info in the middle and decreases as the input context grows longer.

➤ ***Quantization***

Quantization is another area of focus in the evolution of LLMs. We've already seen that quantization can significantly reduce the memory size of LLMs, enabling very large models, such as a Llama 2 70b of parameters, to be loaded onto a single GPU (Dettmers et al. [2023]). These optimization techniques will certainly also enable these models to be used on ever-smaller machines, or on machines without a powerful GPU, using CPU power alone.

However, these techniques must demonstrate their ability to reduce the memory usage of LLMs without impacting too negatively their output and the inference process.

➤ ***Inference***

Inference is a critical aspect of LLMs. Faster inference can lead to significant improvements in the performance of these models. Researchers from Microsoft (Yang et al. [2023b]) have developed methods to allow parallelism at inference time, leading to up to a 100% speed increase. Inference is still not cheap when a model is self-hosted, as the required GPUs can be expensive and would require enough usage to cover the costs.

➤ ***Hallucinations and Reasoning***

Despite their potential, LLMs are not without limitations. These include hallucinations, limited information access, flawed reasoning, and insufficient context for generating responses. Nevertheless, these challenges can be mitigated by enhancing the model's reasoning capabilities and equipping it with tools for better information access.

➤ ***Augmented LLM***

In the enhancement of Large Language Models (LLM), several key components come into play. These include *retrieval*, which allows the model to access unseen data, *chains*, which involve further augmentation through additional LLM calls, and *tools*, which incorporate external resources for augmentation.

In "Augmented Knowledge Survey", Mialon et al. [2023], call "tools" a module that will be used to obtain external information to which the model does not have access in its weights. For example, a document retrieval tool (to search for information in a PDF, like we did in our case studies) or the ability for the model to call an API, use a code interpreter (which already exists with GPT4) or search and navigate through the web.

The advantage of augmented LLMs is that they are designed to be more accurate, thus reducing the likelihood of hallucinations. They also help decrease uncertainty and enhance model reasoning, as the model learns to decompose complex tasks more efficiently (also known as Chain of Thoughts, Wei et al. [2023]) and employs the appropriate tools for the task at hand. This also makes the model more interpretable, as it can explain its reasoning process.

# 8 Conclusion

This thesis has delved into the transformative capabilities of AI, NLP, and LLMs in the context of ESG data processing and analysis. We have dissected the core principles of AI, NLP, and Semantic Search, and the advancements in deep learning that have given rise to LLMs. Our exploration of potential use cases has demonstrated the capacity of AI and LLMs to automate ESG data extraction, augment data analysis, and enable more efficient ESG risk assessment.

The case studies presented offer a practical perspective on the application of AI and LLMs in ESG data processing, shedding light on successful implementations, key learnings, and best practices, while also acknowledging the obstacles and limitations encountered. As we gaze into the future, it is clear that AI and LLMs will increasingly influence ESG data processing, bringing with them both opportunities and challenges.

It is important to note that while LLMs show promise, they are not without their challenges. One significant issue is their propensity for hallucinations, which can lead to inaccurate or misleading outputs. This is a critical concern in ESG data processing where precision is paramount. However, functions provided by OpenAI have shown potential in mitigating this issue by ensuring appropriate output formatting. Therefore, while LLMs are promising, a cautious and meticulous approach is necessary to prevent hallucinations and ensure accurate ESG data processing.

In summary, AI, NLP, and LLMs present a promising avenue for revolutionizing how ESG data is processed, analyzed, and incorporated into financial and investment decision-making. As this field continues to evolve, it is crucial to confront the ethical and practical challenges that may surface, and persistently pursue new research and development pathways to fully harness the potential of these technologies in improving ESG data processing and fostering a more sustainable and responsible financial environment.

# References


European Commission. The european green deal. 2019. URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52019DC0640.

Anne Demartini. La fourniture de données extra-financières : cartographie des acteurs, produits et services, 2020.

Florian Berg, Julian F Kölbel, and Roberto Rigobon. Aggregate Confusion: The Divergence of ESG Ratings*. Review of Finance, 26(6):1315–1344, 05 2022. ISSN 1572-3097. doi: 10.1093/rof/rfac033. URL https://doi.org/10.1093/rof/rfac033.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013a.

Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation, 2013b.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation, 2014. URL https://aclanthology.org/D14-1162.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, 2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond, 2023a.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. Data-centric artificial intelligence: A survey, 2023a.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017.

Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023a.

Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, and Xia Hu. Data-centric ai: Perspectives and challenges, 2023b.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. 2023.

Rohan Taori et al. Alpaca: A strong, replicable instruction-following model. Data Set, 01 2023b.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality, 2023.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning, 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A survey on evaluation of large language models, 2023.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation, 2022.

Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers, 2023.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2022.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, and Furu Wei. Longnet: Scaling transformers to 1,000,000,000 tokens, 2023.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. Inference with reference: Lossless acceleration of large language models, 2023b.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.