Masters Thesis

Optimizing Large Language Models for Structured ESG Data Extraction Through Domain-Specific Fine-Tuning

Author
Rok Rozman

Supervisors
Prof. Dr. Svetlana Borovkova, Dr. Olga Shevchuk

Second reader
Prof. Dr. Sandjai Bhulai

Master's of Business Analytics
Vrije Universiteit Amsterdam

Amsterdam, 24th of October, 2024

Abstract

This thesis explores the optimization of large language models (LLMs) for the structured extraction of Environmental, Social, and Governance (ESG) data from unstructured texts, particularly annual reports. Given the increasing importance of ESG factors in assessing a company's value and performance, the study focuses on developing and fine-tuning domain-specific models to automate the extraction of relevant information, which has traditionally been a manual and labor-intensive process.

The research involves the evaluation of various open-source LLMs, such as Meta's LLaMA and Mistral models, as well as a commercial model (GPT-4o-mini) for benchmarking. A dataset containing ESG data from 30 annual reports was created using a Retrieval-Augmented Generation (RAG) method, enhanced by manual verification to ensure accuracy. The study's key contributions include the development of an automated framework that streamlines the ESG data extraction process, requiring only the input of a PDF document. The thesis also examines the impact of fine-tuning models with domain-specific knowledge, comparing their performance to base versions.

Experimental results show that fine-tuning significantly improves the models' ability to extract structured ESG information, with open-source models demonstrating competitive performance against commercial alternatives. Techniques such as Parameter Efficient Fine-Tuning (PEFT) and Quantized Low-Rank Adaptation (QLoRA) were employed to optimize resource usage during the fine-tuning process. The findings suggest that with appropriate adaptation, LLMs can effectively automate the extraction of ESG data, offering a scalable solution for businesses and analysts seeking to analyze large volumes of annual reports.

Contents

List of Figures

Chapter 1

Introduction

.

Annual reports have long been a crucial part of a company's public relations strategy. These reports not only showcase the company's past achievements but also communicate its future goals. In addition, they include comprehensive financial statements that enhance transparency and accountability. Through the annual report, readers gain a holistic understanding of the company, enabling them to form well-informed opinions. Therefore, it is essential for companies to present themselves effectively, as this shapes the perceptions of both investors and the general public.

In recent years, the scope of annual reports has expanded significantly. Beyond traditional financial statements and operational reviews, modern reports often include detailed discussions of a company's environmental, social, and governance (ESG) initiatives, risk management strategies, and long-term sustainability plans [35, 23]. This expansion highlights the growing importance of non-financial factors in assessing a company's value and overall performance. As a result, financial analysts have begun incorporating ESG factors into models for determining stock prices. Research has shown that, on average, a company's ESG performance can impact stock prices by as much as 5% [52]. Given the valuable information contained in annual reports, performing a detailed analysis is critical.

A significant challenge stems from the lack of a standardized format for annual reports. Since the content and structure of these reports are determined by management, regulations allow for considerable flexibility in their organization and presentation. As a result, automating the analysis of annual reports is difficult, which is why it is often still done manually. This presents an opportunity to develop methods that can overcome these challenges, enabling automated analysis of a large volume of reports more efficiently and accurately.

One promising approach is to extract key ESG information from annual reports. Since annual reports are considered unstructured text, the process of extracting this information is categorized under Information Extraction (IE). IE is a subfield of Natural Language Processing (NLP) that includes tasks such as Named Entity Recognition (NER), Event Extraction (EE), and Relation Extraction (RE).

NLP is inherently complex. However, recent advancements in the field, particularly with the development of large language models (LLMs), have significantly improved many tasks, including IE. LLMs are primarily designed for text generation, but they have shown potential for IE tasks, even though they are not specifically built for tagging tasks like NER. As a result, there is growing interest in leveraging LLMs to generate structured data, moving beyond traditional IE methods.

One of the key advantages of LLMs is their ability to perform a wide range of IE tasks without the need for task-specific training data. By providing a well-defined task description, LLMs can extract entities, relationships, and events from text by utilizing their broad language understanding capabilities. This makes them highly efficient for extracting information from documents like annual reports, which contain large volumes of text. However, there is a wide variety of LLMs available, each with different capabilities. For instance, open-source models like Meta's Llama-3 [1] are available for free, while commercial models like OpenAI's GPT-4 [2] require paid access. Although commercial models tend to perform better, open-source alternatives show promising results and are being developed at a rapid pace. Open-source models benefit from the contributions of the research community, and many of these models are hosted on platforms like Hugging Face [3], which currently hosts more than 700,000 models [34].

Given the growing number of available models, choosing the most suitable one for specific IE tasks, such as summarizing and analyzing text, becomes important. It is therefore valuable to evaluate the performance of different models on these tasks and select the most appropriate one. However, there are currently no models specifically designed for extracting ESG-related data from annual reports. This creates an interesting opportunity to explore the development of an improved model by fine-tuning an existing model on ESG data from annual reports. Fine-tuning would equip the model with enhanced domain knowledge, potentially yielding better performance on ESG-specific tasks compared to using base models alone.

[1] https://about.meta.com
[2] https://openai.com
[3] https://huggingface.co

Chapter 2

Objective and Related Works

The focus of this thesis is to test and compare the performance of open-source LLMs on the task of extracting structured information from unstructured text. Specifically, we focus on extracting ESG data from annual reports. Additionally, we examine how these models perform on the same tasks when fine-tuned on annual report data, providing them with the necessary domain knowledge, and compare their performance to base versions. We also use a commercial model, `gpt-4o-mini`, as a benchmark for comparison.

One of the key contributions of this study is the creation of a dataset containing extracted ESG data from 30 annual reports, which was constructed using an LLM enhanced with the Retrieval Augmented Generation (RAG) [27] method and manually verified for accuracy. However, the main contributions are the empirical evaluation of these models alongside their fine-tuned counterparts, as well as the development of a fully automated framework that streamlines the ESG data extraction process, requiring only the annual report PDF as input.

2.1 Related works

There have been many attempts to automate IE in the past. However, most research papers focus on only one specific task within IE, such as NER or RE. This is largely due to the inherent complexity of IE and the need for traditional rule-based methods to be highly specialized for each task and tailored to specific document types to be successful [37].

With the introduction of LLMs, it has become possible to combine multiple IE tasks within a single framework. Despite this advancement, most papers utilizing LLMs still focus on a single task, with NER being the most common. A notable

domain in which LLMs have been applied is materials science, where the goal is to extract materials data from scientific papers [15, 41, 56, 72, 67].

In this domain, both Dunn et al. [15] and Song et al. [56] employed fine-tuning of LLMs to create efficient and high-performing models. In [15], the authors used a GPT-3 [6] model and fine-tuned it on 500 prompts and completion pairs. Their approach focused on extracting information from both sentences and entire paragraphs, demonstrating that their method performs NER in combination with RE with high accuracy. The extracted information was provided in the form of a JSON object, which is also the approach adopted in this study.

On the other hand, Song et al. [56] used the open-source LLaMA [59] model. They developed a two-step approach, where in the first step they used Chat-GPT[1] to create instruction-based data in the materials science domain, and in the second step, this data was used to fine-tune the LLaMA model. This allowed the model to acquire the necessary domain knowledge to perform a wide array of materials science-based tasks.

Another approach for extracting materials data from scientific papers was proposed by Polak et al. [42]. Instead of fine-tuning, they employed a simple zero-shot prompting technique, where the model is tasked with extracting information without any prior examples or task-specific training. They enhanced this framework by validating responses through a series of follow-up questions, which significantly improved the results. The best-performing model was GPT-4 [39], which outperformed both GPT-3.5 [6] and the LLaMA-2 [60] 70B version.

LLMs have also been applied to IE tasks in the medical domain. In the work by Li et al. [28], a framework was proposed using Google's open-source PaLM-2 [2] model. To enhance the model with clinical domain knowledge, they employed in-context learning (ICL) using two knowledge bases. ICL refers to a method where the model's parameters remain frozen, and the LLM performs the task solely based on the prompt text. One knowledge base was generated internally using an LLM with a small labeled training set, while the other was externally curated by experts. Li et al. focused on extracting lung lesion information from clinical and medical imaging reports, with their approach outperforming techniques such as few-shot learning [6], chain-of-thought (CoT) [66], and RAG [27] in terms of F1-score [55].

Prompt-based methods have also been explored for structured extraction from unstructured text in other domains. For instance, Vijayan [63] developed a method to extract trip information from unstructured text such as emails. They used Google's

[1]https://platform.openai.com/docs/api-reference/chat

BARD [18] model, structuring the extracted information in the form of custom SQL queries. The authors compared four prompting techniques: standard prompting, persona pattern prompting, CoT prompting, and few-shot prompting [6]. Persona pattern prompting involves providing the LLM with a persona to follow while performing a task. Their findings indicated that few-shot prompting significantly outperformed the other techniques, achieving an F1-score of 0.86.

As the importance of ESG has grown, so too has the research surrounding it. Most of the research has focused on predicting price changes in equities based on ESG ratings using machine learning techniques [10, 61]. However, research focused specifically on extracting ESG data remains scarce [33]. Raman et al. [46] classify text in earnings calls as either ESG-relevant or not. Their two-fold approach involved first creating a classification model to categorize text in Corporate Sustainability Reports (CSRs) into three categories: irrelevant, quasi-relevant, and relevant. This was done in an unsupervised manner, combining a model with human annotators who corrected errors. They then fine-tuned BERT [13], XLNet [69], RoBERTa [29], and DistilBERT [51] on the constructed dataset and tested the models on a hold-out set.

Similarly, Fischbach et al. [16] developed a method that automatically classifies news media data as ESG-related or not, and further performed sentiment analysis to classify relevant news as positive or negative. They compared several traditional machine learning models to a deep-learning model, BERT, which predictably outperformed the others.

The research most similar to our study was conducted by Zou et al. [73], who combined an LLM with RAG to extract ESG-related data from ESG reports. They required the model's output to be in a predefined JSON format. The preprocessing of the reports was extensive, employing advanced computer vision tools to extract paragraphs, tables, and headers, and performing structural analysis. Their framework also included a metadata module to ensure the model's output complied with international ESG standards. They compared four models: GPT-4, GPT-3, ChatGLM [14], and QWEN [4], with GPT-4 achieving the highest accuracy at 83.7%, outperforming QWEN, which achieved 61.4%.

Chapter 3

Methods

In this chapter, we present the key concepts and methods used in the implementation and optimization of LLMs for information extraction. First, we begin by discussing the deep learning models that served as precursors to modern LLMs. Then, we explain the fundamentals of language modeling. Next, we introduce the core components of LLMs and their functionality. Finally, we explore how LLMs can be adapted to specific tasks through fine-tuning techniques, with a focus on addressing computational resource limitations.

3.1 Deep learning models

The core of all deep learning architectures is the *feedforward neural network* (FNN), which is a specific type of general neural networks (NNs), also referred to as artificial neural networks (ANNs). The inspiration for NNs came from modeling how the human brain works, and the invention of the perceptron by Rosenblatt [48] in 1958 marks the beginning of NN research. The next major breakthrough occurred in 1986 when Rumelhart et al. [50] applied the backpropagation algorithm to NNs, enabling the training of more complex and deeper models compared to the earlier, more limited perceptrons.

A significant factor in the resurgence and overall success of NNs, particularly in deep learning models, is the availability of large datasets for training, along with advancements in computational power. A general NN architecture consists of an input layer, one or more hidden layers, and an output layer. The number of nodes in each hidden layer is flexible, while the number of nodes in the input layer depends on the size of the input data, and the number of nodes in the output layer depends on the specific task (e.g., regression or binary classification). Connections between nodes, representing the weights, are also flexible. However, in FNNs, the layers are

fully connected, meaning that each node in one layer is connected to all nodes in the subsequent layer.

A simple example of an FNN is presented in Figure 3.1. It consists of an input layer with four nodes, two hidden layers with six and five nodes, respectively, and an output layer with three nodes. The network processes the input data, denoted as x_1, x_2, x_3, and x_4. These input values are propagated forward through the network via multiplication with the weight matrix, and at each node, a non-linear activation function is applied.

To train such networks, the backpropagation algorithm is used to adjust the weights based on the error between the predicted output and the actual target. Backpropagation works by propagating this error backward through the network and updating the weights using gradient descent to minimize the loss function.

This architecture has many impressive applications, including character recognition, speech recognition, text-to-speech transformation, process control, associative memory, and more. However, FNNs require fixed-size input and lack memory capacity, making them less effective for tasks involving sequential data or tasks where dependencies between input features must be modeled.

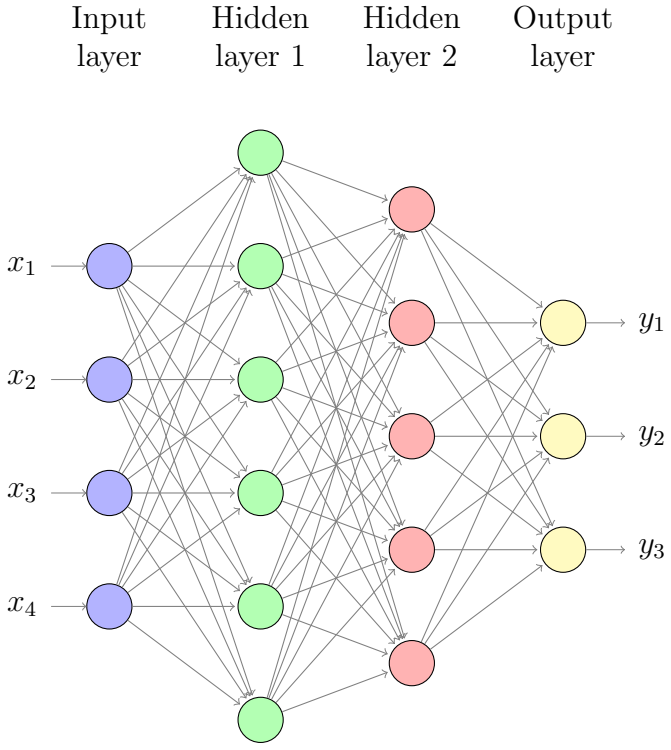


Figure 3.1: Feedforward Neural Network with Four Layers

In response to the fixed input size limitation of FNNs, recurrent neural networks

(RNNs) were invented. RNNs were designed to handle sequential data of varying lengths, such as text sentences or time series. Memory is retained through the hidden state, which holds information about previous inputs. However, RNNs struggle to capture long-range dependencies due to the vanishing gradient problem, where the gradients used to update the model's weights during training become extremely small. This leads to earlier inputs having little to no influence on the model's output in lengthy sequences.

Long short-term memory networks (LSTMs) were developed to address this issue. LSTMs are an improved type of RNN that use gates, which are mechanisms that control what information is allowed to pass through, what should be updated, and what should be discarded, to regulate the flow of information. This allows them to retain memory for longer periods, making them more effective for tasks such as language modeling and text generation. However, LSTMs still have limitations. The first is that the memory is presented as a fixed-sized vector, which limits the amount of information that can be stored. Additionally, LSTMs require more data and computational power because they process inputs sequentially, making it difficult to leverage parallel computation and utilize GPUs to speed up training. Furthermore, LSTMs can only recall past information, leading to the development of bidirectional LSTMs (BiLSTMs). BiLSTMs use two LSTMs: one processes the input from left to right, while the other processes it from right to left. The outputs of both LSTMs are then merged using a model or function (e.g., linear-chain conditional random field).

Another type of deep learning network is the convolutional neural network (CNN). While primarily used for image processing, CNNs can also be applied to NLP tasks. CNNs capture local patterns and model spatial relationships in input data, making them effective for tasks like text classification. They can also leverage GPU parallelism through their feedforward structure. However, CNNs cannot capture long-range dependencies, limiting their usefulness in many NLP tasks.

In 2017, a new deep learning architecture called the transformer was introduced by Vaswani et al. [62]. Transformers are highly parallelizable and can capture long-range dependencies, solving the main challenges faced by RNNs and CNNs. The transformer architecture revolutionized NLP and became the foundation for most state-of-the-art (SOTA) models used today. More specifically, the introduction of Bidirectional Encoder Representations from Transformers (BERT) by Devlin et al. [13] achieved SOTA results on 11 different tasks, according to the benchmarks and datasets used in the original paper.

The models discussed in this thesis are all based on the transformer architecture, which will be described in depth in Section 3.4. However, before delving into transformers, we will define some core concepts of language modeling, beginning with the definition of a language model in Section 3.2. Most of the notation used follows [62, 20].

3.2 Language Modeling

Language models are statistical models that try to learn to model the joint distribution of the sequence of words, based on the data [5]. A statistical language model can be expressed as the conditional probability of the next word given the sequence of preceding words, where the probability of a word sequence $\mathbf{w} = (w_1, \ldots, w_n)$ is given as:

$$P(\mathbf{w}) = \prod_{i=1}^{n} P(w_i \mid w_1, \ldots, w_{i-1}). \tag{3.1}$$

Models that use only the preceding words in a given sequence to predict the next word are called autoregressive models [71]. These are the kind of models we use in this work and more specifically they work on the level of substrings of words, which we define in Section 3.3.

3.3 Tokenization

In natural language processing tasks, the input is naturally provided as a string of characters. However, the models described in the following chapters require a numerical representation of the input data. Therefore the input data is converted from a long string of characters to the sequences of precisely defined substrings of characters to which the model assigns a precise numerical representation. This process is called tokenization [53]. The result of tokenization is the breaking down of a string into shorter substrings, called tokens. The set of all allowed tokens is called a vocabulary. A sequence of tokens is assigned a sequence of natural numbers such that each token is assigned a token identifier from the set $\mathcal{V} = \{1, \ldots, v\}$, where $v = |\mathcal{V}|$ is the size of the vocabulary. In the following sections we will use the term token as both the string of characters it consists of and the number it is represented by.

A simple choice for tokenization is to break the string at the character or byte level. In this case, each token corresponds to exactly one symbol. The advantage of

such a representation is the ability to encode arbitrary strings, but the drawback is the need for a large number of tokens to represent longer strings.

An alternative option for tokenization would be at the word level, where each token corresponds to a word. The problem with this is the need for a large vocabulary and the limitation to only pre-determined words. The advantage of such a representation is the use of fewer tokens to represent the input data.

A method that combines the advantages of character level and word level tokenization is subword tokenization [44]. Subword tokenization represents frequent sequences of characters with individual tokens and rare occurrences of sequences with multiple tokens. The Byte Pair Encoding (BPE) algorithm [53] builds the vocabulary from the input training corpus by starting with a vocabulary composed of all the different bytes or symbols in the training corpus. Then it calculates the number of occurrences of all two consecutive symbols in the data and the pair with highest number is represented with another symbol and the vocabulary is expanded by 1. This is repeated until it cannot be compressed anymore or by the predetermined number of final tokens, which is a hyperparameter of the algorithm.

3.4 Transformer Architecture

The transformer neural network architecture [62] has revolutionized the field of natural language processing with its architecture that enables efficient parallel learning on sequential data GPUs. A key component of the architecture is the attention mechanism, which allows the model to consider the interactions between input tokens and is described in Section 3.4.2.

The transformer architecture consists of an encoder and a decoder, as shown in Figure 3.2. On the left side of the image we have a encoder and on the right we have an decoder. The encoder maps the sequence of tokens $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{V}^n$ into a sequence of continuous representations $\mathbf{z} = (z_1, \ldots, z_n)$, where $z_i \in \mathbb{R}^d$ for $i = 1, \ldots, n$, is a hyperparameter of dimension $d \in \mathbb{N}$. The decoder then uses $\mathbf{z}$ and, in an autoregressive manner, produces the output sequence of tokens $\mathbf{y} = (y_1, \ldots, y_m)$ or probabilities for each token $y_i \in \mathbb{R}^{|\mathcal{V}|}$ for $i = 1, \ldots, m$. The input sequence $\mathbf{x}$ is also referred to as the input context of the model.

3.4.1 Vector Embeddings

The input to the transformer is a sequence of tokens $\mathbf{x} \in \mathcal{V}^n$, which is first mapped into a sequence of continuous representations called token embeddings. This is done

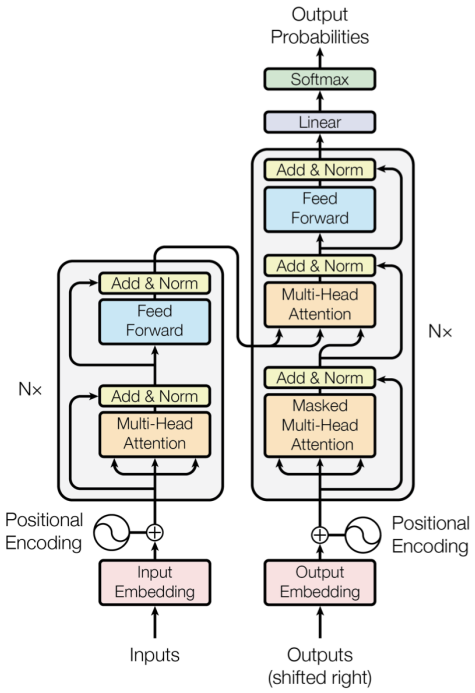


Figure 3.2: Transformer Architecture: Encoder on the left and Decoder on the right (Source Vaswani et al. [62])

in the *Input Embedding* layer of the transformer, as shown in Figure 3.2. Each token $x_i \in \mathcal{V}$ is mapped to a vector $\hat{\mathbf{x}}_i \in \mathbb{R}^d$ using a mapping table. The mapping table can be represented by a matrix $\mathbf{U} \in \mathbb{R}^{|\mathcal{V}| \times d}$, where the i-th row contains the vector embedding of the i-th token from the vocabulary $\mathcal{V}$. The weights of the mapping table $\mathbf{U}$ are learned through neural network training.

Within the layers of the encoder and decoder, the transformer operates with token embeddings of dimension d, and the output of the last decoder layer is also a vector of dimension d. The output vector from the last decoder layer is mapped linearly with $\mathbf{V} \in \mathbb{R}^{d \times |\mathcal{V}|}$ into a vector of dimension $|\mathcal{V}|$, and then the softmax operation $\text{softmax} : \mathbb{R}^{|\mathcal{V}|} \to (0,1)^{|\mathcal{V}|}$ is applied to obtain probabilities for the next token. The softmax output ensures that the probabilities sum to one, which is crucial for stability, as it guarantees a valid probability distribution. Comparable results can be obtained by sharing weights between the input and output mapping tables, setting $\mathbf{U} = \mathbf{V}$ [43, 62], which also has the added benefit of reducing the number of parameters.

Before using the token embeddings in the encoder and decoder, they are enhanced with positional information within the sequence. This is important because transformers lack a built-in mechanism to model the order of tokens in a sequence. Positional encoding enables the model to incorporate the relative or absolute posi-

tions of tokens, which is essential for understanding the sequence's structure. This can be done using either absolute or relative positions [58], and the positions can be fixed [62] or learned [17]. In the paper by Vaswani et al. [62], they propose using sine and cosine functions to compute the positional encoding with the following formula:

$$\begin{aligned} \mathrm{PE}_{\mathrm{pos},2i} &= \sin\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d}}}\right), \\ \mathrm{PE}_{\mathrm{pos},2i+1} &= \cos\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d}}}\right), \end{aligned} \tag{3.2}$$

where pos is the position, and i is the dimension of the token. The positional vector of dimension d is then added to the token embedding vector x_i before the resulting continuous representation is used for processing within the encoder and decoder.

3.4.2 Attention

A key component of the transformer architecture is the attention mechanism, which allows the model to consider the interactions between tokens in the input sequence. Attention enhances the representations of tokens by incorporating information from other tokens in the sequence. Greater weight, or attention, is given to tokens that provide useful information for predicting the next token.

The attention mechanism first linearly maps the token embeddings $\mathbf{X} \in \mathbb{R}^{n \times d}$ into queries $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$, keys $\mathbf{K} = \mathbf{X}\mathbf{W}^K$, and values $\mathbf{V} = \mathbf{X}\mathbf{W}^V$, using learned weight matrices $\mathbf{W}^Q \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}^K \in \mathbb{R}^{d \times d_k}$, and $\mathbf{W}^V \in \mathbb{R}^{d \times d_v}$ for given dimensions of keys and queries d_k, and output values d_v. Based on the match between queries and keys, a weighted sum of the value vectors is calculated. The version of attention that uses scaled dot-product for matching queries and keys is written as:

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_k}}\right)\mathbf{V}, \tag{3.3}$$

where $\mathbf{K}^{\mathrm{T}}$ is the transposed matrix of $\mathbf{K}$, and the softmax operation normalizes the match between queries and keys to compute the weighted sum of the value vectors for each query.

The attention mechanism is extended to multi-head attention, which allows each attention head to focus on different learned representations of queries, keys, and values [62]. Multi-head attention operates by performing attention on h different linear projections of the query, key, and value vectors, where each *head* is computed as:

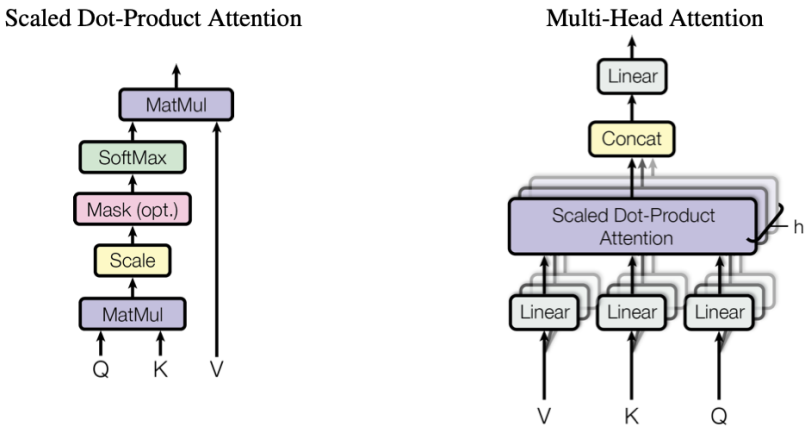


Figure 3.3: Scaled Dot-Product Attention on the left and Multi-Head Attention on the right (Source Vaswani et al. [62])

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V), \tag{3.4}$$

where $\mathbf{W}_i^Q \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d \times d_k}$, and $\mathbf{W}_i^V \in \mathbb{R}^{d \times d_v}$ are linear projections learned during training. The outputs of each attention head are then concatenated into vectors of dimension $h \cdot d_v$ and mapped with a learned linear projection $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d}$ back to the original dimension d. Therefore, multi-head attention is defined as:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h). \tag{3.5}$$

In Figure 3.3, we can see a graphical representation of how these methods are combined.

3.4.3 Encoder

The encoder is composed of a sequence of N identically structured layers. Each encoder layer contains a multi-head self-attention mechanism and a fully connected feed-forward neural network. Additionally, for learning stability, layer normalization [68] and residual connections [19] are included. The dimension of the individual input and output vectors from each layer is equal to the hyperparameter d. Attention within the encoder receives queries, keys, and values from the output of the previous encoder layer, with each position focusing attention on all other positions. Since queries, keys, and values originate from the same source in the attention calculation, such attention is called self-attention.

3.4.4 Decoder

Similar to the encoder, the decoder is composed of a sequence of N identically structured layers. Unlike the encoder, each decoder layer includes two attention mechanisms. The first self-attention mechanism operates on the outputs of the previous decoder layer, but the attention of each position is directed only to itself and previous positions, ensuring autoregressive properties. The second attention mechanism is cross-attention between the encoder and decoder, hence called cross-attention. Cross-attention receives queries from the previous decoder layer, while values and keys come from the encoder outputs, allowing each position to focus attention on the entire encoder output.

3.4.5 Fully Connected Feed-Forward Neural Network

Each layer of the encoder and decoder contains a fully connected feed-forward neural network (FFN), which is applied to each position independently and simultaneously. An example of an FFN used in transformers is expressed by equation 3.6, which includes two linear transformations with a ReLU activation function in between:

$$\mathrm{FFN}(\mathbf{x}) = \max\left(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1\right)\mathbf{W}_2 + \mathbf{b}_2, \tag{3.6}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are matrices of learned weights, and $\mathbf{b}_1$ and $\mathbf{b}_2$ are learned bias vectors. The ReLU function is defined as $\mathrm{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$, where max is applied component-wise. In transformers, the dimension of the input vector $\mathbf{x} \in \mathbb{R}^d$ is equal to the dimension of the output vector $\mathrm{FFN}(\mathbf{x})$, while the intermediate dimension within the network is typically $4d$ [62], denoted as d_{ff}. In modern architectures [60, 8], other non-linear functions, such as SwiGLU [54], are also used instead of ReLU.

3.5 Large Language Models

Building on the foundation of the transformer architecture discussed in the previous section, this design has proven to be versatile and applicable to a wide range of tasks in natural language processing (NLP) [3]. By making specific modifications to the architecture, models can be tailored to perform better on particular tasks. For instance, architectures that use only a decoder are well-suited for autoregressive text generation tasks, where each word or token is generated sequentially based on the previously generated content. Examples of decoder-only models include the Generative

Pre-trained Transformers (GPT) series developed by OpenAI, such as GPT-1 [70], GPT-2 [44], GPT-3 [6], and GPT-4 [39].

Conversely, the encoder is responsible for converting raw input into a contextual representation, and architectures that rely solely on the encoder are used for text understanding tasks. These models can compute vector embeddings for multiple words or sentences, and are also effective in tasks such as sentence classification and named entity recognition (NER). Examples of such models include BERT [13], RoBERTa [29], and ELECTRA [9].

Encoder-decoder architectures, on the other hand, are ideal for tasks that require both text understanding and generation, such as translation or summarization. Models with this architecture include T5 [65] and BART [26].

It is important to note that the use of the transformer architecture in any encoder-decoder configuration does not automatically classify a model as a large language model (LLM). What distinguishes LLMs is the sheer number of parameters, ranging from several billion to over a trillion. Additionally, LLMs have the ability to perform a wide range of NLP tasks without requiring task-specific fine-tuning. This capability eliminates the need for large labeled datasets and reduces the computational cost, as LLMs can be guided to perform tasks via prompting. One of the most well-known examples of an LLM is the GPT series by OpenAI, particularly ChatGPT. ChatGPT is based on InstructGPT [40], with additional safety features. InstructGPT itself evolved from GPT-3 by using Reinforcement Learning with Human Feedback (RLHF) to fine-tune it for more human-like writing. While GPT-3 has 175 billion parameters, InstructGPT has only 1.3 billion parameters.

3.5.1 Open Source LLMs

While OpenAI's models, such as GPT-4, are among the leading general-purpose language models available today, they are not open-source. Access to these models requires payment for API usage, and using them often involves sharing data with OpenAI, which can pose challenges for organizations with strict data privacy requirements. To address these concerns, several high-quality open-source models have been developed, offering competitive performance while allowing for local deployment, thereby reducing the need for external data sharing.

A notable example is Meta's LLaMA (Large Language Model Meta AI) series, which offers impressive performance and can be used locally, mitigating data privacy concerns. One of the newest models in this series, LLaMA-3 [1], was introduced on

April 18th, 2024, and comes in three sizes. The smallest version has 8 billion parameters, the middle has 70 billion, and the largest has 405 billion. All of these models achieve state-of-the-art (SOTA) performance across multiple tasks when compared to similarly sized models.

Hugging Face is another key player in the NLP and machine learning community, providing a platform with a wide range of open-source models, including those developed by organizations like Meta and MistralAI[1], as well as contributions from researchers and individuals. In this thesis, we will use the Hugging Face API to download the models and datasets required for our project, leveraging this rich ecosystem of open-source tools.

3.6 Fine-Tuning

There are several ways to improve the performance of LLMs on specific tasks. One of the simplest methods is prompt engineering, which involves refining the input prompt given to the model. Techniques such as one-shot learning and few-shot learning have emerged as promising approaches within prompt engineering, allowing models to adapt to new tasks with minimal training data [36]. In these approaches, one or a few solved examples of the task are included in the prompt. Another form of prompt engineering is providing a highly detailed description of the task. For instance, techniques like Chain-of-Thought (CoT) prompting have been shown to significantly improve performance on NLP tasks [22, 32].

Fine-tuning, by contrast, involves updating the model's parameters to specialize it for a particular task or domain. While this approach is more complex, it typically yields the best results when executed correctly, which is why we have chosen it for this study. However, fine-tuning requires substantial computational resources and a large dataset, making it resource-intensive and less accessible for all applications.

3.6.1 Quantization

While fine-tuning can yield highly accurate models, it often requires substantial computational resources, which may not always be available. To address this issue, several techniques have been developed to reduce the computational load, one of the most common being model quantization. Quantization reduces the size of the model by lowering the precision of the model's weights, switching from full 32-bit floating point precision to a more compact, discrete set of values [24].

[1] https://mistral.ai

Typically, open-source LLMs are trained using 32-bit precision, which allows for highly accurate representations of weights. However, by reducing this precision to lower bit-widths (e.g., 16-bit, 8-bit, or even 4-bit), the model's memory requirements and computational demands are significantly reduced. Importantly, research has shown that quantization does not substantially degrade model performance, even though the precision of the weights is lowered [11]. This makes quantization a popular method for deploying large models more efficiently, especially in environments with limited computational power.

As a result, many open-source models available on platforms like Hugging Face are provided in quantized forms, allowing users to download and use models that are much smaller in size while maintaining high performance.

3.6.2 Parameter Efficient Fine-Tuning (PEFT)

While quantization helps reduce the size and computational load of large language models (LLMs), further optimizations can be made when fine-tuning these models. To additionally save computational power during the fine-tuning process, Parameter Efficient Fine-Tuning (PEFT) methods have been introduced. These methods offer an alternative to traditional fine-tuning, where all of a model's parameters are updated. In the case of LLMs, traditional fine-tuning can be extremely resource-intensive. In contrast, PEFT methods reduce the number of parameters involved in fine-tuning, leading to faster and more efficient adaptation.

There are several PEFT methods, including Adapter, LoRA, QLoRA, and Prompt Tuning [24]. In this study, we focus on LoRA and QLoRA, as these are the methods used in our approach. Both techniques are explained in detail in Section 3.6.2.1.

3.6.2.1 Low Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA) represents a significant advancement in the fine-tuning of LLMs. Instead of modifying the entire large weight matrix of a pre-trained model, often denoted as $\mathbf{W} \in \mathbb{R}^{d \times d}$, LoRA modifies smaller, compact matrices of lower rank. The changes to the parameters of $\mathbf{W}$ that occur during fine-tuning are represented by two matrices: $\mathbf{A}$ and $\mathbf{B}$. Matrix $\mathbf{A}$ reduces the dimensionality from d to r, while matrix $\mathbf{B}$ expands it back to d. Specifically, $\mathbf{A} \in \mathbb{R}^{r \times d}$ and $\mathbf{B} \in \mathbb{R}^{d \times r}$, and when r is much smaller than d, the total number of parameters for $\mathbf{A}$ and $\mathbf{B}$ is significantly smaller than in the original $\mathbf{W}$ matrix.

During fine-tuning, the original large matrix $\mathbf{W}$ is frozen. Matrix $\mathbf{A}$ is initialized with random values sampled from a Gaussian distribution with a mean of 0 and small

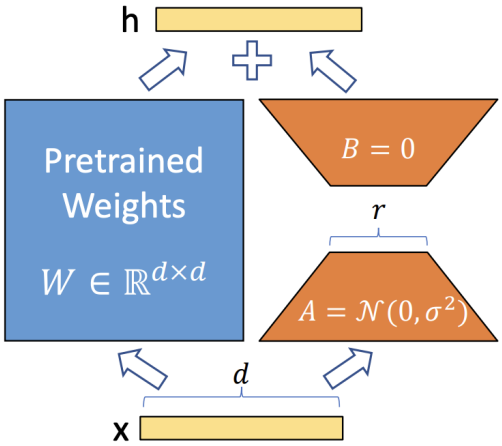


Figure 3.4: Low-Rank Decomposition of Weight Matrix into Matrices **A** and **B** (Source: Hu et al. [21])

variance, while matrix **B** is initialized to zeros. The LoRA adapter, denoted by $\Delta\mathbf{W}$, is defined as $\Delta\mathbf{W} = \mathbf{BA}$. A forward pass is then computed as:

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \Delta\mathbf{W}\mathbf{x} = \mathbf{W}\mathbf{x} + \mathbf{B}\mathbf{A}\mathbf{x}, \tag{3.7}$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input vector. Initially, because **B** is zero, the LoRA adapter does not affect the model's output. However, as training progresses, the values in **B** evolve from zero, and the random values in **A** adjust through backpropagation. This leads to an increasingly effective contribution from $\Delta\mathbf{W}$ on top of the original **W**.

LoRA can be applied to any weight matrix within a transformer model to reduce the number of trainable parameters. Additionally, using small values of r, even as low as 4, has been shown to yield better fine-tuning results than fine-tuning the entire **W** matrix. This suggests that pre-trained models may be over-parameterized, and their essential features can be captured more efficiently in a reduced form.

3.6.2.2 Quantized Low-Rank Adaptation (QLoRA)

Quantized Low-Rank Adaptation (QLoRA) combines the principles of LoRA with quantization. In this method, the precision of the model's weights is reduced to 4-bit precision, further lowering computational requirements during fine-tuning. This additional reduction in precision, when combined with LoRA's low-rank approximation, results in even more efficient fine-tuning. Due to its computational efficiency, QLoRA is implemented in this thesis as a key part of our model fine-tuning strategy.

3.6.3 Supervised Fine-Tuning

The PEFT methods discussed earlier can be applied to various types of fine-tuning, which are typically distinguished by the nature of the data available. The two primary types of fine-tuning are supervised and unsupervised, with a third hybrid approach called semi-supervised fine-tuning, which combines elements of both.

Supervised fine-tuning is a method where the model learns from a dataset containing specific input-output pairs, such as question-and-answer datasets. In this approach, the model is given an input, such as a question, and is trained to produce the corresponding correct output, or answer. The model's predictions are compared with the actual answers, and its parameters are adjusted to minimize the difference, or loss, between them. This targeted approach is highly effective for training models to focus on specific tasks, as it ensures the model learns the exact relationship between input and output.

A key component of supervised fine-tuning is the use of attention and loss masks. The attention mask helps the model focus on relevant parts of the input sequence while ignoring irrelevant information, such as padding tokens used to standardize sequence lengths. The loss mask ensures that the model is only penalized for incorrect predictions related to the answer portion, rather than the question. This focused learning process is crucial for training the model to produce accurate and relevant outputs without being distracted by extraneous input details.

Another important aspect of supervised fine-tuning is teaching the model when to stop generating text. This is typically achieved by training the model to predict an "end of sequence" (EOS) token after the correct answer, preventing it from generating unnecessary information beyond the intended output.

During supervised fine-tuning, the model is typically exposed to the entire dataset multiple times, a process known as training epochs. More than three epochs are generally required to ensure the model adequately learns from the data. However, care must be taken to avoid overfitting, where the model becomes overly specialized to the training data, reducing its ability to generalize to new, unseen inputs. Achieving the right balance in the number of training epochs is key to developing a model that performs well both on the training set and in real-world applications.

3.6.4 Unsupervised Fine-Tuning

In contrast to supervised fine-tuning, unsupervised fine-tuning does not rely on paired input-output datasets. Instead, the model learns by predicting the next word in a

sequence of text, which is then compared to the actual next word in the sequence. The model's parameters are adjusted based on the accuracy of these predictions.

Like supervised fine-tuning, unsupervised fine-tuning employs an attention mechanism, enabling the model to consider the entire sequence of preceding words when making its predictions. This ensures that the model effectively uses the context provided by the input text. The key difference lies in how the loss is calculated. In unsupervised fine-tuning, loss is calculated for every word in the sequence, rather than just for specific output portions as in supervised fine-tuning. As a result, every word contributes to the model's learning, making this process more generalized.

One of the main advantages of unsupervised fine-tuning is its flexibility. It does not require the data to be in any specific format, allowing large, unstructured collections of text, such as annual reports, to be used for training. However, this flexibility also presents a challenge: without the structure provided by explicit input-output pairs, the model may struggle with tasks that require structured outputs, such as generating JSON-formatted answers or conversational responses in a chat format.

Despite these challenges, unsupervised fine-tuning is used in this study due to the lack of access to a structured dataset. By fine-tuning the model on a large corpus of annual reports, the model is expected to learn the tone and style of these documents. We hypothesize that this will improve the model's ability to extract relevant ESG information and enhance the quality of the generated responses.

Chapter 4

Data

In this chapter, we provide a comprehensive overview of the data utilized in this thesis. We begin by detailing the acquisition and pre-processing of the initial raw data. Next, we explain the methodology employed to transform the raw data into a structured dataset, which plays a crucial role in evaluating our models. Finally, we offer a detailed description of the dataset and present an overview of the entire process, illustrated with a diagram for clarity.

4.1 Original Data

In real-world applications, data is frequently unstructured and lacks a consistent format. Therefore, to make this project as realistic and useful as possible, our starting point was a collection of annual reports in PDF format, as they are commonly published by companies. These reports were sourced from either the companies' websites or from `AnnualReports.com` We selected annual reports from different companies for the year 2023, while ensuring variability across industries. This diversity in the dataset allows us to better simulate real-world conditions and assess how the models would perform in practical applications.

Annual reports are generally extensive documents, with lengths ranging from around 100 to several hundred pages. Although they follow an overall structured format, there is considerable flexibility in how specific sections are organized. Typically, annual reports provide a comprehensive overview of a company's financial performance, operations, and future prospects. They usually begin with a letter to shareholders from the CEO or board chair, summarizing the year's achievements and challenges. This is followed by a detailed management discussion and analysis, where key financial results are interpreted, including performance indicators, market conditions, and strategic goals. The core of the report consists of financial statements,

such as the balance sheet, income statement, and cash flow statement, supplemented by detailed notes.

In recent years, an Environmental, Social, and Governance (ESG) section has become a common feature of annual reports. This section outlines the company's commitments to sustainability, ethical practices, and corporate governance. It often addresses topics such as environmental impact, diversity and inclusion efforts, as well as ethical business conduct.

Given the increasing interest from both the public and investors in ESG issues, we decided to focus on extracting data that would be most relevant to investors and researchers. As a result, we set our goal to answer the following two key questions:

- What are the key elements of the company's ESG (Environmental, Social, and Governance) or sustainability policy, including specific commitments or initiatives?

- What is the full name of the senior executive, board member, or employee explicitly identified as responsible for overseeing the company's ESG or sustainability initiatives?

These questions were chosen after consultations with ESG experts and by drawing from reputable sources, such as the report on ESG boardroom questions [7], to determine the information most valuable to investors. Identifying the individual responsible for overseeing ESG efforts is crucial, as it not only provides a clear point of accountability but also allows investors to evaluate the individual's expertise and track record in driving ESG initiatives. Furthermore, providing a detailed breakdown of the key components of the company's ESG policy, including specific commitments or initiatives, such as governance, environmental goals, or alignment with the Sustainable Development Goals (SDGs), is essential. For example, the report encourages companies to assess whether they are providing robust sustainability information and whether their boards have the skills necessary to guide ESG efforts effectively [7].

We compiled a collection of 30 annual reports from companies across various industries and countries, all published in the year 2023. This dataset includes reports from global financial institutions such as Citigroup (CITI) [1], Goldman Sachs (GS) [2], and JPMorgan Chase (JPM) [3], as well as numerous companies listed on the London

[1] https://www.citi.com
[2] https://www.goldmansachs.com
[3] https://www.jpmorganchase.com

Stock Exchange (LSE) [4]. These reports are stored in PDF format and vary in size from 3.2 MB to over 33 MB, which reflects the complexity and depth of information they contain. This dataset provides a comprehensive foundation for in-depth analysis across different sectors and regions.

In addition to answering the two key questions outlined above, we aimed to structure the extracted information in a format that is useful for further analysis and database creation. We have chosen the JSON format for this purpose, as it allows us to organize the data in a clear and hierarchical manner. Specifically, the JSON structure we require from our models consists of an answer to each of the two question as well as the page number from where the answer came from. This is crucial because it enables individuals seeking to extract ESG information to link each answer to its source within the document, providing the ability to double-check the extracted information. To this end, we also decided to create a dataset for testing the performance with the same structure. Hence, each answer in our dataset has that same JSON form. However, the annual reports are extensive documents with on average 230 pages, therefore we decided to create this dataset in a semi automated way, with the help of an LLM. This requires us to have a pre-processing step, which is described in Section 4.2, that enhances the performance of the LLM.

4.2 Data Pre-Processing

The pre-processing step is crucial for improving the LLM's ability to accurately identify the page where the relevant information is located. Although all annual reports include page numbers, these are sometimes unclear or obscured during the conversion from PDF to TXT format. This happens because page numbers get combined with other text or numbers during the conversion process, which makes it difficult to determine the exact page number.

To address this issue, our pre-processing step involves two steps. In the first step, we convert the PDF into TXT format using the `PyPDF2.PdfReader` function from the `PyPDF2` library. This function iterates through each page of the document, extracting the text and appending it to a string variable.

In the second step, we insert clear page markers to indicate the start and end of each page. Specifically, we add markers such as "START OF PAGE" and "END OF PAGE" (e.g. "START OF PAGE 46" and "END OF PAGE 46"). The page numbers correspond to where a specific page appears in the PDF document. Since

[4] https://www.londonstockexchange.com

page numbering in the report may not always start at PDF page one, these numbers can differ. However, because the answers are manually verified, this method allows for faster navigation when searching for a particular page to assess the accuracy of the extracted information.

Additionally, we use regular expressions to correct misplaced newline characters that may be improperly attached to words, ensuring that the text is formatted correctly and consistently.

4.3 Dataset Creation

After obtaining a cleaned and improved version of the annual report in TXT format, the next step is to extract the answers to the key questions along with the corresponding page numbers. We decided to use the OpenAI API, specifically its `gpt-4o-mini` [5] model, in combination with the Retrieval Augmented Generation (RAG) method to extract the data. We chose this approach because OpenAI provides some of the best LLMs currently available, and `gpt-4o-mini` is relatively cost-effective to use. Furthermore, given the size of the annual reports, it is impossible to input the entire document as part of the prompt, making RAG an ideal solution.

The RAG method enhances the LLM's ability to generate answers by supplying it with specific external documents not included in its training data. This allows the LLM, which has the ability to perform tasks based on a user query, to use external information to answer detailed questions about the document. This approach is especially valuable for real-world applications, such as equipping chatbots with comprehensive knowledge about a company. To make the document usable by the LLM, we employ a technique known as embedding a language model. First, we tokenize the data and create vector representations of it. When the LLM receives a query, the query is also converted into a vector representation, which is then matched against the vector database of the document. This process retrieves relevant chunks of the document using a ranking algorithm such as BM25 Okapi [47], which the LLM incorporates into its answer.

For data extraction, we developed two scripts. The first script, `prepare.py`, is designed to prepare the data by segmenting it into chunks and generating corresponding embeddings using the OpenAI API. This chunking is necessary to manage the input size limitations during embedding generation, with each chunk set to a size of 500

[5] https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

tokens. Parallel processing is used to generate the embeddings for each chunk efficiently, optimizing the creation of the vector database. The embeddings and text chunks are then saved in the *data* folder.

The second script, `app.py`, loads the created embeddings (or vector database) and text chunks. It tokenizes the query, creates a vector embedding of it, and retrieves the relevant chunks using two ranking algorithms: `BM25 Okapi` [47] and `cosine similarity` [57]. We employ both algorithms because each excels in different scenarios, depending on the query and data, one may produce more relevant results than the other. By using both algorithms, we ensure that the most relevant chunks are retrieved. Specifically, we select the two most relevant chunks from each ranking algorithm, resulting in four chunks being used by the LLM to generate an answer. Additionally, we have two queries corresponding to the key questions we aim to address, as shown in Listing 4.1.

```
query_1 = """What are the key elements of the company's ESG
          (Environmental, Social, and Governance) or
          sustainability policy, including specific
          commitments or initiatives? """

query_2 = """What is the name of the senior executive or board
          member or employee responsible for overseeing the
          company's ESG (Environmental, Social, and Governance)
          initiatives?"""
```

Listing 4.1: Queries for ESG data extraction.

Since we are creating a dataset to assess the performance of the models, it is crucial to ensure that the data is absolutely accurate. For this reason, we require the model to output the exact text snippets from the relevant chunks, as well as the corresponding page numbers from the annual report where the snippets originate. Additionally, we want the model to provide the output in JSON format, as this makes it easier to read and subsequently correct if necessary. All of these instructions have been incorporated into the prompt, as shown in Listing 4.2.

```
system_prompt = f"""You are a research assistant. You are tasked
with answering questions based on the information provided in
the document excerpts. Please follow the JSON response
structure below:

<response format>
{{
  "response": "Your response here.",
  "citations": [
    {{
```

```
      "page_number": "Page number",
      "snippet": "Exact snippet from the document"
    }}
  ]
}}
</response format>

CRITICAL RULES FOR CITATIONS:

1. Each citation MUST be a word-for-word extract from the original
   text.
2. Citations MUST reflect the full sentence or phrase exactly as
   it appears.
3. Avoid using ellipses (...) or any form of abbreviation in
   citations.
4. DO NOT paraphrase or alter the original text for citations.
5. If your answer relies on multiple sentences, cite all necessary
   sentences fully.
6. Use as many citations as needed to ensure that your response
   is fully backed by the text.
7. Make sure each citation explicitly supports the specific part
   of your response.
8. Verify the page number associated with each citation. If unsure
   of the correct page, leave it blank.
9. If the provided documents do not contain relevant information,
   leave the page number and quote fields empty.

Example of a correct response:
{{
  "response": "John Doe",
  "citations": [
    {{
      "page_number": "12",
      "snippet": "John Doe is the senior executive overseeing the
      company's ESG strategy."
    }}
  ]
}}

Here are the relevant documents for your query:
<documents>
<Document Title: {Path(chunks_file).stem}>
"""
for i, chunk in enumerate(top_chunks):
    system_prompt += f"\n{chunk}\n"

system_prompt += f"""\n</Document Title: {Path(chunks_file).stem}
```

```
>\n </documents>"""

system_prompt += """
Remember:
1. Citations MUST use the exact wording from the source text.
2. Never modify, shorten, or paraphrase the original text.
3. Use several citations if needed to fully support your answer.
4. Ensure each citation is clearly relevant to its corresponding
   part of the response.
5. Be certain the page number is accurate.
6. Leave fields blank if the relevant information cannot be found
   in the provided texts.

Now, use the provided excerpts to answer the query, following
these guidelines, and format your response in JSON.
"""
```

Listing 4.2: Dataset creation system prompt with instructions and relevant document chunks.

In the prompt, we also include an example of a correct response, making this a one-shot learning task. This approach significantly improves the model's performance. Additionally, we provide the models with detailed rules on how to properly cite the text. These rules are listed at the beginning of the prompt and then repeated after the provided chunks. Since LLMs tend to focus more on the beginning and end of the prompt, therefore including the citation rules twice ensures that the model follows them correctly.

Finally, the prompt, along with the query, is passed to the `gpt-4o-mini` model. The model then generates a response, and to ensure the citations are accurate, we created a function called `verify_citations`. This function extracts the citations from the response, normalizes them, and matches them to the normalized chunks to check for exact matches. Additionally, we verify that the response is in the correct JSON format, as it is possible for the model to omit or misplace brackets or commas. For this, we use the `json.JSONDecodeError` function, which detects formatting errors. However, the `gpt-4o-mini` model consistently performed well, producing the correct JSON format in every case.

After the extraction process, we are left with two JSON objects, one for each query. An inspection is then conducted to verify whether the information is correct. Since the citations and page numbers are included, it is relatively easy to check the accuracy of the extracted information. However, the answers generated using the RAG method are not always complete. As a result, we often need to review most of the annual report to ensure the answers are thorough and comprehensive.

The biggest shortcoming of the RAG method was noticeable when extracting the relevant person's name. This is because, in annual reports, it often happens that the relevant person is mentioned by their title, e.g., CEO, CTO, or chair of the ESG committee, without explicitly mentioning their name. Therefore, when providing relevant chunks, the information about the name of the CEO, CTO, or chair of the committee is not included, which is why the model sometimes outputs only the title. On three occasions, it also happened that the name of the person was incorrect, and six times there was not a specific person but instead a committee, which the model was able to identify. However, it did not leave an empty string in the answer.

On the other hand, when extracting the ESG policies, the model performed very well. The information was correct and relevant every time. However, sometimes the model only focused on one part of the ESG policy, and in those cases, we had to complement the answer manually. While the citations were always correct, due to the verification process, the page numbers were accurate only around 60% of the time.

Even though the method we used is very successful in extracting information, there are possibilities for improvement. The first improvement would be to create better chunks that overlap with each other, ensuring no knowledge is lost, as well as experimenting with the optimal size of the chunks. Additionally, we could include more relevant chunks in the prompt, rather than just four. However, the biggest improvement to the method would be to input the model's answer back into the model and ask it to verify whether its answer is correct or makes sense, and to improve it if necessary. This, of course, adds additional costs, as it requires running the model again. We believe this could be a big improvement in the quality of the answer, because this would give the model or method the ability to reason on the initial answer.

We also have to note that in the first selection of the annual reports, we included the reports from ING [6] and Deutsche Bank [7], but we could not include them because their PDF reports were encrypted, and therefore the `PyPDF2.PdfReader` function was unable to process them. We also did not include the annual report from BlackRock [8] because their annual report had a very different structure and did not include the table of contents.

[6] https://www.ing.nl
[7] https://www.db.com
[8] https://www.blackrock.com

4.3.1 Dataset

After these exclusions, the final dataset includes three columns, each with 30 rows: *report_id*, *ESG_policy*, and *ESG_person*. The average number of policies for each annual report is six. This also translates to the answer on average having four sentences and 80 words. After manually examining the answers, we also realized that the policies are on average described on six pages.

The person responsible for ESG policies is identified in 24 annual reports. In three cases, there was neither a responsible person nor a relevant committee. Additionally, we determined that when the responsibility is distributed across various departments rather than assigned to a specific individual, we consider that as having no ESG responsible person. In the remaining three cases, although no specific name or title was mentioned, there was an ESG committee. However, we still categorized these instances as lacking a specific ESG responsible person.

4.3.2 Dataset Generation Diagram

The following diagram 4.1 illustrates the process of generating the dataset, highlighting each step from data collection to final inclusion in the dataset.

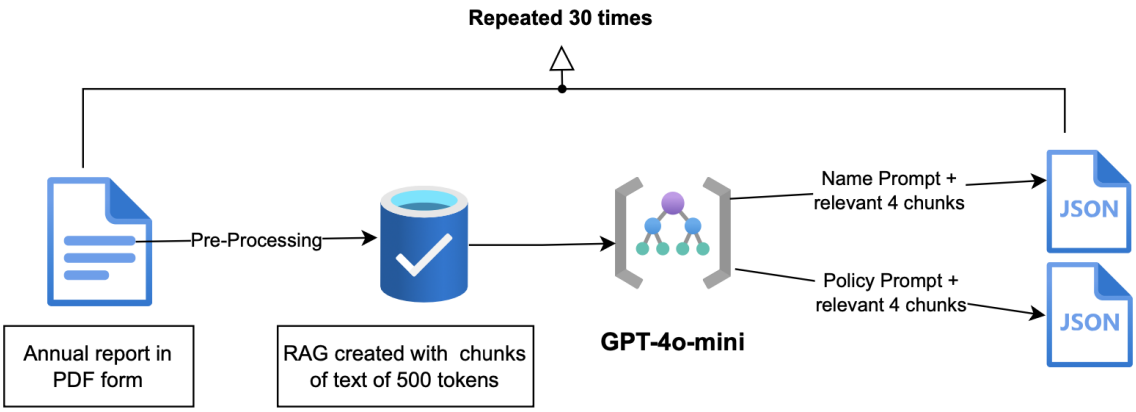


Figure 4.1: Schematic representation of dataset generation.

Chapter 5

Approach

In this chapter, we present the approach taken for the structured extraction of ESG data. First, we introduce the hardware used to run the models. We then describe the necessary data preparation steps required before inputting the text data into the model for extraction. Following this, we outline the base method, in which we use non-fine-tuned models to perform the extraction. Subsequently, we detail the steps involved in fine-tuning a model, including dataset generation and the parameters employed during the fine-tuning process. Additionally, we provide an overview of the models used in the study, covering both open-source models and the commercial model used as a benchmark. Finally, a crucial part of our approach involves the evaluation of the models, for which we develop a custom metrics tailored to our specific problem.

5.1 Hardware

Models used in this study are relatively large, which means that it would be difficult for a laptop's graphical processing unit (GPU) to handle. Therefore, we need a better GPU and for that we make use of the Google Colab [1], where you can upload a Jupyter Notebook and connect to the GPU that they offer. We chose the A100 GPU from NVIDIA, which has 40GB of GPU memory. The cost of using the A100 is 11.76 compute units per hour, which translates to around 1.2 euros per hour.

5.2 Input Data Pre-Processing Pipeline

As mentioned earlier, our computational resources are limited, which means we need to be efficient in both how many times we run the model and how much data we

[1] https://colab.research.google.com

input into it for extraction. For this reason, we pre-process the data before inputting it into the model. Since annual reports are elaborate documents and we are only interested in extracting ESG-related information, it is not necessary to process the entire report. Instead, we focus on extracting only the relevant pages. These relevant pages are identified by an open-source LLM based on the table of contents, after which they are extracted for further processing.

Before we perform the extraction of relevant pages, we again input the "[START OF PAGE]" and "[END OF PAGE]" markers. However, some annual reports do not start counting the pages from the first page of the PDF, but sometimes begin counting after the first three pages, which may contain logos or pictures of the company. This creates a problem since the table of contents references the numbered pages and not the PDF pages. Therefore, we need to input the page markers with the correct page numbers according to the table of contents. We achieve this by using the PDF's metadata, which contains information on where the actual page numbering begins.

To ensures that we include the page with the table of contents, we take all the pages of the report before the numbering starts, and then also the next three pages. Then, we input the pages into the OpenChat 3.5 [64] model, which is described in Section 5.5.4. We chose an open-source model because we want our framework to have the ability to run locally, without sharing any information with third parties. The goal of this pre-processing is to get the list of pages where the relevant information is located. However, asking the model to output the pages in a list has been unsuccessful. Therefore, we demand that the model again provides the answer as a JSON object. More specifically, because in the table of contents the chapters are marked with only the starting page, we achieved the best results by extracting the starting page of the relevant chapter and the starting page of the next chapter. In the prompt, included in Appendix A as Listing A.3, we give detailed instructions on how to do this task, along with an example output, which makes this a one-shot learning technique. An example of the outputted JSON object is given in Listing 5.1.

```
{
  "esg_related_sections": [
    {
      "start_page": 50,
      "next_section_start_page": 56
    },
    {
      "start_page": 62,
      "next_section_start_page": 64
    }
```

```
  ]
}
```

Listing 5.1: Example of the output JSON object for relevant page extraction.

From the JSON object, we then create a list of pages using the `json_to_list` function. This approach yielded excellent results, as the page list included all the pages containing relevant information based on the table of contents. Additionally, all the pages where we found the relevant information to create the dataset in Section 4.3.1 were also included in the page list.

In the final step of pre-processing, we use this list of pages to retrieve the actual relevant pages from the document in TXT format. However, the pages are not combined into one large text file. Instead, we group them in pairs. This is because LLMs have a limited context length, meaning there is a limit on how much text can be input into the model at once. While commercial models generally allow for a longer context, open-source models like Llama 2 have a maximum context length of 4096 tokens, which translates to about three to four pages of an annual report. However, models tend to perform best when the input is significantly less than the maximum context length, which is why we input only two pages at a time. Another reason for this is that LLMs often focus on either the beginning or the end of the input text. If the input is too long, the model might only extract information from the start or end, ignoring the middle. Therefore, inputting two pages strikes a balance between extraction quality and the time required to perform the task, as this method involves several separate extractions.

5.3 Base Method

The main part of this study is the extraction of the data with the open-source models, which are presented in Section 5.5. First, we only use the base model versions to perform the task and then we fine-tune them with text data from annual reports. Besides the open-source models we also use a commercial model, `gpt-4o-mini`, from OpenAI as the benchmark.

The extraction process using the base models is performed with the Jupyter notebook `base_model_extraction`, which is uploaded to Google Colab. The process begins by connecting to the Hugging Face ecosystem using our personal access token. This step is necessary because certain models on Hugging Face, such as the LLaMA collection from Meta, are gated and require prior approval from Meta to access and download. After establishing the connection, we install two key libraries, `transformers`

and `bitsandbytes`. The `transformers` library provides functions for downloading models, while `bitsandbytes` enables model quantization. To download the entire model repository to the local directory, we use the `snapshot_download` function from the `huggingface_hub` library.

The subsequent step involves setting up the quantization configuration, which is kept uniform across all models to ensure a fair comparison. Consistency in quantization configuration is crucial because altering the configuration of the base model parameters can directly impact performance. For instance, higher precision typically leads to improved model performance but at the cost of increased memory usage. We adopt 4-bit quantization. This is a common choice when fine-tuning with QLoRA, as it reduces the memory footprint by a factor of four. Specifically, we use the Normal Float 4 (NF4) data type, as recommended in the QLoRA paper [12]. Moreover, to conserve additional memory we apply the nested quantization technique, which performs a second quantization of the weights without any additional performance loss.

After that, we proceed by loading the model along with its corresponding tokenizer. Next, we configure the model's parameters. We have two parameters to consider: *temperature* and *top_p* (nucleus sampling). The *temperature* controls the level of randomness in selecting the next word during text generation, while *top_p* defines the range of possible words the model can choose from. In both cases the possible values lie between 0 and 1, where value closer to 1 translates to more randomness in the answer and vice versa. In this study we do not want randomness, because we want to have concrete answers that closely resemble the exact text from the annual report. Hence, we set the values for both parameters at 0.1, because we observed that the output of the models was not very sensitive to the parameter values, as long as they were close to 0.

Ultimately, we perform the extraction using the model by inputting the two annual report pages into the `user-prompt` prompt. For each query, we developed a separate prompt, both of which are included in Appendix A. Similar to the approach described in Section 4.3, we provide detailed instructions on how to extract the relevant information. Additionally, we supply an example of the desired JSON object output to guide the model's response. Specifically, for the extraction of the name of the person responsible for ESG policies, we require the output to follow the JSON format shown in Listing 5.2.

```
{
  "response": "Your_response_here.",
```

```
  "page_number": "Page_number"
}
```

Listing 5.2: The JSON format to extract the name of the individual responsible for overseeing ESG policies.

and for the relevant ESG policies, we demand the JSON form given in Listing 5.3.

```
{
  "response": {
    "key_elements": [
      "Element_1",
      "Element_2",
      "Element_3",
      "Element_4",
      "Element_5"
    ],
    "specific_policies": [
      "Policy_1",
      "Policy_2",
      "Policy_3",
      "Policy_4",
      "Policy_5"
    ]
  },
  "page_number": "Page_number"
}
```

Listing 5.3: The JSON format for extracting relevant ESG policies.

The reason why we decided to output a JSON object, like the one in Listing 5.3, for the extraction of policies, is because it is easier to evaluate the extraction. This is because we can check for each policy separately, if it is relevant or not. The *key_elements*, however, serve as indicators of which class of environmental, social or governance the policy falls under.

To give a clearer understanding of what we ask the models to extract, below is an example of a policy that falls under the environmental category:

- "Implement sustainable water management practices across all manufacturing sites, with a goal of reducing water usage by 30% by 2028. Additionally, wastewater recycling technologies will be deployed to minimize environmental contamination.“

Post-Processing

While the model is instructed to output the specified JSON format, it frequently includes additional text outside the desired JSON object. To address this, we perform

a post-processing step where we clean the output by removing unnecessary text. This is done using a script called `extract_JSON.py`, which utilizes regular expressions to isolate the correct JSON object from the rest of the text.

Once we have extracted all the JSON objects from the relevant page pairs, we proceed with combining the results. We use two functions, `combine_json_name` and `combine_json_policy`, to aggregate the extracted JSON objects into an output for further evaluation. The entire extraction process is displayed on Figure 5.1.

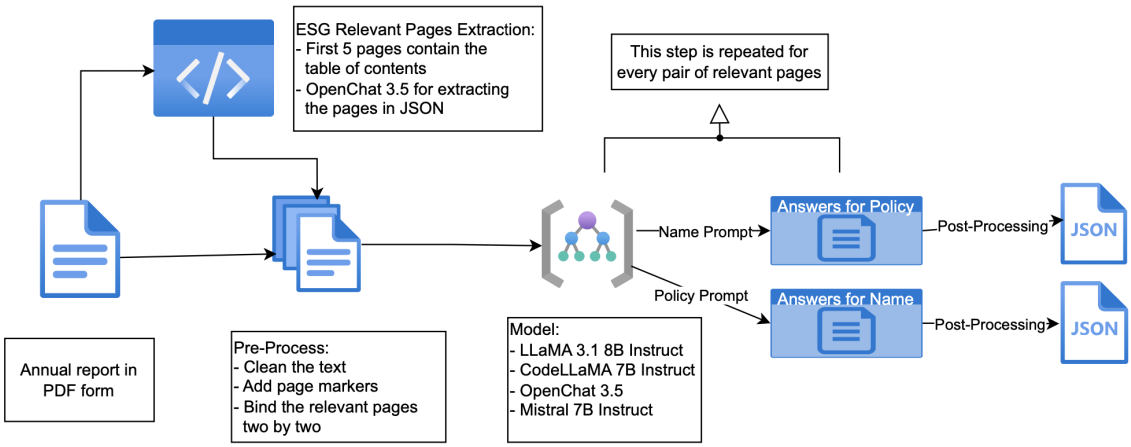


Figure 5.1: Schematic representation of the extraction process

5.4 Fine-Tuning Method

After the base method, we have a more advanced method that includes fine-tuning the base models. It requires several steps in between the downloading of the model and the final extraction. Besides the Hugging Face we also need to connect to another AI developer platform called Weights&Biases [2] with our personal token. This provides us with packages for tracking and visualizing the fine-tuning process, which enables us to determine the right fine-tuning parameters and model.

The first step that is specific to the fine-tuning of the model is to determine the LoRA parameters. The main goal of the fine-tuning process is to teach the models the language used in the annual reports. We do not want the models to learn the exact details and information from the reports, because this is not our objective since we want the model to perform well on extracting the data from the reports it has not seen before.

[2]https://wandb.ai/site

To avoid overfitting on the training data, we carefully selected the model parameters. Additionally, we do not have a large amount of data to fine-tune with, compared to the amount of data the models were pre-trained on. This limits how many layers of the model we can fine-tune, as well as the value of the rank. More layers and higher rank translate to more parameters to fine-tune, which can contribute to overfitting, if there is not enough data. Therefore, we chose suitable parameters for our study and they are found in Listing 5.4.

```
config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=[
              "q_proj",
              "k_proj",
              "v_proj",
              "o_proj",
              "gate_proj",
              "up_proj",
              "down_proj",
              ],
    modules_to_save=["embed_tokens","lm_head"],
    lora_dropout=0.1,
    bias="none",
    task_type="CAUSAL_LM"
)
```

Listing 5.4: LoRA parameters

We selected a rank of eight and the parameter *alpha* to be equal to 32. The parameter *alpha* influences the learning rate, which controls the step size during optimization, determining how quickly or slowly the model updates its parameters. Since LoRA adapters have fewer parameters, this allows for a higher learning rate. Therefore, it is common to set alpha $= 4r$, where r is the rank of the LoRA. The formula for calculating the LoRA *adapter learning rate* is presented in Equation 5.1.

$$\text{adapter learning rate} = \text{base learning rate} \cdot \frac{\text{alpha}}{r}. \tag{5.1}$$

An important parameter in our fine-tuning process is `lora_dropout`, which helps prevent overfitting by randomly setting a fraction of activations in the LoRA layers to zero during training. This regularization technique reduces the model's reliance on specific features in the training data, thereby improving its generalization ability. Selecting the right layers to fine-tune is equally crucial, as this choice directly impacts both the efficiency of fine-tuning and the model's ability to generalize to unseen

data. Although layer names can vary across models and can be viewed by using `print(model)`, all the models in our case share the same structure and layer names, which are listed in Listing 5.4.

We chose to fine-tune all the linear layers and apply LoRA adapters to them, with the layer names specified in the `target_modules` list. In addition, we decided to fine-tune both the embedding layer (`embed_tokens`) and the final output layer (`lm_head`), which are responsible for converting input tokens into vector representations and mapping the model's final hidden states to a probability distribution over the vocabulary, respectively. However, we opted not to apply LoRA adapters to these two layers, as avoiding them generally yields better performance when fine-tuning for a specific writing style, which is the focus of our study. Additionally, we did not train the bias terms, setting `bias` to `"none"`. Finally, since our task is causal language modeling, which involves predicting the next word in a sequence based solely on preceding words, we set the `task_type` to `"CAUSAL_LM"`.

We also need to adjust the tokenizer of the model. More specifically, we need to ensure we have a padding (PAD) token, which is crucial for handling input sequences of varying lengths during training. This token allows the model to process batches efficiently by aligning sequences to the same length, which prevents errors and ensures seamless training. We set the PAD token to `<unk>`, which is an unknown token (UNK) and is reserved for words that are not in the vocabulary.

Fine-Tuning Dataset

Next, we load the fine-tuning dataset from Hugging Face, which we created through a series of steps. We begin with ten annual reports, distinct from those discussed in Section 4.1. After selecting the reports, we merge them into a single PDF file named `train.pdf`. Our data pre-processing workflow begins by converting the document into plain text using the `pdf_to_text.py` script, which generates a file named `rawtrain.txt`. We use the `PyPDF2` function to extract all the text from each page of the reports. However, since these reports include graphs, tables, and scattered text, the extracted text is often poorly formatted. Using such unprocessed text as a fine-tuning dataset could degrade the model's performance, leading to a poorly structured output.

To address this, we first chunk the data into segments of 4000 tokens and use `GPT-4o-mini` to clean the text, to remove unwanted characters and ensure well-formed sentences. Next, we split the data into training and testing sets, with 10% of the chunks randomly selected for testing and the remaining chunks selected for training.

The train dataset has 411 rows, while the test dataset has 46 rows. Finally, we convert the datasets into CSV format and upload them onto Hugging Face platform.

Fine-Tuning

With the dataset prepared, the next step is to set the fine-tuning or training parameters. We use the `Trainer` class from the `transformers` library and the `SFTTrainer` class from the `trl` library. The selection of parameters was constrained by the GPU. The parameters `per_device_train_batch_size`, `per_device_eval_batch_size` and `max_seq_len` were configured as high as possible without exceeding the available GPU memory. With this approach we utilize the parallelization capabilities of the GPU, leading to more efficient fine-tuning. Specifically, we set these parameters to 512, 2, and 2, respectively. The parameter `max_seq_len` defines the number of tokens in each input row fed into the model, while `per_device_train_batch_size` and `per_device_eval_batch_size` control the number of rows from the training and evaluation datasets that are processed in each fine-tuning step or iteration during fine-tuning.

To mitigate the limitations imposed by GPU memory and to ensure smoother and more generalized learning, we set a higher `gradient_accumulation_steps` parameter. This parameter determines how many fine-tuning steps the model processes and accumulates losses before updating the gradients. Averaging losses over multiple steps reduces the model's tendency to overfit to the intricate details of individual batches, thereby improving generalization. Given our goal of producing a more generalized model, we set this parameter to four.

The learning rate was chosen based on the point at which fine-tuning remained stable, where the training loss did not have erratic fluctuations and evaluation loss consistently decreased. Accordingly, we set the `learning_rate` to $2 \cdot 10^{-4}$. Additionally, we set the `warmup_ratio` to 0.1, which ensures that the learning rate gradually increases from zero to its initial value over the first 10 percent of the training process.

We opted to use a cosine learning rate scheduler [30]. After the warmup period, this scheduler starts at the initial learning rate of $2 \cdot 10^{-4}$ and gradually decreases it towards zero, following the shape of a cosine function. An alternative approach would be to use a constant learning rate, which would result in quicker fine-tuning since the learning rate remains steady. However, by using the cosine schedule, we allow for potentially better model performance, as the lower learning rate towards the end of fine-tuning helps reduce the evaluation loss.

For optimization, we employed the AdamW optimizer [31], specifically choosing `"paged_adamw_8bit"`, which is a common choice when fine-tuning quantized models due to its memory efficiency and compatibility with lower-precision computations.

Finally, we set the number of epochs, controlled by the `num_train_epochs` parameter, to three, meaning the model completes three full passes through the entire training dataset. Although this may seem relatively high, we saved model parameters at 10 percent intervals throughout the fine-tuning process. Therefore, even if the model overfits the data at the end of the third epoch, we can still choose a model somewhere in between, where the evaluation loss is the lowest.

Figure 5.2 provides a detailed illustration of the complete fine-tuning framework, which highlights each step of the fine-tuning process.

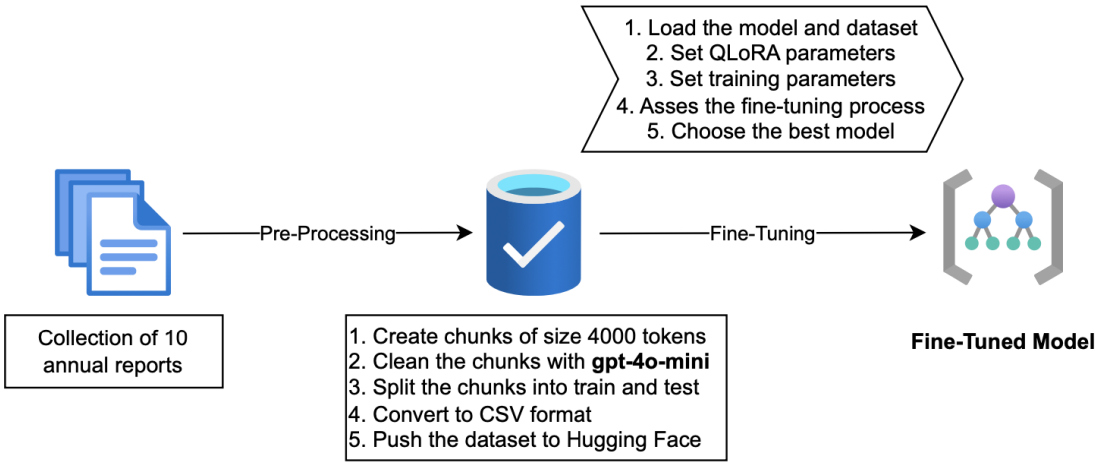


Figure 5.2: Schematic representation of the fine-tuning method

5.5 Models

The models used in this study are all available on the Hugging Face ecosystem, and each is identified by its specific Hugging Face path, which is provided in the model's description. Most of the selected models are `instruct` models, designed to follow instructions provided in the prompt, which makes them well-suited for our task. Additionally, we chose a `code` model, trained to generate code from natural language descriptions. These models excel at extracting specific information, such as names, and are proficient at responding in structured formats like JSON.

Our primary criterion for model selection was size. All the models we selected have approximately seven to eight billion parameters, making them comparable in

terms of performance. Generally, newer models tend to outperform older ones, as companies typically release updated versions that improve upon their predecessors. The decision to use smaller models stems from the fact that fine-tuning large language models is computationally intensive, and our available resources are limited.

When describing the models, it is essential to note their size, as this significantly impacts their performance. The size of a model is typically measured in billions of parameters, and for conciseness, the word "billions" is often replaced by the letter "B", which we will use in the subsequent sections.

5.5.1 Llama 3.1 Instruct

The series of models Llama 3.1 were released by Meta on July 23, 2024. They are an improved version of the Llama 3 [1], which were released on April 18, 2024. The main improvement was on mathematical reasoning and more complex tasks, which comes with a drawback of slightly slower inference. Overall, the models perform well on wide variety of task and sometimes outperform commercial models of bigger size. It comes in three sizes, 8B, 70B and 405B parameters. We chose the smallest 8B model, which outperforms Llama 2 13B on all measured tasks by Meta and for most tasks comes close to the 70B model. We selected the instruction fine-tuned model, which was developed with supervised fine-tuning and reinforcement learning with human feedback (RLHF) [40] to better capture the human preferences. The fine-tuning process incorporated both publicly available instruction datasets and over 25 million synthetically generated examples. In the Hugging Face ecosystem it is identified by the following path: `meta-llama/Meta-Llama-3.1-8B-Instruct`.

bos_token	eos_token	unk_token	pad_token
<|begin_of_text|>	<|eot_id|>	<|eot_id|>	<unk>

Table 5.1: Special token map for Llama 3.1.

Setting up special tokens correctly is crucial for both training and inference, as these tokens play an essential role in guiding the model's behavior. For example when the model gives an output, we want it to stop the inference on its own by predicting the end of sequence (EOS) token and not just continue the inference until it reaches the output tokens limit. Therefore, we provided a special tokens map for every model, which includes the EOS token, beginning of sequence token (BOS), unknown token (UNK) and padding token (PAD). The special token map for Llama 3.1 is displayed in Table 5.1.

5.5.2 CodeLlama Instruct

Code Llama [49] is a special series of models developed by Meta, which perform well on code synthesis and understanding. They are built on the basis of the Llama 2 series and come in four sizes: 7B, 13B, 34B and 70B. The weights were initialized with the Llama 2 model weights and then further trained on 500B tokens of code data, except for the 70B version which was trained on 1 trillion tokens. Besides that, they were also fine-tuned for long context usage. Meta has also released the instruction version, where they used 5B tokens to fine-tune the model to better follow human instruction, and a special Python version, which is specialized for the Python programming language. We again used the smallest instruction model with 7B parameters, which is identified in the Hugging Face ecosystem by the following path: `meta-llama/CodeLlama-7b-Instruct-hf`. Furthermore, the special token map for CodeLlama is displayed in Table 5.2.

bos_token	eos_token	unk_token	pad_token
<s>	</s>	<unk>	<unk>

Table 5.2: Special token map for CodeLlama.

5.5.3 Mistral 7B Instruct

The next open-source model selected for this work was developed by MistralAI, which is a company specializing in the creation of LLMs. Specifically, the Mistral 7B Instruct model, introduced by Jiang et al. [25]. It outperforms the Llama 2 13B on both human and automated benchmarks. Additionally, it demonstrates strong performance in code generation tasks, coming close to the Code Llama 7B model, despite not being fine-tuned on code-specific data. For this project, we employed version three of the Mistral 7B Instruct model, the latest iteration, which includes several improvements such as an expanded vocabulary and support for function calling. The model can be accessed via Hugging Face at `mistralai/Mistral-7B-Instruct-v0.3` and its special token map can be found in Table 5.3.

bos_token	eos_token	unk_token	pad_token
<s>	</s>	<unk>	<unk>

Table 5.3: Special token map for Mistral 7B.

5.5.4 OpenChat 3.5

The OpenChat 3.5 model is built on the Mistral 7B architecture and has been further fine-tuned for enhanced performance. It leverages the OpenChat framework, as proposed by Wang et al. [64]. The framework utilizes a technique called Conditioned-RLHF (C-RLHF). This approach significantly enhances the model's performance on certain tasks, with even the 13B version outperforming ChatGPT in specific areas. C-RLHF extends traditional fine-tuning by making use of non-pairwise, non-ranking-based supervised fine-tuning training data. This data consists of a small amount of expert annotated data combined with a large proportion of easily accessible, sub-optimal data that does not include any preference labels. C-RLFH is then able to leverage this by applying roughly estimated labels to the data. For our purposes, we chose the 7B version, although no instruction-tuned variant is available. The Hugging Face path for this model is `openchat/openchat_3.5` and the special token map is shown in Table 5.4.

bos_token	eos_token	unk_token	pad_token
<s>	<|end_of_turn|>	<unk>	<unk>

Table 5.4: Special token map for OpenChat 3.5.

5.5.5 Models Paramaters

Parameters	**Llama 3**	**CodeLlama**	**Mistral**	**OpenChat3.5**
hidden_size	4096	4096	4096	4096
num_hidden_layers	32	32	32	32
intermediate_size	14336	11008	14336	14336
num_attention_heads	32	32	32	32
max_position_embeddings	131072	16384	32768	8192
vocab_size	128256	32016	32768	32002

Table 5.5: Comparison of model parameters for Llama 3.1 8B, CodeLlama 7B, Mistral 7B, and OpenChat 3.5.

As explained in Section 3.4, several key parameters characterize the transformer architecture, and some of these are presented in Table 5.5. The parameters were obtained from the model's `config` attribute, and therefore, the parameter names

in the table reflect this attribute. The `hidden_size` parameter refers to the dimension of the hidden layers, which is represented as d in Section 3.4. Next, the `num_hidden_layers` parameter indicates the number of layers in the model's encoder and decoder, corresponding to N as described in Section 3.4.4 and Section 3.4.3. The `intermediate_size` refers to the size of the intermediate dimension within the FFN layers of each transformer block, denoted by d_{ff} in Section 3.4.5. Another crucial parameter is `num_attention_heads`, which corresponds to h in Section 3.4.2 and represents the number of attention heads used in multi-head attention. The final two parameters, `max_position_embeddings` and `vocab_size`, differ slightly from those presented in the original papers of the models, as these values can change with each new version and improvement of the models. The `max_position_embeddings` parameter defines the maximum sequence length the model can process, which limits the input length that the model can attend to at once. Finally, the `vocab_size` parameter indicates the number of tokens in the vocabulary, which is denoted by v in Section 3.3.

5.5.6 GPT-4o Mini

We selected the GPT-4o-mini model [38] from OpenAI as a benchmark model for our study. Although the specific details of this commercial model have not been disclosed, it is widely speculated to be comparable in size to the Llama 3 model, making it a reasonable and relevant choice for our comparative analysis. Despite its similar size to open-source models, GPT-4o-mini consistently outperforms them on most benchmarks. Therefore, our objective is to achieve performance as close as possible to that of GPT-4o-mini.

5.6 Evaluation Methods

Evaluating models that generate text, especially from language models, can be challenging because correct answers may be phrased in multiple ways, and there can be various valid responses, such as is the case with the ESG policies. Therefore, most of the time evaluation of an LLM has been done manually, usually by experts in the field. However, in our case it is relatively easy to classify which answer is correct and falls under the ESG policy criteria and which does not. Therefore we decided that each of the extracted policies will be given a 1, if it is relevant and 0, if it is not. Then the relevant metric will be the proportion of correct answers given all the extracted policies. We will also provide the number of extracted policies, which puts the metric into perspective.

Furthermore, extraction of the name does have a unique answer, which makes evaluating the performance of the model on extracting the name of the responsible person for ESG commitments easier. In the following sections, such a person will be referred only as the "ESG person", to make the explanations more concise and easier to understand. We decided that the accuracy metric does not capture the "goodness" of the model completely. Therefore, we will use a confusion matrix and consequently the F1 score. The confusion matrix, which is commonly used in binary classification tasks, illustrates the model's predictions by categorizing them into true positives, true negatives, false positives, and false negatives. Nevertheless, we can still use this metric by defining the criteria for the previously mentioned categories as follows:

- **True Positive (TP):** An answer is classified into TP, if the answer includes the correct name of the ESG person, given that an ESG person exists.
- **False Positive (FP):** An answer is classified into FP, if a model does not clearly indicate or explicitly say that an ESG person does not exist, given that an ESG person does not exist.
- **False Negative (FN):** An answer is classified into FN, if the answer includes an incorrect name, just a title of a person or it indicates that such a person does not exist, given that an ESG person does exist.
- **True Negative (TN):** An answer is classified into TN, if the answer explicitly says or indicates that an ESG person does not exist, given that such a person indeed does not exist.

It is important to note that we do several extractions for each annual report, as explained in Section 5.3 and each of the extractions is classified in one of the classes above. This means that we classify one extraction based on the two pages that we gave to the model as an input. Therefore, it is very common to get an extraction classified as TN, because the ESG person is usually only mentioned once on only one page in the whole annual report. For this reason, we need to establish additional rules to classify all combined extractions from a single annual report into a single category:

1. If we obtain one or more TPs and the rest are TNs, then the answer for the whole annual report is classified as TP.
2. If we obtain one or more TPs and one or more FNs/FPs, then the answer is classified as FN.

3. If all the obtained answers are TNs, then the answer is TN.

4. If we obtain one or more TNs and one or more FPs, then the answer is classified as FP.

The reason for defining the first and the third rule is obvious, because it is the only correct way for positive and negative class. The reason for defining the second rule in that way, is because we take into account the whole annual report. Therefore, if the annual report has an ESG person, then if one of the answers of extractions provides a wrong name, just a title or does not provide a name, then the whole answer is incorrect, which means it is FN. The fourth rule is defined this way, because if the model hallucinates an answer in at least one of the extractions, then the whole answer is classified as FP.

With answers classified into those groups, we can define several metrics, the first two being *precision* and *recall*:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

from which we can define the *F1*-score:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The *F1* metric is a harmonic mean of the *precision* and *recall*, therefore it is a perfect metric of our task, since we consider both *precision* and *recall* to be equally important.

Chapter 6

Results

In this chapter, we present the results of our study. We begin by evaluating the effectiveness of the fine-tuning process for each model. Following this, we examine the results of the policy extraction task, discussing the structure and format of the models' outputs in the process, and provide a comparative analysis of the models' performances. Finally, we assess the results of the name extraction task, once again addressing the structure and format of the models' outputs alongside the performance evaluation.

6.1 Fine-Tuning Results

Before evaluating the performance of the models on the extraction task, it is crucial to present the fine-tuning results and explain which model versions were ultimately selected. As outlined in Section 5.4, each model was fine-tuned for three epochs, which corresponds to 150 steps based on our chosen fine-tuning parameters.

Figure 6.1 illustrates the training and evaluation losses for all the models across these epochs. Immediately, we observe similar patterns between the Llama 3.1 model and CodeLlama, as well as between the Mistral 7B and OpenChat 3.5 models. This can be explained by the fact that Llama 3.1 and CodeLlama are both based on the LLaMA architecture. Although CodeLlama is built upon Llama 2, Meta reused much of the same data that was used to pre-train Llama 3, which explains the similar loss trajectories.

Furthermore, the resemblance between Mistral 7B and OpenChat 3.5 is even less surprising, as OpenChat 3.5 is essentially a fine-tuned version of Mistral 7B. The loss patterns are almost identical across these models, which underscores their connection.

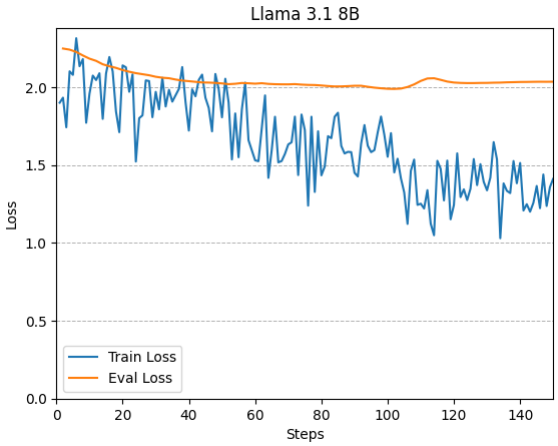
It is also evident from the Figures 6.1 that the Mistral 7B and OpenChat 3.5 models reached their minimum evaluation loss quicker than the Llama 3.1 and CodeL-

lama models. Specifically, Mistral 7B and OpenChat 3.5 reached their minimum after roughly the first epoch, while Llama 3.1 and CodeLlama required until the second epoch. After hitting their lowest evaluation loss, all models exhibited a jump in evaluation loss and a sharp drop in training loss, which is indicative of overfitting.

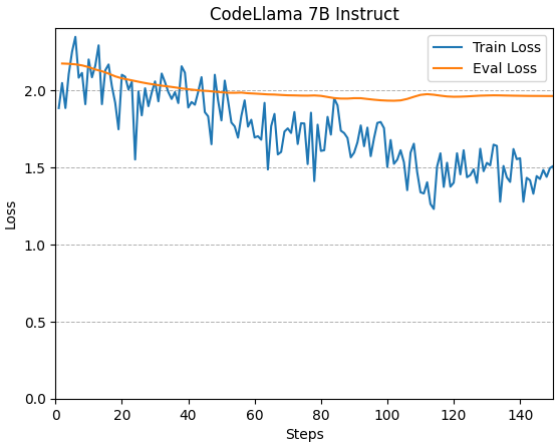
As shown in Figure 6.1a, the evaluation loss for Llama 3.1 8B started at approximately 2.24 and reached its lowest point of 1.98 around step 104, which corresponds to just over two epochs. Similarly, Figure 6.1b indicates that CodeLlama 7B began with a loss of 2.17, and the lowest evaluation loss achieved was 1.93 at step 102. In the case of Mistral 7B, as seen in Figure 6.1c, the model started with a loss of 1.98 and reached its minimum evaluation loss of 1.74 by step 54. Finally, Figure 6.1d shows that OpenChat 3.5 started with a loss of 2.03 and reached its lowest evaluation loss of 1.76 by step 52.

It is worth noting that all the models achieved around a 0.25 reduction in evaluation loss, which might seem modest, but is still significant given the nature of our training data. Since the annual reports use a type of technical text that is also included in the pre-training dataset for the models with the exception of domain specific knowledge such as ESG-related terms. Hence, it is expected that the models would have relatively low evaluation losses from the start. Nevertheless, a 0.25 reduction indicates that the models were able to improve their understanding of the content and writing style used in the annual reports.

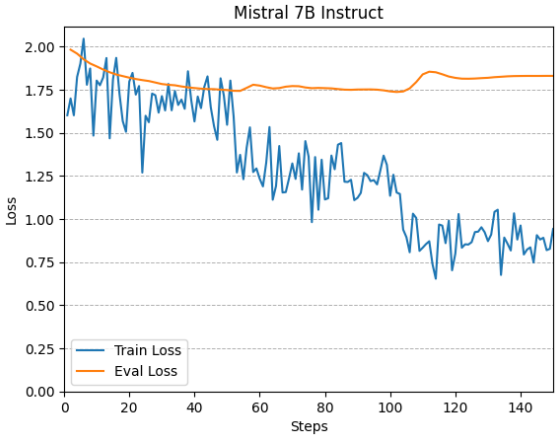
We can also infer that the Mistral-based models likely included more technical text during pre-training compared to the Llama-based models. This is evident from the fact that the evaluation loss for the Mistral models started at around 2, whereas the Llama models began with a higher loss of around 2.2.



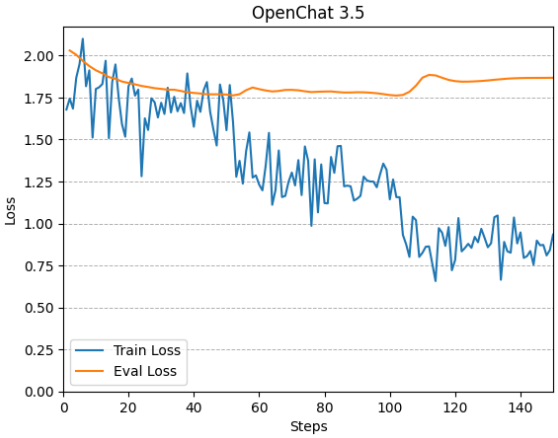
(a) Training and evaluation loss over 3 epochs for the Llama 3.1 8B Instruct model.



(b) Training and evaluation loss over 3 epochs for the CodeLlama 7B Instruct model.



(c) Training and evaluation loss over 3 epochs for the Mistral 7B Instruct model.



(d) Training and evaluation loss over 3 epochs for the OpenChat 3.5 model.

Figure 6.1: Training and evaluation losses over 3 epochs during fine-tuning for all models.

6.2 Policy Extraction

After examining the fine-tuning results we first look into the performance of the models on the policy extraction task. The process of extraction is described in Section 5.3, where we also explain that the desired output format is the JSON object, presented in Listing 5.3. We also provided the prompt we used for this task in the Appendix, Listing A.1. The evaluation of performance for this model is described in Section 5.6. We focus on two main metrics, which are: average percentage of relevant policies and average number of extracted policies. We also include important information about

those metrics, such as, minimum, maximum and standard deviation. This gives us a more in depth understanding of the performance of the models.

Given that we are interested in examining the effectiveness of the fine-tuning process, we will first compare the metrics of the base version against the fine-tuned version for each model separately. After that we will also compare the models to each other and examine the overall quality of the output.

Llama 3.1 8B Instruct

Metric	Llama 3.1 Base	Llama 3.1 Fine-Tuned
Avg. % Relevant Policies	85.84%	89.22%
Max. % Relevant Policies	100.00%	100.00%
Min. % Relevant Policies	35.71%	31.00%
Std. Dev. % Relevant Policies	14.44%	15.55%
Avg. Total Policies	16.63	19.60
Max. Total Policies	38	38
Min. Total Policies	5	3
Std. Dev. Total Policies	7.94	9.05

Table 6.1: Model performance metrics for ESG policy extraction for Llama 3.1 Instruct base and fine-tuned version.

The first model we will examine is the Llama 3.1, and immediately we can see in Table 6.1 that the fine-tuned version performs better on both key metrics. It achieves a 3.38 percentage point increase in the average percentage of relevant extracted policies. Furthermore, it also extracts, on average, 2.97 more policies. This is a relatively significant improvement in performance, given that the base version already performs well. However, the fine-tuned version does have some drawbacks, as the minimum for both metrics is lower and the standard deviation is higher than for the base version. Ideally, the model would score high on both key metrics while maintaining a low standard deviation, indicating consistency. Nevertheless, the differences in minimums and standard deviations are not substantial. Therefore, we still consider the fine-tuned version to be superior.

In terms of outputting a correct JSON format, both versions performed excellently, consistently producing a valid JSON format. However, it was also common for both versions to output unnecessary text alongside the JSON object, but this is not an issue as we have a post-processing step to handle it. It is also worth noting that both versions sometimes struggled with `page_number` extraction. It often occurred that

only one page number was provided for each JSON object, which was incorrect, as the extracted policies were typically present on both provided pages.

CodeLlama 7B Instruct

Metric	CodeLlama Base	CodeLlama Fine-Tuned
Avg. % Relevant Policies	67.30%	55.76%
Max. % Relevant Policies	100.00%	100.00%
Min. % Relevant Policies	28.57%	0.00%
Std. Dev. % Relevant Policies	19.51%	32.34%
Avg. Total Policies	13.33	10.97
Max. Total Policies	31	29
Min. Total Policies	5	1
Std. Dev. Total Policies	5.13	5.82

Table 6.2: Model performance metrics for ESG policy extraction for CodeLlama 7B Instruct base and fine-tuned version.

Next, we examine another LLaMA architecture-based model, CodeLlama 7B Instruct. We chose to test the performance of this model because we required the output to be in JSON format, and given that CodeLlama is designed to output code, we believed it would excel in this regard. We were proven right, as the output was a correct JSON format every time. However, the model generally struggled with generating text. It frequently happened that specific policies were written in bullet points and contained numerous spelling mistakes. This problem was even exacerbated by fine-tuning the model. The poor performance is reflected in the values of the metrics, presented in Table 6.2. Overall, the base CodeLlama model performed poorly compared to Llama 3.1. However, it outperformed the fine-tuned version on all metrics by a considerable margin. In the average percentage of relevant policies, it outscored the fine-tuned version by an impressive 11.54 percentage points.

We believe that the reason for such poor performance of the fine-tuned model is due to the fact that we used unstructured text for fine-tuning, which is the opposite of what the model was fine-tuned on during its development.

Mistral 7B Instruct

Metric	Mistral Base	Mistral Fine-Tuned
Avg. % Relevant Policies	82.38%	86.15%
Max. % Relevant Policies	100.00%	100.00%
Min. % Relevant Policies	33.33%	50.00%
Std. Dev. % Relevant Policies	16.68%	12.84%
Avg. Total Policies	16.53	16.97
Max. Total Policies	42	46
Min. Total Policies	5	5
Std. Dev. Total Policies	8.29	8.56

Table 6.3: Model performance metrics for ESG policy extraction for Mistral 7B Instruct base and fine-tuned version.

Moving on from the LLaMA architectures, we assessed the performance of the Mistral 7B Instruct. As visible from Table 6.3, it performed this task excellently. In addition to the high values for key metrics in both versions, it is remarkable that the maximum number of extracted policies exceeded 40 for both. Furthermore, the fine-tuned version outperformed the base version on almost all metrics. Beyond the significant increase in the average percentage of relevant policies, it maintained a minimum of 50%, while the base version had a minimum of 33%. This is an important criterion, as it indicates that the fine-tuned model better understands the distinction between relevant and non-relevant policies. This stems from the fact that, at times, the provided pages contain very little information on ESG policies. When the model better understands the text, it outputs fewer policies, resulting in a higher minimum score.

Another notable achievement for both versions is that they consistently produced correct JSON formats without adding unnecessary text. Additionally, they both performed impressively in extracting the correct page numbers. If the information was found on both provided pages, the models accurately identified both page numbers. Similarly, when the information was found on only one page, the models were able to identify that correctly as well.

OpenChat 3.5

Metric	OpenChat 3.5	OpenChat 3.5 Fine-Tuned
Avg. % Relevant Policies	76.71%	81.54%
Max. % Relevant Policies	100.00%	100.00%
Min. % Relevant Policies	36.36%	45.16%
Std. Dev. % Relevant Policies	16.51%	15.68%
Avg. Total Policies	20.80	24.83
Max. Total Policies	55	65
Min. Total Policies	5	5
Std. Dev. Total Policies	10.45	13.26

Table 6.4: Model performance metrics for ESG policy extraction for OpenChat 3.5 base and fine-tuned version.

The last of the open-source models is OpenChat 3.5, which overall achieved good results. Additionally, the fine-tuned version surpassed the base version by a considerable margin on most metrics, as seen in Table 6.4. Most notably, it was able to achieve a score of 81.54% on the average percentage of relevant policies, representing a 4.83 percentage point increase over the base model.

Moreover, both versions achieved impressive results in terms of the total number of extracted policies, with 55 for the base version and 65 for the fine-tuned version. This translated into a high average number of extracted policies, with the base version averaging 20.80 and the fine-tuned version 24.83.

In addition to the sheer number of extracted policies, OpenChat 3.5, particularly the fine-tuned version, had a tendency to output very long policies. It frequently happened that a single policy was written over multiple sentences, sometimes taking up the length of an entire paragraph. Nevertheless, these lengthy policies were usually highly accurate. Both models also usually returned answers in proper JSON format, though they often included unnecessary text. They were also able to extract policies from the correct pages in most cases. However, sometimes they would extract both relevant pages, and occasionally, they extracted none.

GPT-4o-mini

Metric	Value
Avg. % Relevant Policies	94.32%
Max. % Relevant Policies	100.00%
Min. % Relevant Policies	54.55%
Std. Dev. % Relevant Policies	10.70%
Avg. Total Policies	13.40
Max. Total Policies	25
Min. Total Policies	6
Std. Dev. Total Policies	3.69

Table 6.5: Model performance metrics for ESG policy extraction for GPT-4o-mini.

In order to put the performance of the open-source models into perspective, we selected the GPT-4o-mini model as a benchmark. Given that GPT-4o-mini is a commercial model with likely more parameters, trained on a larger dataset, and not in a quantized form, we expected it to outperform all other models. Therefore, the objective was to develop a model that could come as close as possible to its results, which are shown in Table 6.5.

GPT-4o-mini achieved a remarkable average percentage of relevant policies at 94.32%, along with a very high minimum percentage, which resulted in a low standard deviation. However, the average number of total policies extracted was not outstanding, standing at only 13.40. This is due to the fact that the model adhered strictly to the JSON template, shown in Listing 5.3, which includes five policies, and this is generally how many the model output. Furthermore, it is evident that the model's strategy was to output fewer policies, but ensure they were of higher quality.

Surprisingly, however, the model had trouble generating a correct JSON format and often simplified it by outputting only two lists and a page number. Additionally, it never provided both page numbers, even when it found the information across multiple locations.

Comparison

Following our investigation into the performance of individual models and their fine-tuned counterparts, we now compare all the models against one another. This comparison is illustrated through two figures, each showcasing one of the key metrics. We also included confidence intervals (CIs), which offer a more comprehensive understanding of the models' performance on each specific metric.

Given the computed metrics for each model, we have all the necessary values to construct the confidence intervals. In this case, the sample size is 30, denoted by n. We also have $\bar{x}$, which represents the sample mean—namely, the average percentage of relevant policies and the average number of total policies. Finally, we use the parameter σ, which is the standard deviation corresponding to each metric. Due to the relatively small sample size, we determined that a 90% confidence interval would be most appropriate. This choice corresponds to a Z-value of 1.645, derived from the standard normal distribution. The formula for computing the confidence interval is:

$$\text{CI} = \bar{x} \pm Z \cdot \frac{\sigma}{\sqrt{n}}$$

Here, $\frac{\sigma}{\sqrt{n}}$ represents the standard error of the mean (SEM), providing an estimate of the uncertainty associated with the sample mean.

For example, to compute the confidence interval for the average percentage of relevant policies for the Llama 3.1 base model, we use a mean of 85.84%, a standard deviation of 14.44%, and a sample size of 30. The standard error is calculated as $\frac{14.44}{\sqrt{30}} \approx 2.635$, and the confidence interval is $85.84 \pm 1.645 \cdot 2.635$, resulting in a range of approximately 81.51% to 90.17%. Similarly, confidence intervals for other metrics are calculated using the same method, ensuring we account for variability in the data.

The comparison of the models on the average percentage of relevant policies is presented in Figure 6.2. From the graph, it is evident that GPT-4o-mini stands out as the top-performing model, achieving the highest scores. Next, we have the fine-tuned version of Llama 3.1, followed by the fine-tuned version of Mistral 7B in third place. This demonstrates the effectiveness of fine-tuning on the performance of policy extraction, as fine-tuning improved the Mistral 7B model to such a degree that it outperformed the base version of the superior Llama 3.1 model. However, the biggest improvement with fine-tuning was seen in the OpenChat 3.5 model, with a 4.83 percentage point increase, compared to Llama 3.1 and Mistral 7B, which saw gains of 3.38 and 3.77 percentage points, respectively.

On the other hand, while CodeLlama performs the worst from the start, fine-tuning further degrades its performance. This results in a performance difference of 33.46 percentage points between the worst model and the best open-source model, and 38.56 points against the benchmark.

Overall, the graph highlights Llama 3.1, Mistral 7B, and OpenChat 3.5 as the most promising models for extracting relevant policies, with consistent improvements observed after fine-tuning.

Average percentage of relevant policies, however, does not present the whole picture, if we want find the model. As explained before, the model could choose to output very few policies and therefore keep the percentage high. This is why in Figure 6.3 we present the average number of total extracted policies, along with the corresponding confidence intervals. Surprisingly, the benchmark model, was only able to outperform the CodeLlama model, which was again the only model where fine-tuning worsened the performance. However, the GPT-4o-mini does have the smallest confidence interval, which indicates the lowest standard deviation for this metric.

By far, the best-performing model is OpenChat 3.5, as even the base version outperforms all the other models. The fine-tuned version managed to output an average of 24.83 policies, while the third-best model, Llama 3.1 fine-tuned, managed to extract 19.60.

In addition, OpenChat 3.5 is the model that showed the greatest improvement with fine-tuning, increasing its average by 4.03 policies. In comparison, Llama 3.1 improved by 2.97, and Mistral 7B by only 0.44. However, it is important to note that OpenChat 3.5 has the largest confidence intervals, and therefore the highest standard deviation. Furthermore, fine-tuning contributed to a higher standard deviation over the base model, which can be seen as a negative effect of outputting more policies.

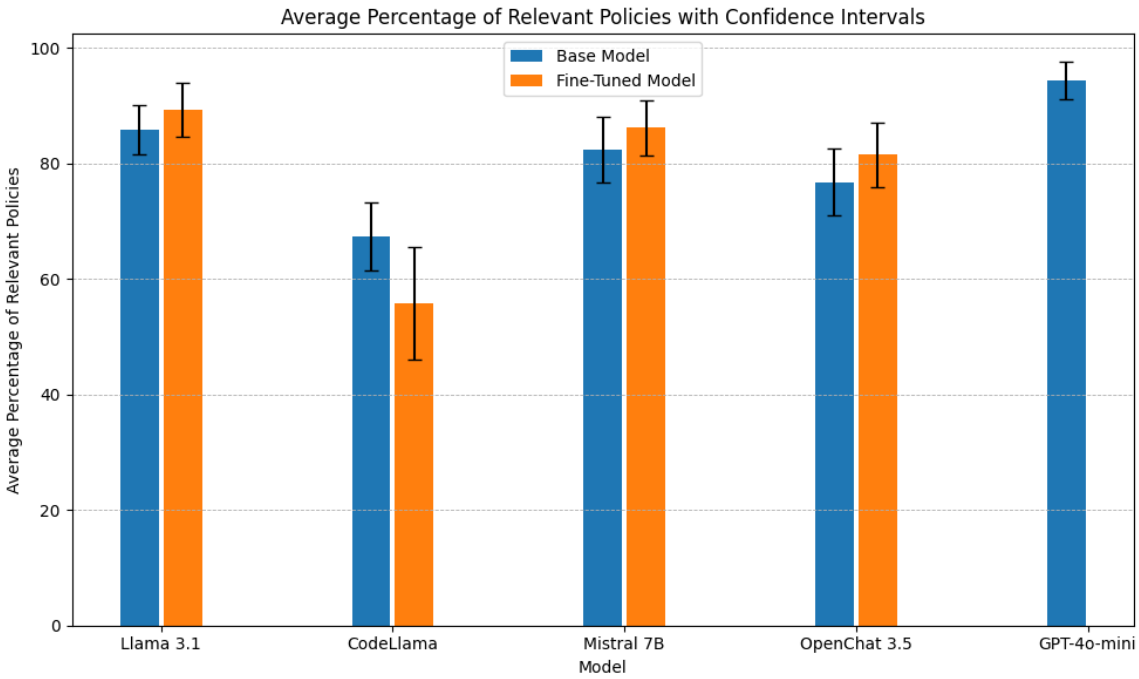


Figure 6.2: Comparison of all models, base and fine-tuned versions, on average percentage of relevant policies with 90% confidence intervals..

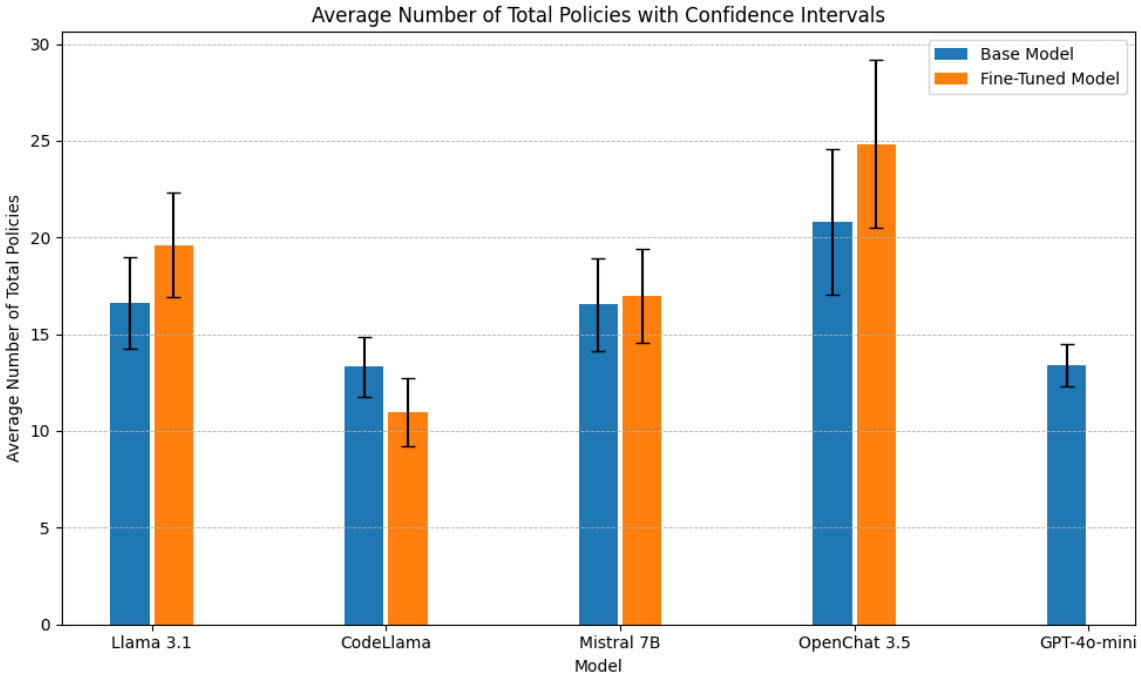


Figure 6.3: Comparison of all models, base and fine-tuned versions, on average number of total policies with 90% confidence intervals.

6.3 Name Extraction

The second task we focused on was name extraction, as outlined in Section 5.3. To accomplish this, we utilized a prompt, provided in Listing A.1 in the appendix. Furthermore, the evaluation process for this task is detailed in Section 5.6. Each response was categorized into one of four classes: true positive, false negative, true negative, or false positive. Based on the number of instances in each class for each model, we computed the F1 score, which serves as the primary metric for evaluating model performance.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 5	FP = 1
Positive	FN = 5	TP = 19

Precision: 0.9500 **Recall:** 0.7917
F1 Score: 0.8637

Table 6.6: Confusion Matrix, Precision, Recall and F1 score for Llama 3.1 8B Instruct.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 5	FP = 1
Positive	FN = 5	TP = 19

Precision: 0.9500 **Recall:** 0.7917
F1 Score: 0.8637

Table 6.7: Confusion Matrix, Precision, Recall and F1 score for Llama 3.1 8B Instruct Fine-Tuned.

For Llama 3.1, the performances of the base model and the fine-tuned version are identical, with both achieving a high F1 score, shown in Table 6.6 and Table 6.7. The models made six mistakes, five of which were false negatives. These errors mostly occurred when extracting incorrect names, particularly in cases where multiple people were mentioned as part of the ESG committee. Additionally, the models generally succeeded in recognizing when there was no main ESG person, correctly classifying true negatives five out of six times. This is significant because it demonstrates that the models do not simply extract a random name from the text.

Similar to the policy extraction task, both models consistently produced a correct JSON format. However, the outputs often contained additional, unnecessary text outside the JSON object. Notably, the fine-tuned version tended to include more irrelevant text compared to the base model. Furthermore, instead of leaving the JSON template empty or indicating the lack of an ESG person in the text, both models just printed out the template.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 4	FP = 2
Positive	FN = 13	TP = 11

Precision: 0.8462 **Recall:** 0.4583
F1 Score: 0.5946

Table 6.8: Confusion Matrix, Precision, Recall, and F1 Score for CodeLlama Base.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 3	FP = 3
Positive	FN = 11	TP = 13

Precision: 0.8125 **Recall:** 0.5417
F1 Score: 0.65

Table 6.9: Confusion Matrix, Precision, Recall, and F1 Score for CodeLlama Fine-Tuned.

CodeLlama, on the other hand, performed poorly in terms of F1 scores, as shown in Table 6.8 and Table 6.9. However, unlike in the policy extraction task, fine-tuning led to a noticeable improvement in performance, with the fine-tuned model achieving an F1 score of 0.65, compared to 0.5946 for the base version. Notably, the fine-tuned version had three false positive cases, indicating the model's tendency to output a name regardless of its relevance to ESG. This also contributed to the high number of false negatives, as the model would frequently output a correct name but would also extract an additional, irrelevant name, leading to the entire answer being classified as a false negative.

As expected, both versions of CodeLlama consistently output a correct JSON object, although they included a significant amount of unnecessary text outside of the JSON structure. Additionally, the model occasionally inserted the name of the

company instead of leaving the field empty or simply returning the template. Despite these issues, it was generally able to identify the correct page number.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 6	FP = 0
Positive	FN = 4	TP = 20

Precision: 1.000 **Recall:** 0.8333
F1 Score: 0.9091

Table 6.10: Confusion Matrix, Precision, Recall, and F1 Score for Mistral 7B Base.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 5	FP = 1
Positive	FN = 3	TP = 21

Precision: 0.9545 **Recall:** 0.8750
F1 Score: 0.9130

Table 6.11: Confusion Matrix, Precision, Recall, and F1 Score for Mistral 7B Fine-Tuned.

Next, we examine the performance of the Mistral 7B model, where, once again, the fine-tuned version outperformed the base model. This improvement was not due to the number of mistakes but rather because the errors were more evenly distributed between false positives and false negatives. As seen in Table 6.11, the fine-tuned model registered one false positive and three false negatives, leading to an impressive F1 score of 0.9130. In contrast, the base version had four false negatives, resulting in a lower F1 score.

The output quality for both models remained consistently high. The models produced correctly formatted JSON objects without unnecessary text outside the JSON structure. Furthermore, both models reliably identified the correct page from the two provided, demonstrating accuracy in locating the relevant information.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 5	FP = 1
Positive	FN = 5	TP = 19

Precision: 0.9500 **Recall:** 0.7917
F1 Score: 0.8637

Table 6.12: Confusion Matrix, Precision, Recall, and F1 Score for OpenChat 3.5 Base.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 5	FP = 1
Positive	FN = 4	TP = 20

Precision: 0.9524 **Recall:** 0.8333
F1 Score: 0.8889

Table 6.13: Confusion Matrix, Precision, Recall, and F1 Score for OpenChat 3.5 Fine-Tuned.

Lastly, the OpenChat 3.5 model demonstrated strong performance, achieving a high F1 score. The fine-tuned version was able to correctly identify one additional true positive, resulting in an F1 score of 0.8889, as shown in Table 6.13.

Moreover, the model consistently produced a correct JSON format in most cases, while accurately detecting the relevant page number. However, instead of indicating when an ESG responsible person was not present, the model would occasionally return the company name in place of an individual.

	Predicted	
Actual	**Negative**	**Positive**
Negative	TN = 6	FP = 0
Positive	FN = 2	TP = 22

Precision: 1.000 **Recall:** 0.9167
F1 Score: 0.9565

Table 6.14: Confusion Matrix, Precision, Recall, and F1 Score for GPT-4o-mini.

The performance of GPT-4o-mini was nearly flawless, making only two errors, both of which occurred because it extracted an additional name that was relevant to ESG but not the primary one. This demonstrates the model's strong understanding of the text, resulting in an impressive F1 score of 0.9565, as shown in Table 6.14.

However, the model struggled with consistently outputting the information in the required JSON format. It frequently produced the correct name without wrapping it in a JSON object and also failed to include the page number. This is a significant drawback, as the structured format is essential for our extraction task, and GPT-4o-mini fell short in meeting this requirement.

Comparison

Comparing all the models, we can immediately see in Figure 6.4 that there is only a small difference in performance among the top three models: Llama 3.1, Mistral 7B, and OpenChat 3.5. The slight improvement in the fine-tuned versions suggests that these models might have already reached their upper performance limit, and the enhancements in text understanding from fine-tuning were not substantial enough to dramatically increase their ability to perform this task.

Additionally, we observe that while both OpenChat 3.5 and Llama 3.1 achieved the same F1 score with their base versions, OpenChat 3.5 showed improvement after fine-tuning, whereas Llama 3.1 did not.

Furthermore, the benchmark model, GPT-4o-mini, outperformed the rest of the models by a considerable margin, achieving an impressive F1 score of 0.9565. In comparison, the best open-source model, the fine-tuned version of Mistral 7B, attained an F1 score of 0.9130.

Interestingly, the models made mistakes on different cases, suggesting that each model has distinct preferences for language type and text structure, further indicating variation in how they process and interpret the data.

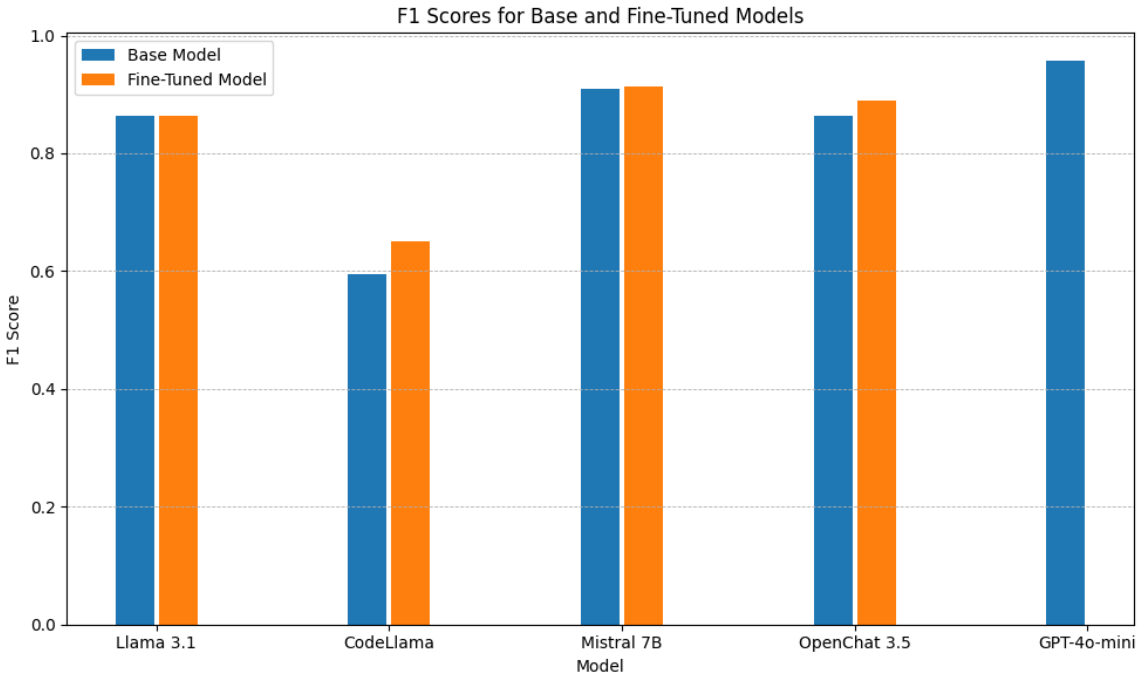


Figure 6.4: Comparison of all models, base and fine-tuned versions, on the F1 score.

Chapter 7

Discussion

Considering all the performance metrics presented, choosing the best overall model is not a straightforward decision, because we have a different top performing model for each task. In the case we did not have a severely limited computational power, we could use different models for different tasks. This would naturally lead to the best possible result in extracting the ESG data from annual reports. In this case we would choose the fine-tuned LLama 3.1 8B Instruct model for the policy extraction and the fine-tuned Mistral 7B Instruct for the name extraction.

However, if we must choose only one model and consider both tasks equally important, we need to evaluate the differences between the two models and assess the percentage difference for each specific metric. We consider an average percentage of relevant polices as the most important metric regarding the policy extraction task. To this end, Llama 3.1 fine-tuned version achieved a score of 89.22% which means that it is 3.5% better than the Mistral 7B fine- tuned, with the score of 86.15%. On the other hand, the Mistral model is 5.7% better in terms of the F1 score compared to the Llama 3.1, for the name extraction task.

Taking both tasks into account, the fine-tuned Mistral 7B Instruct emerges as the more favorable model overall. In addition to its performance, it consistently produced the best-structured output.

Limitations

Overall, we are satisfied with the performance of the models on this task, though there were some constraints related to time and computational resources that limited our ability to further refine and evaluate the models.

One key limitation was that our evaluation approach for policy extraction did not fully capture the quality of each response. While we assessed whether the extracted

policies were relevant, we did not measure the completeness or depth of the answers. A policy could be accurate and pertinent but still differ in the level of detail it provided. We observed that fine-tuned models generally produced more thorough and complete responses. However, assessing the quality of these answers would have required a more sophisticated scoring system and considerably more time, which was beyond the scope of this study.

Another constraint was the relatively small size of the testing dataset, which consisted of a limited number of annual reports. A larger and more diverse dataset might have yielded different insights, offering a better understanding of the models' performance across various contexts.

Additionally, the uniformity of prompts used across all models posed a challenge. Applying the same prompt to each model is not ideal, as different models may respond better to differently structured instructions. Tailoring prompts to align with each model could have potentially enhanced performance, but this approach was not feasible within the scope of this study.

The process of fine-tuning the models also faced challenges related to the dataset's size and quality. The dataset was derived from a limited selection of annual reports, which restricted the variety of text patterns and contexts available for the models to learn from. This limitation may have contributed to inconsistencies in the models' performance across different tasks, indicating difficulties with generalization.

Moreover, issues with data cleanliness were a significant concern. Extracting text from annual reports often resulted in poorly formatted content, particularly due to the presence of tables, graphs, and other elements that introduced noise. While our preprocessing pipeline included steps to improve data quality, such as inserting clear page markers and correcting misplaced characters, some inconsistencies persisted. These issues likely impacted the models' ability to learn effectively and generate structured outputs.

Future Work

Several opportunities exist for improving our framework, which presents exciting directions for future research.

One area of improvement is to experiment with larger models and re-examine the impact of fine-tuning on performance. This would require significantly more computational resources, particularly for models like Llama 3.1, where the next size up contains 70B parameters, which is nearly nine times larger than the models we used. Additionally, it would be beneficial to explore newer models of similar size,

as approximately every six months, a new model is released that outperforms its predecessors.

Improving our fine-tuning dataset is another area of potential enhancement. This could involve further cleaning the dataset or focusing exclusively on the ESG-related pages of the annual reports. It would also be interesting to study how different fine-tuning datasets affect model's performance.

A more advanced approach would be to create a labeled dataset with input and output features to perform supervised fine-tuning. This would allow us to create a model specialized in a specific task or question, likely resulting in better performance for that task. Supervised fine-tuning is generally more effective for task-specific problems, and applying it here would likely lead to significant performance improvements.

Moreover, integrating Direct Preference Optimization (DPO) [45] offers a promising path for improving adaptability. DPO focuses on directly optimizing for preferences, such as user feedback or specific outcomes. This could enhance the model's ability to prioritize and learn from feedback, improving its understanding of subtle differences in ESG policies across various reports. For instance, DPO could adjust the model's focus to emphasize nuanced components of sustainability or governance practices.

Additionally, DPO could tailor model outputs to meet specific industry requirements. By integrating feedback from domain experts or end-users, DPO could guide the models to prioritize extracting elements most valuable to stakeholders, increasing the relevance and accuracy of the extracted data.

Finally, we observed that the models' output is relatively sensitive to the instructions provided in the prompt. While considerable thought went into designing the prompts, exploring different prompt structures and further refining them could yield better results. Given more time, this would be a promising area for further experimentation.

Appendix A

Prompts

Policy Extraction Prompt

```
user_prompt = f"""
    What are the key elements of the companys ESG (Environmental,
    Social, and Governance) or sustainability policyincluding
    specific commitments or initiatives? Please answer in the
    following JSON format.

    <response format>
    {{
      "response": {{
        "key_elements": [
          "Element_1",
          "Element_2",
          "Element_3",
          "Element_4",
          "Element_5",
        ],
        "specific_policies": [
          "Policy_1",
          "Policy_2",
          "Policy_3",
          "Policy_4",
          "Policy_5"
        ]
      }},
      "page_number": "Page number"
    }}
    </response format>

    IMPORTANT RULES:
    1. The "specific_policies" field MUST include DETAILED
      descriptions of any explicit promises, initiatives, or
      actions the company has committed to under its ESG policy.
```

```
    2. The "key_elements" field MUST include SHORT
      descriptions of the major components of the companys
      ESG policy.
    3. If relevant information is not found, leave fields empty.

    <document>

    {text}

    </document>

    Remember:
    1. Ensure each page number is directly relevant to the
       information provided.
    2. If you cannot find relevant information, clearly state
       this in your response.
    3. Provide short descriptions for "key_elements" and detailed
       ones for "specific_policies".

    Now, please answer the given query using the provided
    information and following these guidelines. Answer in ONLY
    JSON format!
    """
```

Listing A.1: The user prompt used for relevant policies extraction.

Name Extraction Prompt

```
user_prompt = f"""
    What is the NAME of the senior executive, board member,
    or employee responsible for overseeing the company's ESG
    (Environmental, Social, and Governance) or sustainability
    initiatives in the provided document?

    <response format>
    {{
      "response": "The name of the employee",
      "page_number": "Page number"
    }}
    </response format>

    IMPORTANT RULES:
    1. Extract only the name that is explicitly mentioned in the
       provided document text.
    2. Do not infer or add any name that is not directly stated.
    3. There is only ONE possible name.
    4. Adhere strictly to the specified JSON format.
```

```
    5. Ensure that the response format does not include any extra
       spaces or text outside the schema.
    6. If the name of the responsible person is not found in the
       text, leave the space empty.

    Here are the relevant documents for your query:

    <document>

        {text}

    </document>

    Remember:
    1. Do not include any information that is not explicitly
       mentioned in the text.
    2. Analyze the text carefully to ensure the correct name is
       extracted.
    3. There is only ONE possible name.
    4. Adhere strictly to the response format without adding extra
       spaces or text.
    5. Answer with only the filled-in template.

    Now, please answer the given query using the provided
    information and following these guidelines. Answer ONLY in
    the specified JSON format!
    """
```

Listing A.2: The user prompt used for ESG person name extraction.

Page Extraction Prompt

```
user_prompt = f"""
  You are a research assistant. Your task is to analyze the table
  of contents of an annual report and identify the relevant
  sections that contain information related to the company's ESG
  (Environmental, Social, and Governance) or sustainability
  policy, as well as details regarding the senior executive, board
  member, or employee responsible for these areas.

  IMPORTANT INSTRUCTIONS:
  1. For each relevant section, provide the starting page number.
  2. Additionally, provide the starting page number of the
     following section, so that the range of pages covered by the
     ESG or sustainability section can be clearly identified.
  3. Ensure that your response is accurate and includes only
     the relevant sections.
```

```
Please provide the following information in JSON format:
A list of the starting pages of all sections likely to contain
information about ESG or sustainability policies or about the
employees responsible for these policies, along with the
starting page of the next section.

Format the response as shown below:

{{
  "esg_related_sections": [
    {{
      "start_page": starting_page_of_section,
      "next_section_start_page": starting_page_of_next_section
    }},
    ...
  ]
}}

For example:

{{
  "esg_related_sections": [
    {{
      "start_page": 45,
      "next_section_start_page": 59
    }},
    {{
      "start_page": 121,
      "next_section_start_page": 204
    }}
  ]
}}

<FIRST 5 PAGES OF THE ANNUAL REPORT>
"""

system_prompt += f"{pages}"

system_prompt += """

Remember:
For each relevant section, provide the starting page and the
starting page of the next section. This will help identify the
range of pages covered by the ESG or sustainability section.
"""
```

Listing A.3: The user prompt for relevant ESG pages extraction.

Bibliography

[1] Meta AI. Introducing meta llama 3: The most capable openly available llm to date. https://ai.meta.com/blog/meta-llama-3/, 2024.

[2] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. Palm 2 technical report, 2023.

[3] Wazib Ansar, Saptarsi Goswami, and Amlan Chakrabarti. A survey on transformers in nlp with focus on efficiency, 05 2024.

[4] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report, 2023.

[5] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[7] KPMG Belgium Board Leadership Center. Environmental, social and governance (esg) boardroom questions. `https://kpmg.com/be/en/home/insights/2021/07/blc-environmental-socia-and-governance-esg-boardroom-questions.html`, July 2021. Accessed: 2024-10-18.

[8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier

Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

[9] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020.

[10] Caterina De Lucia, Pasquale Pazienza, and Mark Bartlett. Does good esg lead to better financial performances by firms? machine learning and logistic regression models of public enterprises in europe. *Sustainability*, 12(13), 2020.

[11] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

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

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

[14] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling, 2022.

[15] Alexander Dunn, John Dagdelen, Nicholas Walker, Sanghoon Lee, Andrew S. Rosen, Gerbrand Ceder, Kristin Persson, and Anubhav Jain. Structured information extraction from complex scientific text with fine-tuned large language models, 2022.

[16] Jannik Fischbach, Max Adam, Victor Dzhagatspanyan, Daniel Mendez, Julian Frattini, Oleksandr Kosenkov, and Parisa Elahidoost. Automatic esg assessment of companies by mining and evaluating media coverage data: Nlp approach and tool, 2024.

[17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.

[18] Google AI Blog. An important next step on our ai journey. https://blog.google/technology/ai/bard-google-ai-search-updates/, 2024. Accessed: 2024-08-28.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[20] Marko Hostnik. *S poizvedovanjem obogateno dopolnjevanje programske kode za lokalne projekte z velikimi jezikovnimi modeli: magistrsko delo*. PhD thesis, 2024.

[21] 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.

[22] Yan Hu, Qingyu Chen, Jingcheng Du, Xueqing Peng, Vipina Kuttichi Keloth, Xu Zuo, Yujia Zhou, Zehan Li, Xiaoqian Jiang, Zhiyong Lu, Kirk Roberts, and Hua Xu. Improving large language models for clinical named entity recognition via prompt engineering, 2024.

[23] Cheng-Li Huang and Fan-Hua Kung. Drivers of environmental disclosure and stakeholder expectation: Evidence from taiwan. *Journal of Business Ethics*, 96:435–451, 10 2010.

[24] Mathav Raj J, Kushala VM, Harikrishna Warrier, and Yogesh Gupta. Fine tuning llm for enterprise: Practical guidelines and recommendations, 2024.

[25] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[26] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[27] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[28] Diya Li, Asim Kadav, Aijing Gao, Rui Li, and Richard Bourgon. Automated clinical data extraction with knowledge conditioned llms, 2024.

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[30] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[31] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[32] Man Luo, Xin Xu, Yue Liu, Panupong Pasupat, and Mehran Kazemi. In-context learning with retrieved demonstrations for language models: A survey, 2024.

[33] Martina Macpherson, Andrea Gasperini, and Matteo Bosco. Implications for artificial intelligence and esg data. *SSRN Electronic Journal*, June 2021. Available at SSRN: `https://ssrn.com/abstract=3863599` or `http://dx.doi.org/10.2139/ssrn.3863599`.

[34] MarkTechPost. With 700,000 large language models (llms) on hugging face already, where is the future of artificial intelligence ai headed?, 2024. Accessed: 2024-08-23.

[35] J. F. G. Morros. The integrated reporting: A presentation of the current state of art and aspects of integrated reporting that need further development. *OmniaScience*, 12(1), 02 2016.

[36] Golam Md Muktadir. A brief history of prompt: Leveraging language models. (through advanced prompting), 2023.

[37] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticæ Investigationes*, 30(1):3–26, 2007.

[38] OpenAI. Gpt-4o model. https://openai.com/gpt-4o-contributions/, July 2024. Acknowledgments to leads: Jacob Menick, Kevin Lu, Shengjia Zhao, Eric Wallace, Hongyu Ren, Haitang Hu, Nick Stathas, Felipe Petroski Such. Program Lead: Mianna Chen.

[39] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok

Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

[40] 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.

[41] Maciej P. Polak, Shrey Modi, Anna Latosinska, Jinming Zhang, Ching-Wen Wang, Shaonan Wang, Ayan Deep Hazra, and Dane Morgan. Flexible, model-agnostic method for materials data extraction from text using general purpose language models. *Digital Discovery*, 3(6):1221–1235, 2024.

[42] Maciej P. Polak and Dane Morgan. Extracting accurate materials data from research papers with conversational language models and prompt engineering. *Nature Communications*, 15(1), February 2024.

[43] Ofir Press and Lior Wolf. Using the output embedding to improve language models, 2017.

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

[45] Eric Mitchell Stefano Ermon Christopher D. Manning Chelsea Finn Rafael Rafailov, Archit Sharma. Large language models for generative information extraction: A survey, 2024.

[46] Natraj Raman, Grace Bang, and Armineh Nourbakhsh. Mapping esg trends by distant supervision of neural language models. *Machine Learning and Knowledge Extraction*, 2:453–468, 10 2020.

[47] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389, 01 2009.

[48] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[49] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

[50] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[51] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[52] Willem Schramade. Integrating esg into valuation models and investment decisions: the value-driver adjustment approach. *Journal of Sustainable Finance & Investment*, 6(2):95–111, 2016.

[53] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016.

[54] Noam Shazeer. Glu variants improve transformer, 2020.

[55] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. volume Vol. 4304, pages 1015–1021, 01 2006.

[56] Yu Song, Santiago Miret, Huan Zhang, and Bang Liu. Honeybee: Progressive instruction finetuning of large language models for materials science, 2023.

[57] Harald Steck, Chaitanya Ekanadham, and Nathan Kallus. Is cosine-similarity of embeddings really about similarity? In *Companion Proceedings of the ACM on Web Conference 2024*, volume 201 of *WWW '24*, page 887–890. ACM, May 2024.

[58] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[59] 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.

[60] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang,

Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[61] Akshat Gupta Utkarsh Sharma and Sandeep Kumar Gupta. The pertinence of incorporating esg ratings to make investment decisions: a quantitative analysis using machine learning. *Journal of Sustainable Finance & Investment*, 14(1):184–198, 2024.

[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[63] Aishwarya Vijayan. A prompt engineering approach for structured data extraction from unstructured text using conversational llms. In *6th International Conference on Algorithms, Computing and Artificial Intelligence, ACAI 2023, Sanya, China, December 22-24, 2023*, pages 183–189. ACM, 2023.

[64] Guan Wang, Sijie Cheng, Xianyuan Zhan, Xiangang Li, Sen Song, and Yang Liu. Openchat: Advancing open-source language models with mixed-quality data, 2024.

[65] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. Codet5+: Open code large language models for code understanding and generation, 2023.

[66] 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.

[67] L. Weston, V. Tshitoyan, J. Dagdelen, O. Kononova, A. Trewartha, K. A. Persson, G. Ceder, and A. Jain. Named entity recognition and normalization applied to large-scale information extraction from the materials science literature. *Journal of Chemical Information and Modeling*, 59(9):3692–3702, 2019. PMID: 31361962.

[68] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization, 2019.

[69] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2020.

[70] Gokul Yenduri, Ramalingam M, Chemmalar Selvi G, Supriya Y, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, Deepti Raj G, Rutvij H Jhaveri, Prabadevi B, Weizheng Wang, Athanasios V. Vasilakos, and Thippa Reddy Gadekallu. Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions, 2023.

[71] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. `https://D2L.ai`.

[72] Xintong Zhao, Jane Greenberg, Yuan An, and Xiaohua Tony Hu. Fine-tuning bert model for materials named entity recognition. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 3717–3720, 2021.

[73] Yi Zou, Mengying Shi, Zhongjie Chen, Zhu Deng, ZongXiong Lei, Zihan Zeng, Shiming Yang, HongXiang Tong, Lei Xiao, and Wenwen Zhou. Esgreveal: An llm-based approach for extracting structured data from esg reports, 2023.