MASTER'S THESIS 2024

Leveraging Large Language Models for Event Extraction

Jonathan Desnoyer

ISSN 1650-2884
LU-CS-EX: 2024-62

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-62

Leveraging Large Language Models for Event Extraction

Jonathan Desnoyer

Leveraging Large Language Models for Event Extraction

Jonathan Desnoyer
jo7364de-s@student.lu.se

September 26, 2024

Master's thesis work carried out at QuantCube Technology.

Supervisors: Carla Martin, c.martin@quant-cube.com
Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

In light of generative AI's recent explosion in popularity, a lot of research is being made to leverage large language models that have demonstrated impressive capabilities across numerous fields. These developments could be advantageous for event extraction, as previous methods depended heavily on large annotated datasets and lacked generalizable methodology. Using three recent large language models (Phi-2, Phi-3, and Mistral 7B) that are categorized as small language models, we investigate the process of fine-tuning for the event extraction task. We show that fine-tuning these models enhances the results on both document and sentence-level event extraction, with Mistral 7B outperforming Phi-2 and Phi-3 on both levels. Document level extraction being a challenging task, reaching 0.329 and 0.089 F1-score against 0.576 and 0.316 for sentence-level extraction on trigger and argument extraction, respectively.

Keywords: Natural Language processing, Event Extraction, Large Language Models, Prompt Engineering, Fine-tuning, Embeddings, LORA, QLORA, GPU, Transformers

Acknowledgements

I would like to express my special appreciation and thanks to :
Prof. Pierre Nugues, for his continuous advice, patience and encouragements over these past months, providing his scientific expertise as well as his tutoring experience.
Carla Martin for giving me the opportunity to engage in such an interesting project and for her thoughtful guidance throughout.
The entire QuantCube team, for their kindness and the enriching exchanges we've shared.
My family and friends for their love and support.

Contents

Chapter 1
Introduction

With the exponential growth of unstructured data from news articles, social media, and web pages, *information extraction* (IE) has become a key task in natural language processing (NLP). With its emergence, we needed automatic methods to make use of this data, extract key information, and convert it to a structured format for easy analysis and use. This led to a specific task in the research domain: *event extraction*. The goal of event extraction is to detect events from trigger words and retrieve arguments related to these events, answering questions like "who", "when", "where", "what", "why", and "how". Recent approaches focus on deep learning models and transformers (Li et al., 2022) that combine multiple steps to perform end-to-end event extraction.

However, most deep learning-based methods require large quantities of annotated data for training. This process is costly and labor-intensive. Recently, the emergence of *large language models* (LLMs) has sparked significant interest in the NLP field. A lot of work is now focused exploring LLMs' capabilities for various tasks, including event extraction. LLMs offer the advantage of requiring minimal fine-tuning and demonstrate proficiency in understanding complex semantic relationships within text.

In context learning or prompt engineering has emerged as a method to leverage the capabilities of LLMs by providing specific instructions to guide them towards desired outputs without modifying the core model parameters (Sahoo et al., 2024). This approach allows users to interact and steer these models to achieve specific tasks. However, some challenges remain: although pre-trained language models possess vast language knowledge, they can lack specialization in specific areas. Fine-tuning pre-trained models can address this issue by exposing the model to task-specific examples.

We investigate the capabilities of large language models for event extraction by benchmarking their performance on an event extraction dataset, on both document and sentence level. Specifically, we employ Microsoft's Phi-2 and Phi-3 mini 4k instruct models, as well as the Mistral 7B instruct v2 model, evaluating them both with and without fine-tuning. Our findings indicate that while fine-tuning consistently improves model performance, the overall results remain modest, especially on the document-level extraction.

1.1 Objective

The goal of this Master's thesis, is to explore the use of large language models in the context of event extraction on both sentence and document-level event extraction. We explore the fine-tuning process across multiple models to assess how model size impacts the outcomes of the fine-tuning process and if they can be used for event extraction with optimization techniques such as quantization.

1.1.1 Research Questions

To address our objective, we formulate the following research questions:

1. How effectively do large language models perform when fine-tuned for event extraction tasks?

2. To what extent does the size of language models influence their performance in event extraction?

3. What is the effect of dataset constraints on the performance of fine-tuned models in event extraction?

1.2 Related Work

This Master's thesis investigates how LLMs can be used in the field of event extraction, expanding on earlier studies in LLMs and NLP techniques relevant to this endeavor. It enhances its contributions by utilizing state-of-the-art (SOTA) results from several fields of NLP.

Event extraction has long been a focus in NLP, with recent advancements fueled by the transformers architecture (Vaswani et al., 2017) and pre-trained language models. Newer, lighter, and faster language models offers opportunities for further exploration.

More recently, research has emphasized enhancing LLMs' capabilities through two primary paradigms: increasing the number of model parameters to improve their performance on a wide range of tasks, and developing smaller, specialized models tailored for specific tasks. This thesis makes use of recent advancements in LLMs, emphasizing quantization methods and the fine-tuning process. Specifically, recent work on smaller models, known as small language models (Microsoft, 2024), that solve the difficulties brought on by constrained computer resources has made it conceivable.

1.2.1 Historical Context and Evolution

We provide a brief overview of significant turning points in the domains of large language models and event extraction.

Event Extraction

In early efforts to event extraction, feature-based machine learning techniques or rule-based systems were frequently used (Ahn, 2006). However, these methods were limited in their ability to capture complex semantic relationships and often required extensive domain expertise for feature engineering.

The introduction of neural network-based approaches marked a significant change in event extraction methodologies. As both local and long-range, textual relationships were better captured by convolutional neural networks (CNNs) and recurrent neural networks (RNNs), especially Long short-term memory (LSTM) networks (Nguyen et al., 2016). More recently, transformer-based models have become the standard in event extraction tasks (Liu et al., 2018).

Large Language Models

LLMs have revolutionized natural language processing in recent years, demonstrating remarkable capabilities across a wide range of tasks. These models, based on the transformer architecture (Vaswani et al., 2017), are trained on vast amounts of text data and have shown particularly good performance in various NLP tasks, including text generation, question answering, and text classification.

The development of LLMs has been characterized by a rapid increase in model size and complexity. GPT-3 being a perfect example (Brown et al., 2020), reaching 175 billion parameters, this model was used to showcase large language models few-shot learning capabilities on various sets of tasks. However, these efforts towards larger and larger models have raised concerns about computational resources, environmental impact, and accessibility. This has led to research in several key areas:

- Efficient pre-training and fine-tuning techniques, such as parameter-efficient fine-tuning methods like LoRA (Hu et al., 2022).
- Exploration of small language models that aim to achieve competitive performance with significantly fewer parameters (Microsoft, 2024).
- Development of instruction-tuned models that can follow natural language instructions to perform various tasks (Wei et al., 2022a).

1.3 Scientific Contribution

This Master's thesis contributes to the advancement of natural language processing by investigating the fine-tuning process of large language models on an event extraction dataset. We evaluate how well the fine-tuning process improves the results on both document and sentence-level extraction for three recent small language models: Mistral 7B, Phi-2 and Phi-3. Additionally, we explore the effectiveness of the fine-tuning process with various optimization techniques, including LoRA and QLoRA.

We observed overall improvements across all models on both sentence-level and document-level extraction tasks, especially at the document-level, where the base models initially performed poorly. Additionally, our results demonstrate that Mistral 7B outperforms Phi mod-

els, indicating its superior capabilities for information extraction tasks. These findings highlight the potential of large language models for extracting information from text, even when constrained by lesser computational resources.

Chapter 2
Theoretical Background

In this chapter, we present in detail the technical aspects covered in this thesis. The first section delves into natural language processing techniques and the current breakthroughs tied to large language models. In a second section, an overview of the area of event extraction is provided before going into detail on the particular goal of event extraction and various methodologies.

2.1 Natural Language Processing

The following section provides the detailed description of all natural language processing techniques discussed in this Master's thesis.

2.1.1 Word Embeddings

Natural language processing is all about making words understandable by machines. Machines cannot read like humans, so we provide a numerical representation of words for them to understand. A naive approach is to convert every word in the dictionary to a vector of zeros and put a one in the vector at the position of the word. This is called one-hot encoding. However, this method suffers from the curse of dimensionality (Bera et al., 2021). As the dictionary size grows, the vectors become very large and sparse. Additionally, it does not give any contextual information between words.

To tackle this problem, Bengio et al. (2003) introduced word embeddings. This method projects words into a vector space with a fixed dimension, capturing relationships between words. For example, for a correct space representation, the words "Queen" and "Princess" would be close together because they refer to similar concepts. Conversely, "King" and "Queen" would have opposite vectors as they are antonyms. Traditionally, these word embeddings are constructed using neural network models, such as word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014).

Recently, more sophisticated word embedding techniques that leverage language models have been created. Contextual embedding models, such as ELMo (Embeddings from Language Models) (Peters et al., 2018), have emerged as a significant improvement over traditional static word embeddings. These models generate dynamic word representations that adapt based on the surrounding words, allowing for a more nuanced capture of word semantics and polysemy. Furthermore, large-scale transformer-based language models employ self-attention mechanisms (Vaswani et al., 2017) to generate embeddings that effectively capture long-range dependencies in text.

2.1.2 Transformers

Ever since Vaswani et al. (2017) presented the transformer architecture, it has gained enormous attention within the field of natural language processing. This new architecture, driven by the mechanism of self-attention, has become a cornerstone in NLP.

Vaswani et al. (2017) applied first transformers in machine translation and extended to any task that transforms an input sequence to an output sequence. Their introduction marked an improvement over traditional sequential models, such as RNNs and LSTMs, which often had difficulty with capturing long-range dependencies in sequences.

The self-attention mechanism allows transformers to weigh the importance of different words in a sequence in order to capture relationships and dependencies across varying distances within text. This architecture has proven itself by achieving SOTA results on a wide variety of NLP tasks, including, to cite a few, machine translation, text summarization, sentiment analysis, and question-answering.

In this section, we will cover the components of the transformer's architecture, providing explanations of self-attention mechanisms, multi-head attention, position-wise feed-forward networks, positional encoding, the transformer's architecture, and the transformer decoder-only architecture.

Self-Attention

Self-attention is the core of transformers. Transformers are the foundation of most modern language models and employ self-attention to capture context within a sequence. Traditionally, sequential models processed input data sequentially, often struggling to model relationships in long sequences. Transformers, on the other hand, can dynamically focus on different segments of the input sequence thanks to the attention mechanism, by assigning varying degrees of importance to each element based on its relevance to the current processing step.

Attention works by attributing a weight to each token in function of their importance with the current token. Hence, it allows each element in the sequence to attend to tokens that are most relevant to them. This is achieved by computing attention scores through a learned linear transformation of the input embeddings, followed by a softmax activation to obtain normalized weights. The resulting weighted sum is used to determine the most probable next token.

Also known as "Scaled Dot-product Attention", this mechanism is mathematically expressed as shown in the equation, as illustrated in Figure 2.1.

Given an input sequence of vectors $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, where $\mathbf{x}_i$ represents the embedding of the i-th token in the sequence, the self-attention mechanism computes attention

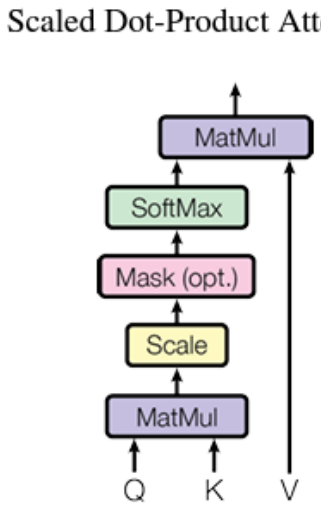


Figure 2.1: Scaled Dot-Product Attention. After Vaswani et al. (2017).

scores, $\mathbf{A}$, as follows:

$$\mathbf{A}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}.$$

Here, $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are linear projections of the input vectors $\mathbf{X}$, representing queries, keys, and values, respectively. The division by $\sqrt{d_k}$ scales the dot-product to prevent the gradients from becoming too small during backpropagation. The softmax function normalizes the attention scores across the sequence, ensuring that the weights sum to 1. The attended output is then calculated as the weighted sum of the values.

Multi-Head Attention

Instead of computing a single attention score, Vaswani et al. (2017) realized it was more beneficial to use multiple parallel attention modules (see Fig. 2.2), known as heads, with distinct learned linear projections. The idea behind this is that it will allow the model to focus better on the different parts of the semantic space. The resulting outputs from these individual heads are then combined through concatenation and linear operations to generate the ultimate self-attention output.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h) \cdot \mathbf{W}^O.$$

Here, each head_i corresponds to an attention head, and $\mathbf{W}^O$ is the weight matrix for the linear combination. The specific formulation of each attention head is given by:

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

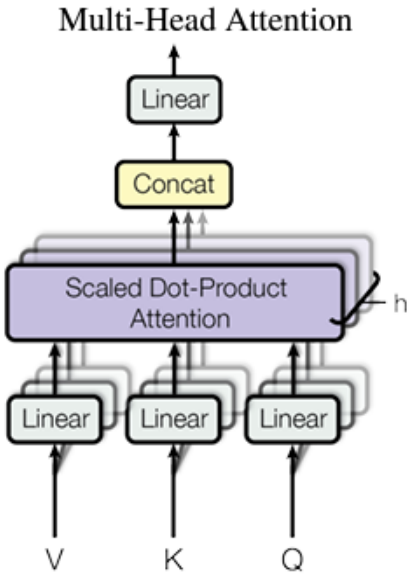


Figure 2.2: Multiple Head-Attention. After Vaswani et al. (2017).

Position-Wise Feed-Forward Networks

In each encoder layer, positioned after the multi-head self-attention mechanism, the feed-forward network consists of fully connected layers that operate independently on each position. This position-wise nature enables the network to process different positions within the sequence, capturing intricate patterns and local dependencies. The feed-forward network employs two linear transformations and a non-linear activation functions, the rectified linear Unit (ReLU), facilitating the extraction of complex features.

$$\mathrm{FFN}(\mathbf{X}) = \mathrm{ReLU}(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2.$$

The linear transformations are the same for all positions but can have different parameters from layer to layer.

Positional Encoding

So far, this model has no way of capturing the position of the tokens in a sequence. To make up for it, the authors decided to introduce "positional encoding" to the input embeddings. These encodings allow the model to track the position of each word in the sequence, helping it to distinguish between words with the same content but different positions.

The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. Each dimension in the embeddings is associated with a distinct sinusoidal wave, characterized by varying frequencies. The embedding's early dimensions are linked to low-frequency waves, while the frequencies progressively rise towards the end of the embedding. Additionally, the choice between sine and cosine waves alternates based on the parity of the embedding's dimensions. To obtain a positional embedding, the position value is inserted into this series of waves:

$$\begin{aligned} \mathrm{PE}(pos, 2i) &= \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \\ \mathrm{PE}(pos, 2i+1) &= \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right). \end{aligned}$$

Here, pos represents the position of the token in the sequence, i denotes the dimension, and d_{model} is the dimensionality of the model.

Transformer's Architecture

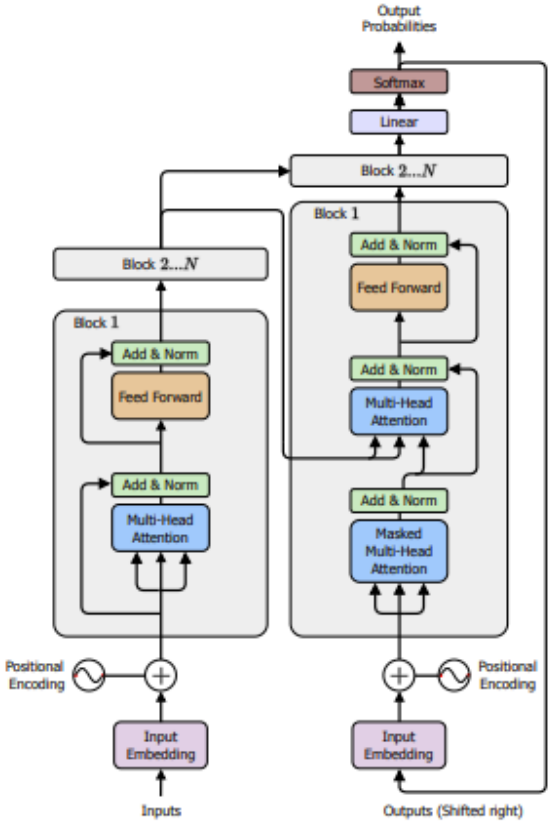


Figure 2.3: The transformer architecture from Vaswani et al. (2017), with the encoder block (left) and decoder block (right).

The transformer architecture was first used with both the encoder and decoder blocks, as shown in Figure 2.3. It was mainly used for sequence-to-sequence tasks, such as machine translation or text summarization, where the encoder block transforms the input sequence into a latent space representation that is then fed to the decoder block that generates the output sequence.

Encoder block

As described in Figure 2.3, the encoder block comprises multiple layers of N identical units, where each encoder layer consists of two main sub-components: the multi-head self-attention module and the position-wise fully connected feed-forward network.

As previously mentioned, the multi-head self-attention mechanism allows the encoder to capture complex relationships among various parts in a sequence. Concurrently, the position-wise feed-forward network makes it easier to extract intricate characteristics by individually performing linear transformations to every position.

Each encoder layer uses residual connections and layer normalization, contributing to stable and efficient training of the model. By stacking these encoder layers, a deep architecture is produced, which enables the transformer to gradually improve its representation of the input data. The output of the encoder serves as context-aware representation for the decoder block but can also be used for downstream tasks.

Decoder block

The decoder architecture is designed to generate output sequences by attending to the encoded representations of the input sequences produced by the encoder. Similar to the encoder, each block contains a masked multi-head attention submodule, a feedforward network, and several layer normalization operations. Blocks are put in sequence to make the model deeper. The output of the last block is fed through one more linear and softmax layer to obtain the final output of the model.

The decoder transformer architecture is frequently referred to as auto-regressive because it processes the input sequence sequentially such that each output token is generated based on the previous tokens. This auto-regressive process is ensured by the masked multi-head attention module. The encoded input sequence and the output are then sent into a standard multi-head attention block. This allows the model to handle the tokens that were previously created in addition to the input sequence.

Transformer's Decoder-Only Architecture

The original transformer model consists of both an encoder and a decoder. Many models, nevertheless, only make use of one of these blocks. Encoder-only models like BERT (Devlin et al., 2018) are used for natural language understanding (NLU) tasks such as text classification and named entity recognition (NER). In contrast, decoder-only models are used for natural language generation (NLG) tasks (Tunstall et al., 2022).

Decoder-only models repeatedly forecast the next token in a sequence until a unique end token is generated, this process is frequently referred as *inference*. These models employ causal or autoregressive attention, which are representations depending on the left context. This includes the *generative pretrained transformer* (GPT) family (Radford et al., 2018).

As seen in Figure 2.4, the decoder-only architecture is similar to the encoder-decoder architecture but it does not include the encoder block. As a result, the multi-head attention layer is not linked to the encoder output. A feed-forward layer and a masked multi-head attention layer make up each decoder block.

2.1.3 Large Language Models Pre-Training

Pre-training is the first phase of training a large language model. It is an important process that allows these models to learn to generate and understand large amounts of language. Pre-training usually entails exposing the model to large amounts of heterogeneous textual data so that it can learn unsupervised patterns, relationships, and language representations.

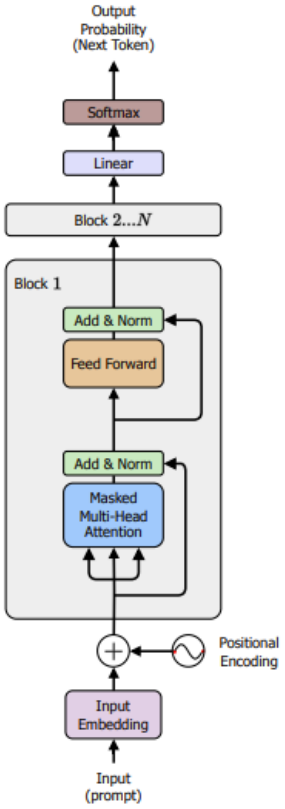


Figure 2.4: Transformer's decoder-only architecture.

The pre-training stage of LLMs might use a variety of pre-training tasks. One of the most common tasks is left-to-right language modeling, as utilized in models like GPT (Radford et al., 2018). This method trains the model to predict the next token in a sequence given a set of tokens. To be more precise, the model looks at the order in which the final token has been eliminated and makes an attempt to predict it. Through capturing syntactic and semantic patterns, this step aids the model in learning the distribution of tokens in the language. The model modifies its weights and biases in back-propagation in response to the prediction error. The error is calculated by comparing the predicted token with the actual token. Through this iterative process, the model is able to learn the language's fundamental structure and gradually improve its predictions.

A different strategy is masked language modeling (MLM), in which the model's task is to forecast the masked tokens within a sentence by masking random tokens (Devlin et al., 2018). Because it must deduce the masked token from the surrounding words, this approach compels the model to gain a deeper comprehension of the sentence's overall context. Understanding complicated language patterns is made easier by MLM's ability to teach the model bidirectional representations, which capture context from both preceding and succeeding tokens.

2.1.4 In-Context Learning

In-context learning, also known as *prompt engineering*, consists in giving instructions to guide LLMs towards a desired output for a given task. This method, was first presented by Brown et al. (2020) in their seminal work on GPT-3, and it represents a significant change in how we interact with and use LLMs.

This concept is closely related to the concept of *few-shot learning*, in which we provide the model instances of instructions and desired responses so that it can make generalizations and complete the task for unknown examples. This ability to draw generalizations from a limited set of cases seems to emerge as the scale of the model increases and has been trained on a variety of datasets.

One area of recent study interest has been on improving LLMs' reasoning ability. It has been demonstrated that using chain-of-thought (CoT) prompting (Wei et al., 2022b) improves the reasoning of the model and improves its performance on certain tasks. It improves the model's reasoning process in a way that is similar to human cognitive problem-solving by enabling models to decompose complicated problems into smaller, more manageable tasks.

2.1.5 Fine-tuning

Fine-tuning LLMs involves additional training on a smaller, domain-specific dataset of a pre-trained model (PML), often referred to as a foundation model, which has undergone the process described in Section 2.1.3. While LLMs have shown remarkable performance on a variety of tasks even in zero-shot settings, fine-tuning allows us to adapt the model to a specific task. This process is really useful since it avoids us having to train a large language model from scratch, which requires a lot of time and computational power. And it allows the model to achieve better performances on a specific task with a large reduction in the amount of data and computing power needed by utilizing information and capabilities already present in the PML.

Full Fine-Tuning (Instruction fine-tuning)

Full fine tuning, also known as instruction tuning, is a supervised training process of LLMs on a dataset made up of (instruction, output) pairs. This process adapts a pre-trained model to perform specific tasks (Zhang et al., 2024). It produces a new version with improved features, by updating all of the model weights. However, this approach needs a lot of memory and computational power to handle the gradients, optimizers, and other elements during the training phase, just like pre-training does.

Parameter-Efficient Fine-Tuning

Alternatively, Houlsby et al. (2019) presented a more resource-efficient method than full fine-tuning that led to the development of *parameter-efficient fine-tuning* methods. While full fine-tuning requires substantial computational resources and memory allocation, not only for storing the model but also for managing parameters during training, parameter-efficient fine-tuning methods only targets a subset of layers and keeps the rest of the model frozen. In

addition to significantly reducing the memory needs, this helps the model retain the majority of its knowledge and prevents catastrophic forgetting (Liao et al., 2023).

Sparse fine-tuning and infused fine-tuning are the two main categories of parameter-efficient fine-tuning methods (Liao et al., 2023). Sparse fine-tuning involves selectively adjusting a small subset of existing parameters without introducing new ones. On the other hand, infused fine-tuning introduces new parameters to the PLM and updates only these additional parameters during training. An example of infused fine-tuning that was widely used, prior to LoRA (see next Section 2.1.5), is adapter fine-tuning, where one or multiple small multilayer perceptron (MLP) modules are inserted into each layer of the PLM. However, because it adds new sequential layers, this second technique increases inference time.

Low-Rank Adaptation (LoRA)

Nowadays, low-rank adaptation (LoRA) is the most popular method for effectively fine-tuning LLMs. LoRA, first presented by Hu et al. (2022), addresses three key aspects of the fine-tuning procedure. Firstly, it significantly lowers memory needs and allows LLMs to be optimized with just one GPU. Secondly, by lowering the number of trainable parameters, it accelerates and lowers the cost of the training process. Last but not least, it enables users to keep a single instance of an LLM with several LoRA adapters for multiple tasks that may be inverted using straightforward matrix operations.

Fine-tuning a language model involves modifying its underlying parameters. This modification can be represented as:

$$h = W_0 x + \Delta W x$$

In contrast to the standard fine-tuning approach, LoRA introduces a more efficient approach. As depicted in Figure 2.5, LoRA models this parameter update using a low-rank decomposition, implemented through a pair of small linear projections. Instead of altering the pre-trained layers, LoRA keeps them fixed and injects a trainable rank decomposition matrix into each target layer:

$$h = W_0 x + W_A W_B x$$

where $W_0 \in \mathbb{R}^{d \times d}$ is the pre-trained weight matrix, $\Delta W \in \mathbb{R}^{d \times d}$ is the updated weight matrix which is decomposed into low-rank matrices $W_A \in \mathbb{R}^{d \times r}$ and $W_B \in \mathbb{R}^{r \times d}$. The matrix W_A is initialized with small random values, while W_B is set to zero, ensuring that the fine-tuning process begins with the model's original pretrained weights.

2.1.6 Memory Optimization Techniques

Quantized Low-Rank Adaptation (QLoRA)

Quantized Low-Rank Adaptation (QLoRA) was introduced by Dettmers et al. (2023) as an improvement to LoRA that uses quantization to reduce memory requirements in LLMs. Quantization consists in lowering the precision of the data types used for the model's parameters.

Figure 2.6 illustrates how QLoRA uses a number of innovations to lower memory usage without compromising performance:

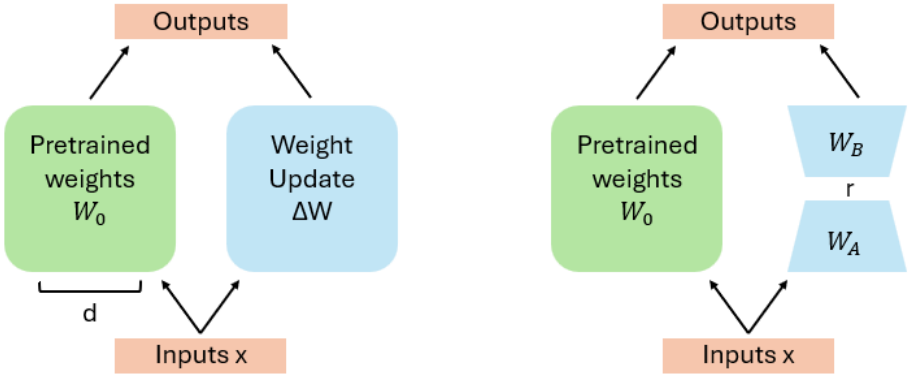


Figure 2.5: Regular fine-tuning process where weight update is $h = W_0x + \Delta Wx$ (left) vs. LoRA process with $h = W_0x + W_AW_Bx$ (right).

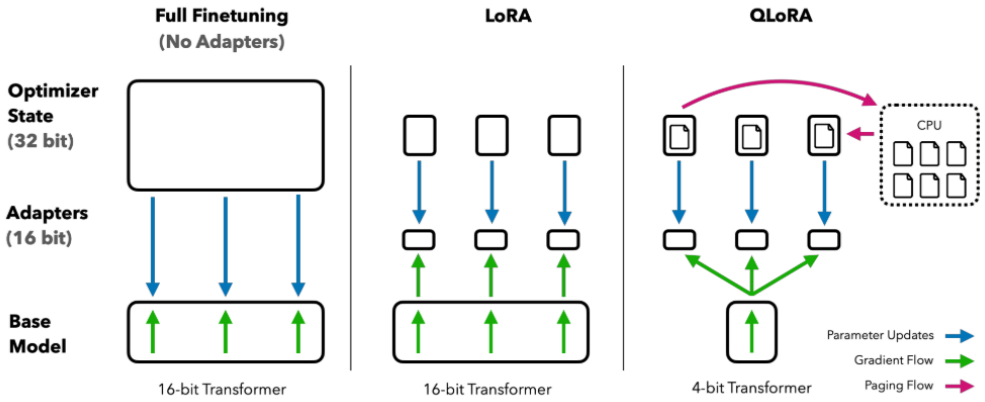


Figure 2.6: Comparison of different fine-tuning methods and their memory requirements. After Dettmers et al. (2023).

1. **4-bit NormalFloat (NF4):** To quantize the weights of pre-trained language models, QLoRA use NF4. NF4 is optimized for normally distributed weights, typical in neural networks.

2. **Double Quantization:** QLoRA employs double quantization, reducing the quantization constants themselves from 32-bit to 8-bit.

3. **Paged Optimizers:** To manage memory spikes during training, QLoRA uses paged optimizers leveraging NVIDIA's unified memory feature. Automatic page-to-page transfers between CPU and GPU memory are made possible by this.

Gradients are backpropagated into the LoRA adapters during training via the frozen, 4-bit quantized pre-trained language model. QLoRA employs 16-bit BrainFloat for compute and NF4 for storage data types. For both forward and backward passes, the storage data type is dequantized to the computation data type.

Gradient Checkpointing

Gradient checkpointing is a technique used to reduce memory usage during training of a deep neural network, and it can also be used when fine-tuning LLMs. It trades off increased computation time for memory usage and can therefore reduce the need for GPUs with large VRAM.

In practice, for the training of deep neural networks, the input data is passed through the entire network in the forward pass. This involves computing at each layer intermediate activation values in order to compute the final output. Then, in the backward pass, the gradients of the loss function with respect to the model parameters are computed using the stored intermediate activation values. Finally, the gradients are used to update the model parameters.

As you can imagine, storing these intermediate activation values during forward pass can take quite a lot of memory and can lead to a bottleneck in terms of batch size or simply size of the model during the training of a deep neural network. Gradient Checkpointing addresses this challenge by storing only a subset of intermediate activation values, known as checkpoints, during the forward pass. And then during backward pass, the intermediate activation values that weren't stored are recomputed using the cached checkpoint activation values. It reduces the memory footprint from a linear scale that is proportional to the number of layers n to approximately a square root scale $\sqrt{n}$ (Singh et al., 2024).

Mixed Precision Training

Mixed precision training optimizes different parts of the model using varying numerical precisions during training. FP16, a half-precision floating-point format that uses 16 bits for number representation, is commonly employed for its capacity to conserve memory and accelerate computations (Narang et al., 2018).

QLoRA (see Section 2.1.6) and mixed precision training can be employed during the fine-tuning process. The pre-trained weights of the frozen model will be quantized to 4-bit, just like in QLoRA. While FP16 will be utilized in the LoRA adapters and computational steps. Gradients passing through these adapters during backpropagation are in FP16, and optimizer updates for these adapters are also computed in FP16. With this strategy, we can take advantage of both optimization techniques.

2.1.7 Models

For this Master's thesis, I chose to use three different language models: Mistral 7B instruct v2, Phi-2, and Phi-3 mini 4k instruct. While Mistral 7b instruct v2 is a member of the Mistral series, which was created by Mistral AI (Jiang et al., 2023), the Phi models were created by Microsoft as a part of a collection of small language models (SLMs).

Although Phi-2 and Phi-3 mini have different sizes and functionalities, they have similar architectures. There are 2.7 billion parameters in Phi-2, and 3.8 billion in Phi-3 mini. Both models were pre-trained on high-quality, textbook-like data, contributing to their performance which is on par with much bigger LLMs. Phi-3 mini 4k instruct benefits from an improved training dataset compared to its predecessor as well as an instruction fine-tuning stage. The context window sizes vary among these models. Phi-2 can have trouble digesting

longer texts because of its 2048 token context window. There are two options for the context window on Phi-3 mini: 4k and 128k tokens. We used the 4k context window for this investigation because it met our needs.

Mistral 7b instruct v2 is almost twice as large as Phi-3 mini, with 7 billion parameters. With its architecture combining sparse mixture of experts and sliding window attention, Mistral 7B instruct v2 can handle lengthy sequences—up to 32k tokens—in an efficient manner. Hugging Face open-source instruction datasets were used to refine Mistral 7B instruct v2, improving its capacity to carry out a variety of tasks and adhere to specific instructions. The model should be more adaptable and usable than its counterpart thanks to this additional instruction training phase.

We will refer to Phi-2 model as Phi-2, Phi-3 mini 4k instruct as Phi-3, and Mistral 7B instruct v2 as Mistral 7B throughout the remainder of this thesis.

2.2 Event Extraction

2.2.1 Information Extraction

The need to make sense of the massive amounts of unstructured text data that are available in several sources, including documents, web pages, and social media, gave rise to the field of information extraction (IE). As digital content expanded at an exponential rate, automated techniques were required to extract relevant information from this unstructured data and transform it into formats that could be readily accessed, analyzed, and used.

Information extraction is a field that uses NLP techniques to extract important information from unstructured data and transform it into structured formats for simpler use and management. It involves several steps, such as *name entity recognition* (NER) and *relation extraction*, to mention a couple. NER focuses on identifying and categorizing named entities, such as people, organizations, locations, dates, and numerical expressions, within the text. Once these entities are identified, relation extraction comes into play by analyzing the relationships between them. Relation extraction involves identifying and classifying semantic relationships between entities mentioned in text. Relation extraction would ascertain the relationship between an entity and a person, such as employment or ownership, whereas NER would identify the entities in a statement citing a person or a firm.

2.2.2 Event Extraction Task

In IE, event extraction is a difficult and well-studied task. Its goal is to extract structured information from unstructured text. Event extraction is often separated into two categories: closed-domain event extraction and open-domain event extraction (Li et al., 2022). Events in closed-domain are frequently given predefined schemas to indicate which particular events and arguments should be extracted. Conversely in open-domain event extraction, events are defined as a list of descriptions and the problem is then reformulated into a clustering or classification task. In both cases, the goal is to detect events and their arguments from a predefined list and output it in a structured format. As illustrated in Figure 2.7, the aim of event extraction is to extract event triggers and arguments from a text. It can be done on document-level but also on sentence-level.

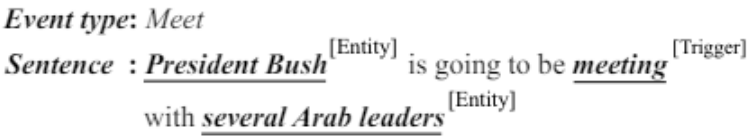


Figure 2.7: Example of an event of type *Meet*. After Yang et al. (2019).

Event Extraction Key Concepts

An event is an occurrence of an action or a shift in status that is frequently brought about by verbs or gerunds. Arguments are the components of an event that give more information or background. They include elements like places, times, entities, and so on. Event extraction technologies are capable of extracting all mentions of events from a text, together with their corresponding triggers and parameters (Liu et al., 2021).

The following broad concepts are definable:

Event mention: The sentence that refers to an event and includes the arguments and trigger;

Event type: Characterizes the kind of event and typically matches the kind of event trigger;

Event trigger: Frequently a verb, it is the crucial element required to identify a reference of an event;

Event argument: Refers to the "attributes" of an event and can include entities, places, agents, etc.;

Argument role: Corresponds to the role played by an argument in an event.

Event Extraction Problem Formulation

Event Extraction can be seen as four subtasks (Li et al., 2022): trigger extraction, trigger classification, argument extraction, and argument role classification:

- **Trigger Identification**
  Most people agree that the trigger is the fundamental component of event extraction that can accurately describe the occurrence of an event.
- **Trigger Classification**
  The classification of a trigger serves to identify the type of occurrence to which it pertains. For example, "meeting" in Figure 2.7 denotes the event "Meet". Therefore, trigger classification can be seen as a multi-label text classification task.
- **Argument Identification**
  Finding the arguments supporting a specific event mention is known as argument identification. It depends on the trigger identification and trigger classification processes in the majority of approaches.
- **Argument Role Classification**
  The event extraction schema's arguments serve as the foundation for the argument role classification, and each argument's category is categorized in accordance with the arguments that have been identified.

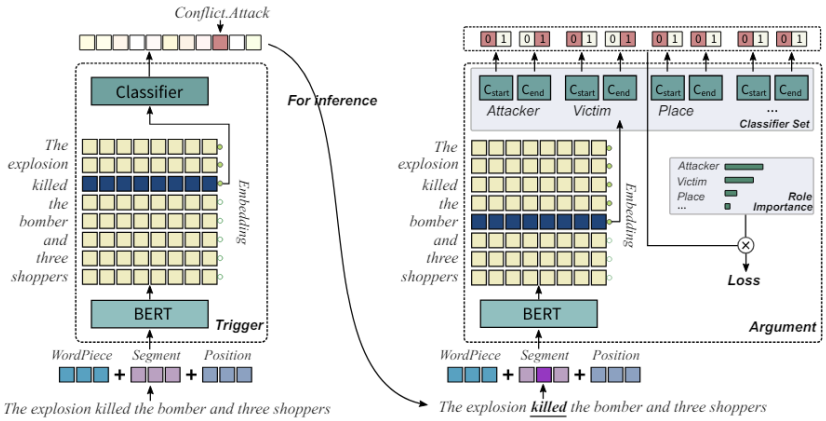


Figure 2.8: Example of pipeline event extraction architecture from Yang et al. (2019), a trigger extractor first performs the event detection before feeding the result to an argument extractor

2.2.3 Deep Learning Approaches

The majority of current methods, as stated in the introduction, have concentrated on employing transformers and deep learning models. Pipeline base models and joint models are the two basic paradigms that were established to do an end-to-end event extraction (Lu et al., 2021). Pipeline-based models first perform trigger identification and extract the event type according to the triggers. In a second step, the model then extracts arguments and classifies argument roles according to the prediction results of event type and the triggers. To avoid the error propagation, researchers have proposed joint models that combine trigger and argument extraction simultaneously.

Pipeline-Based Paradigm For Event Extraction

For pipeline-based methods, the four main subtasks are run sequentially. In practice, this type of model converts event extraction tasks into a multi-stage classification problem. The necessary classifiers are an argument classifier that ascertains whether a word is an argument of the event; a trigger classifier that ascertains whether a term is an event trigger and defines the kind of event; and an argument role classifier that establishes the category of the arguments.

As outlined in Singh (2018), pipeline models commonly include components like named entity recognition (NER), named entity linking (NEL), coreference resolution (CR), temporal information extraction, and relation extraction (RE). These tasks encompass lower-level NLP tasks such as parts-of-speech tagging (POST), chunking, and parsing.

This method's main flaw is that each subtask's accuracy determines how effective it is. Inaccuracies or mistakes in any of the intermediary steps have the potential to spread throughout the pipeline and reduce overall performance.

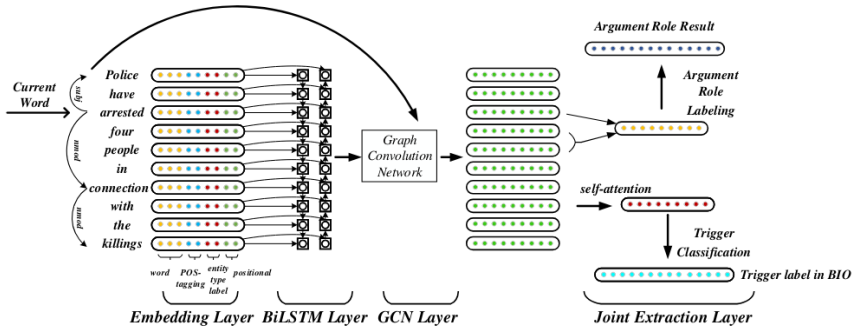


Figure 2.9: Joint model architecture from Liu et al. (2018).

Joint Model Paradigm For Event Extraction

In order to avoid the errors from propagating from one step to another, researchers proposed joint model architectures that perform trigger classification and argument classification simultaneously. As outlined in Liu et al. (2018), usually these model architectures first find a trigger candidate and then go over each (trigger, entity) pair to perform trigger and argument classification. This involves retrieving semantic representations for each word and relation tuple, in most cases using neural network encoders.

A joint model architecture, as illustrated in Figure 2.9, allows for simultaneous extraction of events and arguments from text. This approach has demonstrated improved performance compared to pipeline models that handle trigger prediction and argument extraction separately.

2.2.4 Large Language Models Approaches

Large language models, which have several advantages over deep neural approaches, have recently opened the door to possible new approaches. Compared to conventional methods, LLMs demonstrate a reduced reliance on extensive datasets and token-level annotations (Lu et al., 2021). Since LLMs process natural language inputs directly, they do not require the use of intermediate preparatory steps like part-of-speech tagging, dependency parsing, or coreference resolution.

Moreover, LLMs are remarkably flexible with respect to certain event schemas. In cases with little or no task-specific training data, Hsu et al. (2022) reported encouraging results, indicating LLMs can use pre-trained information to generalize across a variety of event types.

These developments point to a paradigm change in event extraction techniques, opening the door to more adaptable, effective, and broadly applicable strategies.

Chapter 3
MultiMedia Event Extraction Dataset

This chapter gives a deeper explanation on the dataset used in this Master's thesis, detailing how it was processed, formatted and used for both inference and fine-tuning for document and sentence-level extraction.

3.1 Overview

The MultiMedia Event Extraction (M^2E^2) dataset (Li et al., 2020) was initially created to demonstrate a new method for event extraction from multimedia news items that includes both visual and written content. In this work, I will only use the textual data from this dataset, though. The trigger event and arguments in this dataset are nested within the same phrase because it is intended for sentence-level event extraction. It contains 245 annotated multimedia news articles. After processing these with events in the texts, we end with 203 news articles. We use the raw articles and specific event annotations. For our research, the texts and event annotations are formatted into a question-answer format, adapted for inference and fine-tuning.

The event types are derived from the Automatic Content Extraction (ACE) community. This makes it possible to expand our work utilizing the multilingual ACE 2005 dataset (Walker et al., 2005), which addresses five main tasks: entity, value, temporal expression, relation, and event recognition.

3.2 Data Statistics

The dataset covers eight different types of events, with a total of 1105 event mentions across 203 articles. For each type of event, there are specific arguments. The different arguments and their counts are shown in Table 3.1 as well as the number of events per type.

Figure 3.1 shows a box plot with the number of events per article.

Event Type	Argument	Count
Conflict:Demonstrate (98)	Place	78
	Entity	50
Justice:Arrest-Jail (121)	Person	72
	Place	30
	Agent	29
Movement:Transport (229)	Destination	123
	Artifact	136
	Origin	78
	Agent	44
	Vehicle	21
Contact:Meet (104)	Entity	97
	Place	62
Life:Die (176)	Victim	133
	Agent	48
	Place	69
	Instrument	11
Conflict:Attack (302)	Attacker	166
	Target	152
	Place	134
	Instrument	25
Contact:Phone-Write (43)	Place	11
	Entity	43
Transaction:Transfer-Money (32)	Recipient	21
	Giver	23

Table 3.1: Number of events per type (indicated in parentheses) along with the different arguments and their counts, providing a detailed overview of the dataset's composition.

For the fine-tuning process, the dataset was split into a training, validation, and test set as described in Table 3.2.

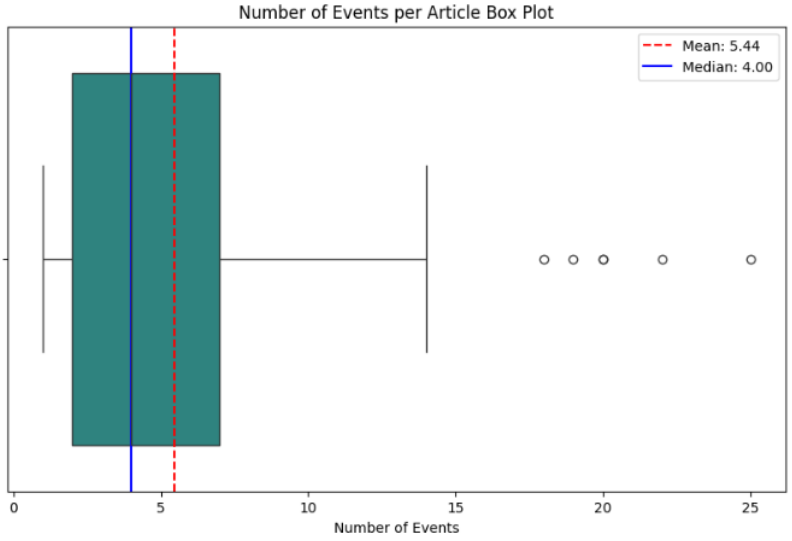


Figure 3.1: Box plot with the number of events per article

Event Type	Train Set	Validation Set	Test Set
Number of Articles	140	31	32
Conflict:Attack	209	40	53
Movement:Transport	167	36	26
Life:Die	120	27	29
Justice:Arrest-Jail	89	13	19
Contact:Meet	73	17	14
Conflict:Demonstrate	55	28	15
Contact:Phone-Write	30	5	8
Transaction:Transfer-Money	20	5	7
Total Number of Events	763	171	171
Grand Total Number of Events: 1105			

Table 3.2: Number of events per type for each dataset split (Train, Validation, and Test) along with the number of articles.

3.3 Data Formatting

We processed the dataset in two different ways to accommodate for document and sentence level event extraction. For both we kept the same split as described in Table 3.2.

3.3.1 Document Level Extraction

As said in Section 3.1, we transform the content of articles and annotations into a question-answer format. Given an article, the goal for our model is to extract data into a dictionary format as shown in the example below:

```
{
    'events': [
    {
        'event': 'Contact:Phone-Write',
        'trigger': 'conversation',
        'event_arguments': {'Entity': 'Teddy Riner'}

    },
    {
        'event': 'Contact:Meet',
        'trigger': 'meeting',
        'event_arguments': {'Place': 'France', 'Entity': 'Teddy Riner'}
    }
    ]
}
```

We give the model a list of event types, the available arguments for each event type, and contextual instructions in order to use in-context learning. The template used for document-level event extraction during the model's evaluation, both for inference and fine-tuning, is shown below:

```
### Instruct : You are an AI expert specialized in Event Extraction.
Your task is to identify triggers and arguments of events from this
list of event types: {event_types}
Here is a dictionary with the possible argument roles for each
event type: {arg_roles_by_event_type}
You will only extract arguments corresponding to these roles.
Triggers are clearly identifiable in the text, and arguments for a given
trigger are nested within the same sentence. Not every event has arguments,
and there should not be more than 5 events per article, except in rare cases.
Now for the following article : {article}
### Output : {annotation_dictionary}
### End
```

3.3.2 Sentence-Level Extraction

For sentence-level extraction we adapt the prompt and only use event mentions from the articles. So the target corresponds to the elements from the above dictionary, given this sentence:

> "And if he is arrested in the street, they might charge him with trying to get other people on the street to commit 'debauchery' as well."

The expected output is the following:

{

'event': 'Justice:Arrest-Jail',
'trigger': 'arrested',
'event_arguments': **{'Place':** 'street'}

}

Below is the template used for sentence-level event extraction for both fine-tuning and inference during the model's evaluation:

```
### Instruct : You are an AI expert specialized in Event Extraction.
Given a sentence, your task is to classify the sentence with an event from
this list of events type: {sorted_event_types}
You shall extract the trigger (word that indicates the event) and potential
arguments with their roles, given this mapping : {arg_roles_by_event_type}
Now for the following sentence:  {event_mention}
### Output : {annotation_dictionary}
### End
```

Chapter 4
Method

In this chapter, we provide a detailed explanation of the methodology employed in this Master's thesis, aligned with the research questions outlined in the introduction.

4.1 Overview

Before delving into the methodology, let's outline the key steps undertaken in this thesis. First, we retrieved and pre-processed an open-source dataset for event extraction tasks, for both document and sentence level. Next, we evaluated different models using two-shot approach on a test dataset to establish baseline performances. Subsequently, each model was fine-tuned using training and validation sets, applying techniques discussed in Chapter 2. Finally, we experimented with various parameter configurations to optimize model's performance.

4.2 Dataset Retrieval and Preparation

We conducted our research using a publicly available dataset that we preprocessed in compliance with the guidelines in Section 3.3 to facilitate experiment replication. The dataset is available from the GitHub repository of the original research paper (Li et al., 2020). This approach ensures that other researchers can build upon and validate our findings. The 2005 ACL dataset (Walker et al., 2005) could be added to the dataset, hence increasing its size and diversity of event categories, due to its flexibility. This extensibility provides an opportunity for a more comprehensive evaluation of LLMs capabilities and fine-tuning process in event extraction.

We prepared the dataset in two different ways for both document and sentence-level event extraction. For the document-level extraction the targets are created with the event mentions annotations and stored into a dictionary as described in Section 3.3. While for the

sentence-level extraction we used only the event mentions annotations, creating one data point for each event mention.

Our primary objective was to establish a benchmark for the fine-tuning process, using models with different sizes. Our goal in doing this was to gain greater insight into the connections between model size and performance on a small dataset.

4.3 Baselines

Prior to fine-tuning, the models must be evaluated in order to comprehend the advances brought about by the process and to gauge the corresponding abilities of the different models. Two-shot settings were used for this evaluation, utilizing in-context learning (for more information, see Section 2.1.4).

By establishing these baselines, it is possible to compare the models more accurately and provides a clear reference point to measure the impact of fine-tuning.

4.4 Fine-Tuned Models

One of the main focuses of this work is the fine-tuning method, which produced the best outcomes for our particular objective. This section will include a detailed description of the infrastructure, libraries, and quantization methods utilized, as well as steps required for fine-tuning.

4.4.1 Fine-Tuning Process Steps

Fine-tuning the different large language models involves several steps to ensure that the models are well-adapted to the specific task. The following subsections outline these steps in detail:

1. Data Preparation

The first step in fine-tuning is preparing the dataset. This involves cleaning and formatting the data to ensure it is suitable for training. This step includes tokenization, normalization, and splitting the data into training, validation, and test sets.

2. Model Initialization

The next step in the fine-tuning process is the initialization of the pre-trained model. In our study, we utilized pre-trained models from Hugging Face open source platform.

3. Configuring Hyperparameters

It is essential to set the hyperparameters for the training procedure. This include figuring out the number of steps/epochs, batch size, learning rate, and any other pertinent parameter. A hyperparameter search was not possible due to the size of the models and the calculation time needed for each training loop.

4. Validation

After each epoch or number of steps (25 in our case), the model's performance is evaluated on the validation set. This step helps monitoring overfitting and adjusting training accordingly. Unfortunately, this was only done on the loss function which isn't adapted to monitor performance on our task.

5. Checkpointing

Checkpoints are periodically saved to safeguard against potential disruptions and to facilitate the selection of the optimal model. These checkpoints allow training to resume from the last stored state, if needed, by storing the states of the deconstructed matrices.

6. Hyperparameter Tuning

It could be necessary to modify the hyperparameters in light of validation performance. To determine the ideal settings, this iterative approach entails adjusting the hyperparameters and repeating the training loop.

7. Final Evaluation

The test set is used to assess the model's performance once it has been fine-tuned. This final assessment uses precision, recall, and F1-score measures to clearly quantify how effectively the model generalizes to new data.

4.4.2 Infrastructure

It is crucial to have an infrastructure in place before beginning a data science project that can support the project's requirements. While fine-tuning large language models, it is important to have adequate GPUs and VRAM to conduct the training process and prevent hitting bottlenecks. These days, a wide range of companies offer a catalog of virtual machines with various specifications, from which you can select the one best suited to your needs.

AWS

Amazon Web Services (AWS) was chosen as the primary infrastructure as it provides powerful GPU-equipped instances that are ideal for fine-tuning large language models. For our research, we employed an AWS EC2 g4dn.12xlarge instance equipped with NVIDIA T4 GPUs. This instance offers a fair trade-off between cost and performance, with enough GPU memory and processing capacity to accommodate our models' fine-tuning (see Table 4.1).

Graphic Processing Unit (GPU)

The initial purpose of *graphics processing units* (GPUs) was to process images and videos for use in computer graphics applications. These are specialized processors made to handle numerous tasks concurrently. GPUs are particularly good at performing the intricate computations needed for graphics rendering, including transformations and shading. They are ideal

Table 4.1: Specifications of the g4dn.12xlarge Instance

Specification	Details
vCPUs	48
GPU	4 NVIDIA T4
GPU Memory	64 GB
Instance Memory	192 GB
Storage	900 GB NVMe SSD
Network Bandwidth	50 Gbps
EBS Bandwidth	Up to 9.5 Gbps

for parallel computing tasks because of their architecture, which has many cores capable of simultaneous processing. In the field of artificial intelligence, they are now widely utilized.

GPUs' parallel processing capabilities, which greatly speed up the training process, are essential for fine-tuning LLMs. They are able to handle the massive amounts of matrix and tensor operations required in deep learning, making them indispensable for this task. The choice of GPU can impact both the speed and cost-effectiveness of the training process.

Video Random Access Memory (VRAM)

One of the most common errors that can happen during fine-tuning is *"RuntimeError: CUDA error: out of memory"*. In addition to using optimization techniques, choosing a GPU with enough *Video random access memory* (VRAM) is the best way to avoid this error. VRAM or GPU memory plays a crucial role in the efficient operation of LLMs on GPUs. LLMs, such as those based on transformer architectures, often require significant VRAM due to their massive parameter sizes. VRAM is dedicated GPU memory that stores not only the model parameters but also the intermediate activation values and gradients during the training process. Insufficient VRAM can severely limit the size of models that can be trained or fine-tuned, leading to performance degradation or outright failure.

4.4.3 Libraries and Memory Optimization Techniques

In this thesis, we employed several advanced libraries and memory optimization techniques for the fine-tuning process. Each component played an important role in enhancing computational efficiency and memory utilization.

Transformers Library

Hugging Face created the renowned *Transformers* library, which we utilized (Hugging Face, 2024c). This library works with TensorFlow and Pytorch and offers cutting-edge general-purpose architectures for NLU and NLG. It enables us to load and fine-tune the most recent pre-trained language models for our investigations.

BitsAndBytes for Quantization

We used the BitsAndBytes package for quantization to meet the memory requirements of LLMs (Hugging Face, 2024b). In particular, this library allowed us to use the NF4 data type to import models in a 4-bit quantized format (see Section 2.1.6). BitsAndBytes allowed us to deploy larger models even on devices with limited GPU RAM by drastically lowering the memory footprint.

LoRA and QLoRA Integration

In our study, we integrated LoRA and QLoRA techniques using the PEFT (Parameter-Efficient Fine-Tuning) python library (Hugging Face, 2024a). As described before (see 2.1.5), LoRA, implemented through PEFT, focused on updating specific layers. This method involved configuring key parameters like the LoRA alpha learning rate (α), dropout rates, and the rank (r) of decomposition matrices, typically setting α to 1 or 2 times the value of r to optimize the integration of low-rank decomposition into the model's structure.

Building upon LoRA, QLoRA further enhanced memory efficiency by maintaining the majority of pre-trained model weights in a 4-bit quantized format. We used FP16 precision for smaller LoRA adaption layers that handle important computational tasks. This hybrid approach effectively combined the benefits of quantization—reducing memory footprint with the precision of half-precision floating-point computations, thereby optimizing both memory usage and computational performance in our experiments.

Gradient Checkpointing

To mitigate peak memory usage during training, we employed gradient checkpointing. This method involved storing intermediate activation values and recalculating them during the backward pass, effectively reducing the overall memory footprint while preserving computational accuracy (see Section 2.1.6).

SFTTrainer (Supervised Fine-Tuning Trainer)

To manage the fine-tuning process, we employed the *SFTTrainer* (supervised fine-tuning trainer). This class is provided by the TRL (transformer reinforcement learning) library, which is built on top of the Hugging Face transformers ecosystem. It easily combines with the PEFT library, making the implementation of LoRA and QLoRA simple.

4.5 Training Configuration

Identifying optimal hyperparameters is important in order to achieve the best possible results. In the context of fine-tuning, the rank of the LoRA decomposition matrices is one of the key parameters that significantly influence performance. Additionally, the selection of target layers plays a critical role, as it also impacts the number of trainable parameters.

We experimented with multiple configurations for each model to identify the most effective setup. The parameters used for the models discussed in the subsequent chapter are detailed in Tables 4.2 and 4.3. Across all models, we employed a specialized version of the

AdamW optimizer, which integrates paging and 8-bit quantization to enhance computational efficiency.

	Mistral 7B	**Phi-3**	**Phi-2**
LoRA Configuration			
lora alpha	8	32	32
lora dropout	0.1	0.05	0.05
r	8	16	16
target modules	q_proj, k_proj, v_proj, o_proj, up_proj	o_proj, k_proj, v_proj, dense	o_proj, k_proj, v_proj, dense
Training Arguments			
warmup steps	3	3	3
batch size	1	1	1
gradient accumulation steps	4	4	4
max steps	500	500	500
learning rate	2e-4	2e-4	2e-4
Training time			
Total Training time	3 hours	2 hours	1 hour

Table 4.2: Training Arguments, LoRA Configuration, and Runtime for Mistral 7B, Phi-2, and Phi-3 on document-level extraction

	Mistral 7B	**Phi-3**	**Phi-2**
LoRA Configuration			
lora alpha	8	8	8
lora dropout	0.1	0.1	0.1
r	8	8	8
target modules	q_proj, k_proj, v_proj, o_proj	o_proj, k_proj, v_proj, dense	o_proj, k_proj, v_proj, dense
Training Arguments			
warmup steps	3	3	3
batch size	1	1	1
gradient accumulation steps	4	4	4
max steps	500	500	500
learning rate	2e-4	2e-4	2e-4
Training time			
Total Training time	1 h 30 min	50 min	35 min

Table 4.3: Training Arguments, LoRA Configuration, and Runtime for Mistral 7B, Phi-2, and Phi-3 on sentence-level extraction

4.6 Evaluation

The evaluation phase constitutes a critical component in the context of fine-tuning an LLM for a specific task. This study employs a rigorous evaluation methodology to assess the performance of our fine-tuned model on the M^2E^2 dataset, with a focus on assessing both trigger and argument extraction. To ensure comparability with existing literature and facilitate benchmarking, we utilize a set of established metrics widely adopted in the field of information extraction.

4.6.1 Metrics

To evaluate our models, it is important to choose metrics that are relevant to our task. In the case of event extraction, precision, recall, and F1-score are the metrics usually chosen. These metrics have a long history in information extraction evaluations, dating back to the *message understanding conferences* (MUC) (Chinchor and Sundheim, 1993).

Precision measures the proportion of correctly identified events among all events detected by the system. Recall measures the proportion of correctly identified events among all actual events in the dataset. The F1-score is the harmonic mean of precision and recall, providing a single balanced metric.

4.6.2 Evaluation on the M^2E^2 dataset

For our specific dataset, we evaluate the trigger and argument extraction separately using precision, recall and F1-score defined above. The evaluation is intended to be an end to end evaluation as in Li et al. (2020), however we do not extract any information on the position of the triggers or arguments in the text.

For both, document and sentence level event extraction, we consider a trigger as correct if its event type and text matches a reference trigger. Similarly, for arguments we consider them correct if the event mention/trigger they are associated with is correct and their argument role and text matches a reference argument.

Chapter 5
Results

This chapter presents the outcomes of this work, detailing the results for both document-level and sentence-level event extraction. Initial experiments with document-level event extraction yielded modest results, leading us to explore alternative strategies, such as fine-tuning models on sentence annotations. A discussion of these results is provided in Chapter 6.

5.1 Document-Level Event Extraction

As explained in Chapter 4, we first fine-tuned models on entire articles for document-level event extraction. We compared base and fine-tuned versions of Phi-2, Phi-3, and Mistral 7B by computing precision, recall, and F1-scores for both trigger and argument extraction across each event type, as well as overall scores. The base models were evaluated with two-shot prompts while the fine-tuned models were evaluated with the same prompt used for the fine-tuning (zero shot).

In Figure 5.1, we present the overall metrics for trigger and argument extraction tasks for the three fine-tuned models. The models are performing consistently across both precision and recall metrics. This is important to make sure that the models aren't generating too many false positives versus missing too many relevant events. The results clearly indicate that the argument extraction task is more difficult than the trigger extraction task for both fine-tuned and base models. The base models achieve really poor results, while the fine-tuned models demonstrated improvements, especially for the trigger extraction task. Overall, Mistral 7B fine-tuned model achieve the highest performance, with 0.329 and 0.089 F1-score on respectively trigger and argument extraction tasks.

Figure 5.2 presents the F1-scores for both trigger and argument extraction of the fine-tuned models. For argument extraction, all three models show relatively similar results. For the trigger extraction task, some event types prove more challenging than others, but no clear trend emerges, even between the three models, and even between Phi models. These results suggest that larger models tend to achieve higher performance.

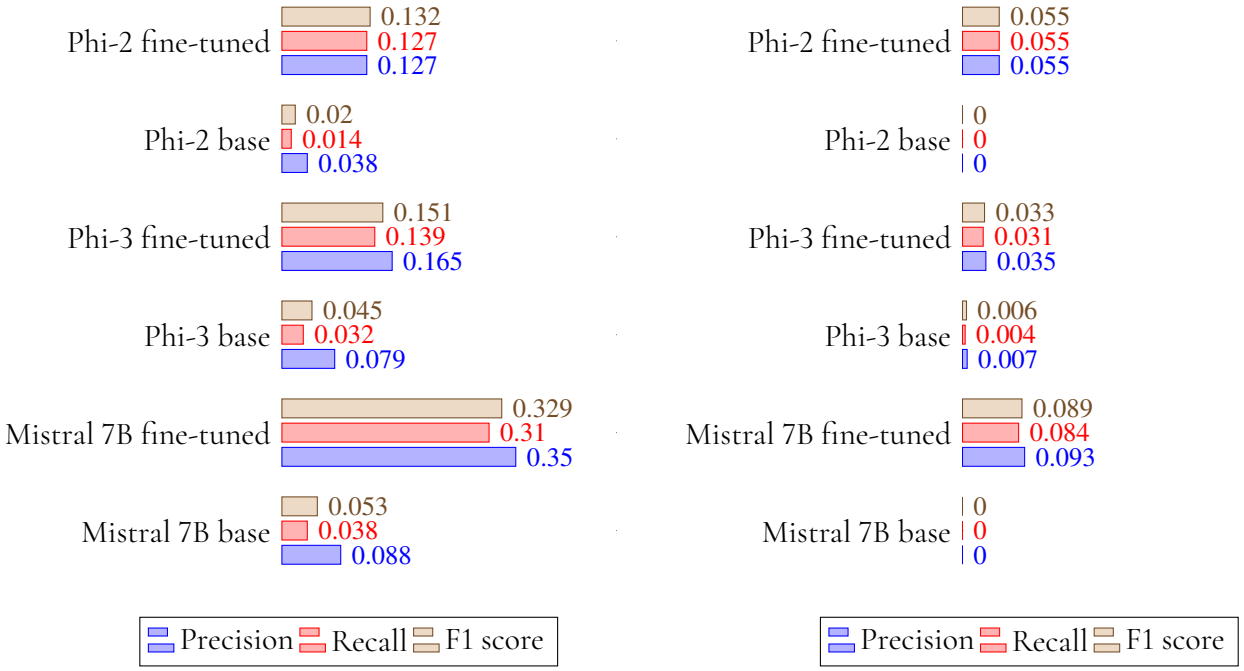


Figure 5.1: Overall metrics for base and fine-tuned models on trigger (left) and argument (right) extraction tasks

Figure 5.3 shows the training and validation lost for all three models. But it's not a good indicator to when the model is overfitting or not since the loss function is not aligned with the F1-score computed on the extracted triggers and arguments.

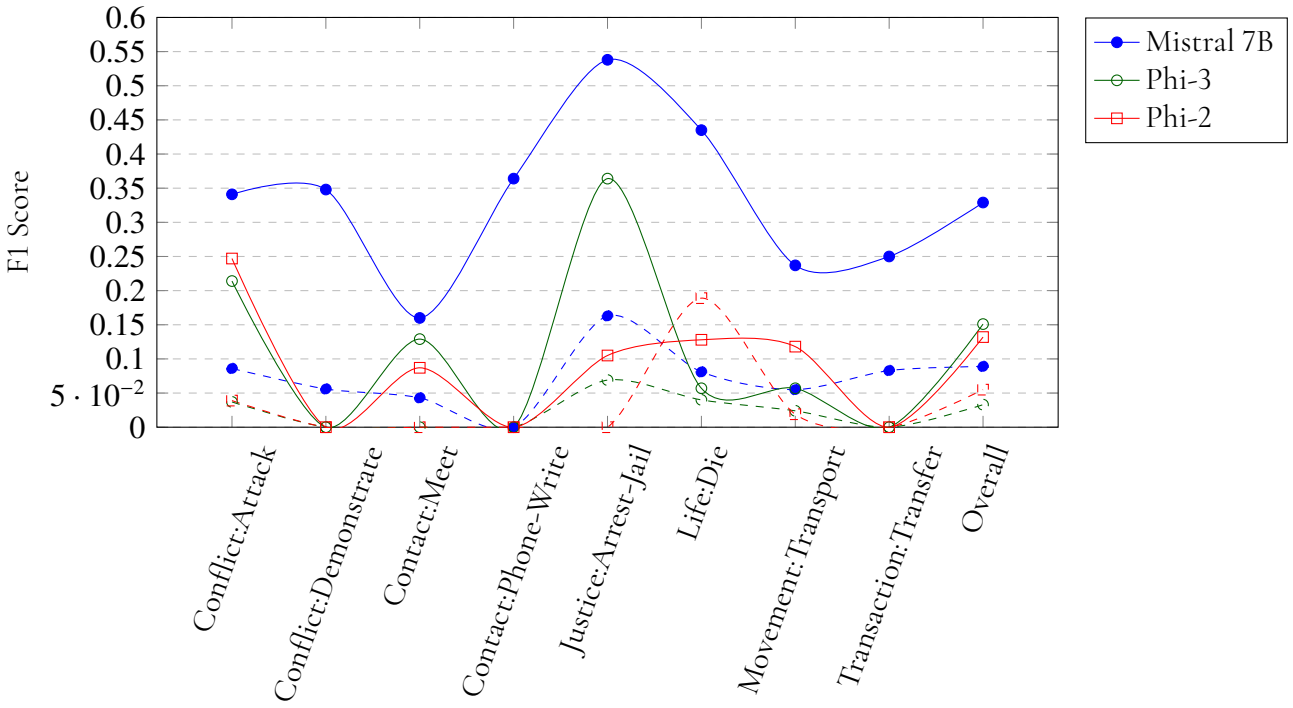


Figure 5.2: F1 Score by class for the trigger (solid lines) and argument (dashed lines) document-level extraction for the three fine-tuned models

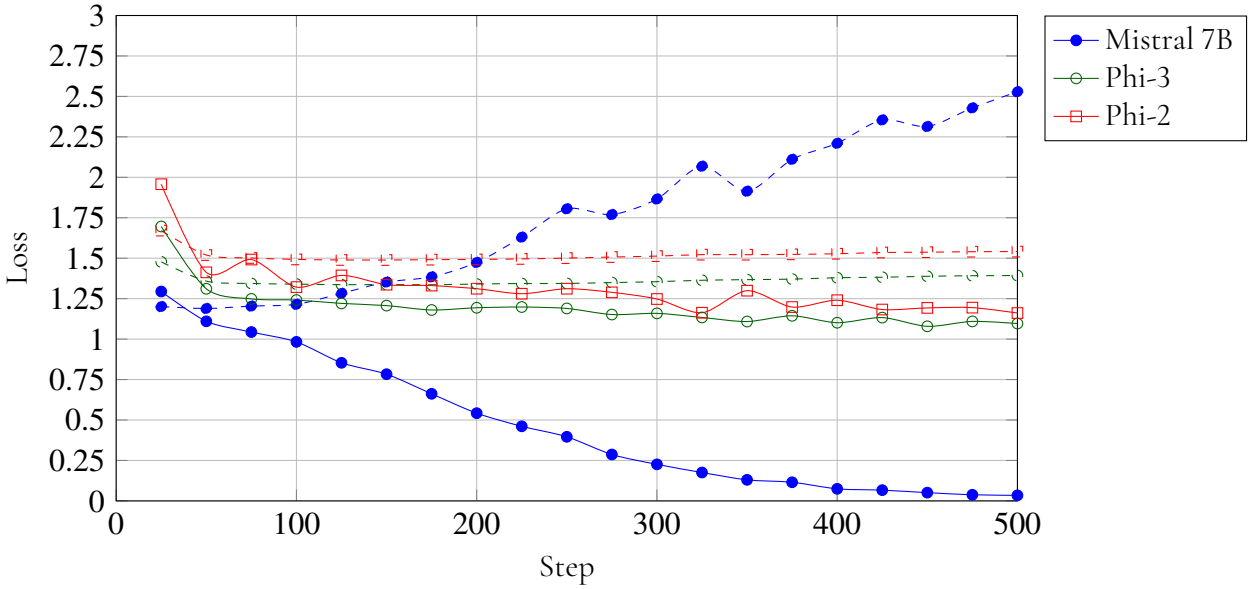


Figure 5.3: Training (solid lines) and Validation Loss (dashed lines) over gradient steps for the three models on document level extraction

5.2 Sentence-Level Event Extraction

After observing modest results from fine-tuning models for document-level event extraction, we aimed to delve deeper into the limitations posed by a small dataset. Consequently, we fine-tuned the models using sentence-level annotations. Similar to our previous approach, we compared the performances of the base models with two-shot prompts and the fine-tuned model with the regular prompt, computing precision, recall, and F1-scores for both trigger and argument extraction tasks.

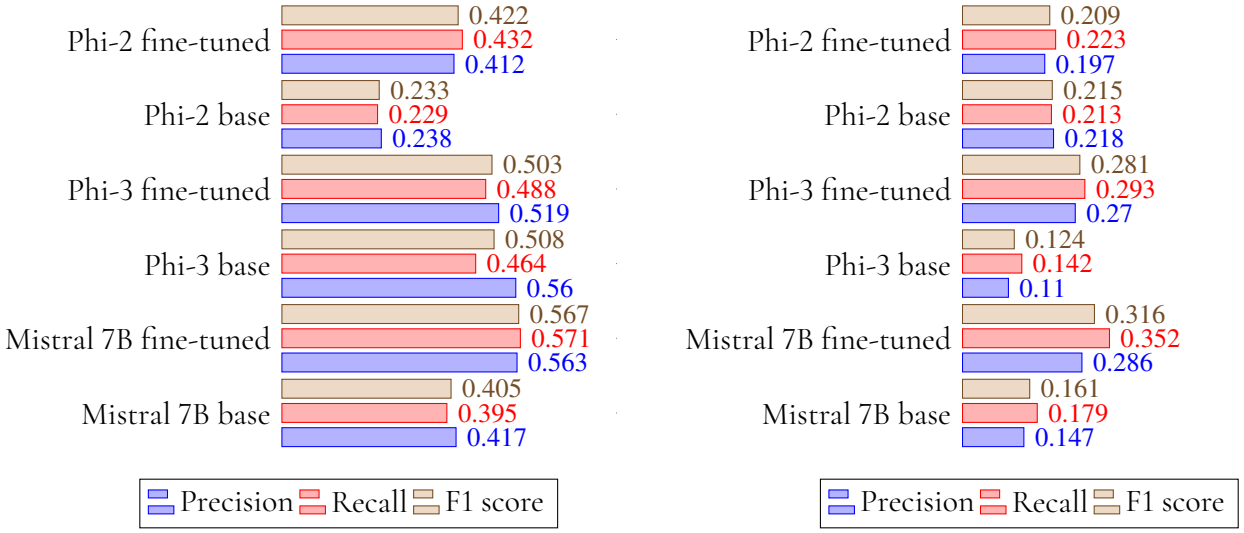


Figure 5.4: Overall metrics for base and fine-tuned models on trigger (left) and argument (right) extraction tasks

Figure 5.4 shows the metrics across all event types comparing the base models with the fine-tuned models. As anticipated, sentence-level event extraction is less complex, and the fine-tuned models achieved higher scores compared to document-level extraction. As for the document level extraction, Mistral 7B fine-tuned model demonstrated best results reaching 0.567 and 0.316 F1 score on trigger and argument extraction tasks respectively. Surprisingly, Phi-3 base model demonstrated better precision and F1-score on the trigger extraction task compared to its fine-tuned counterpart, beating also the two other base models on this task. And Phi-2 base model achieved better results than Phi-3 and Mistral 7B base models on the argument extraction. Overall, taking into account argument and trigger extraction combined, the fine-tuning process improved the results for all three models.

Figure 5.5 presents the F1 scores by class for the fine-tuned models, on both trigger and argument extraction. As with document-level event extraction, no clear trend emerged across all models between the event types. It's also the case for the arguments, suggesting no type of event seem to have arguments that are easier to capture.

Figure 5.6 shows the validation and training losses for the three models. Again it's not the best indicator for our task, but we can see that the models fits better to this loss than for the document level extraction.

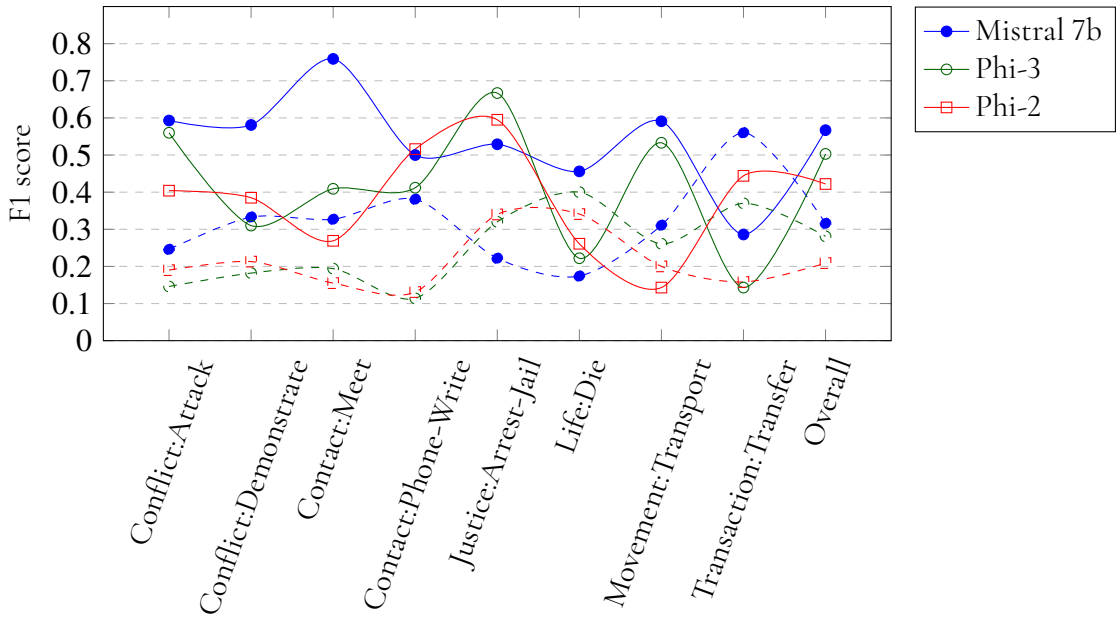


Figure 5.5: F1 score by class for the trigger (solid lines) and argument (dashed lines) sentence-level extraction for the three models

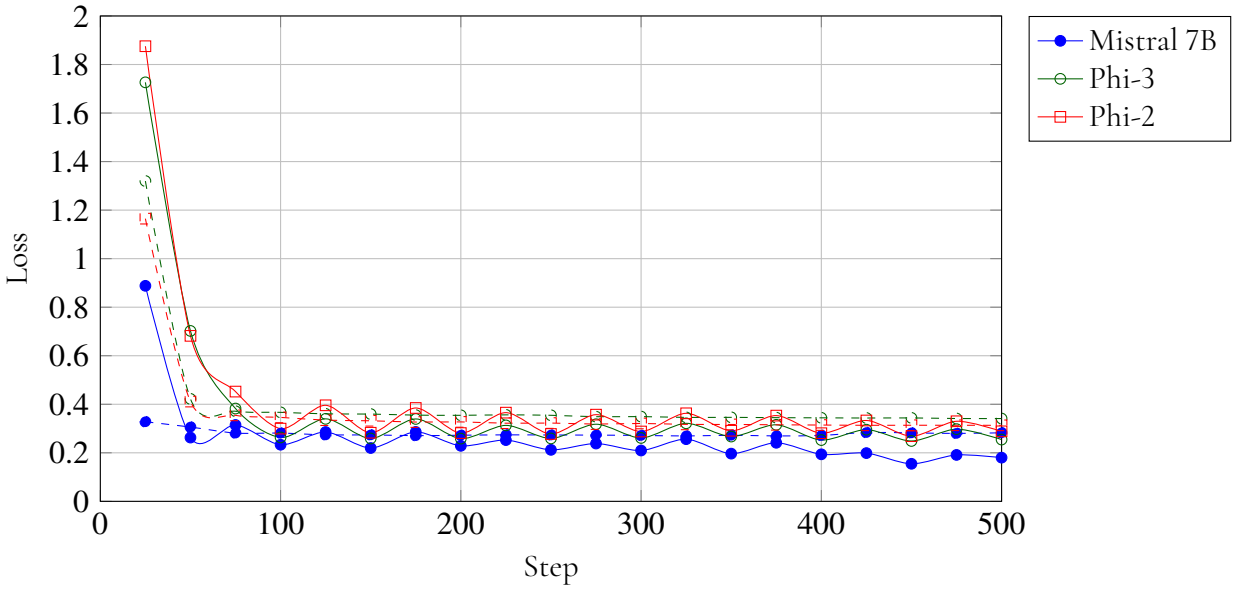


Figure 5.6: Training (solid lines) and Validation Loss (dashed lines) over gradient steps for the three models on document level extraction

Chapter 6
Conclusions

In this chapter, we elaborate on the results from the previous chapter and draw some conclusions to try and answer the research questions initially stated in this Master's thesis.

6.1 Results Discussion and Limitations

The results presented in the previous chapter offer a better understanding of the capabilities of LLMs, specifically Phi-2, Phi-3, and Mistral 7b, after fine-tuning for event extraction tasks. Our research demonstrates that fine-tuning greatly enhances the model's performance, particularly with regard to verbosity, the decrease of hallucinations, and output formatting. Our findings show that even small language models like Phi-2, Phi-3, and Mistral 7b can potentially be used for event extraction with some fine-tuning. Obtaining better results for sentence-level event extraction, but there is still space for improvement.

Regarding the specific tasks covered in this Master's thesis, our results indicate that trigger extraction is less complex than argument extraction. This discrepancy is probably caused by the fact that argument roles vary more widely within a single event type and by the semantic difficulties that come with argument extraction. Arguments were annotated using a dependency parsing tree logic, selecting only the root word. However, LLMs don't possess knowledge related to the semantic of the words. For example, in the sentence:

> Two black men were *shot* (trigger: Conflict: Attack) by white *police* (Arg: Attacker) officers at point-blank range in the states of *Louisiana* (Arg: Place) and Minnesota.

The trigger is quite evident, but the arguments are less straightforward. LLMs would tend to output 'officers' or 'police officers' for the 'Attacker' argument. This global observation seems to be aligned with the results in Li et al. (2020), which reported significantly lower metrics for argument extraction compared to trigger extraction.

Another important consideration for the document level-extraction is that this dataset focuses on intra-sentence event extraction, requiring triggers and arguments to be within the same sentence. We attempted to address this by including target sentences in the annotations, but this approach led to degraded performance. Since the loss function is computed token-by-token by comparing the predicted probability distribution with the actual target token at each position, it likely resulted in the model focusing more on the sentences rather than on accurately extracting triggers and arguments.

Further on, the dataset is likely not big enough for the model to be able to capture all the variations of trigger words and arguments in order to achieve satisfying performances.

6.2 Fine-Tuning Large Language Models

The method of fine-tuning has shown to be a breakthrough in enhancing LLM performance in specific tasks and domains. It expands upon PLMs with a general understanding of language but no task-specificity. Although LLMs have demonstrated some ability to follow instructions, especially when given examples, their outputs often display verbosity and hallucinations. Our results demonstrate that fine-tuning, without requiring a significant amount of data, has the ability to overcome this problem by restricting the model's possible answers to a more accurate and task-specific space.

Smaller language models (SLMs) have significantly increased fine-tuning accessibility. It is now possible to fine-tune SLMs like Phi-2, achieving performances on certain tasks comparable to non-fine-tuned models like GPT-4, using just a single GPU and 16 GB of VRAM. Because it enables companies to host their own models that have been refined with proprietary data, this feature is particularly noteworthy because it eliminates the need for external API endpoints and maintains control over model parameters. This lower cost and improved accessibility of fine-tuning LLMs have opened up chances to address real-world challenges. Companies can now design customized approaches without requiring substantial computing resources or relying on third-party services.

Additionally, SLMs are now being developed by large AI companies. Microsoft leads this movement with its Phi series, which varies in model size, including three variations of the Phi-3 model and Meta has enhanced its Llama series, with Llama 3 8B being their most recent SLM. Google has also entered this space with the development of Gemma 7B. The field of natural language processing has advanced with the rise in SLMs and the faster rate at which they may be fine-tuned for certain applications. LLMs offer a more general approach than typical deep neural network models, which may need extensive modeling and task-specific architectures. This feature makes it easier for users to apply these models to a variety of applications.

6.3 Considerations

As artificial intelligence advances, particularly in the area of generative AI, it is important to closely monitor the dangers and misuses associated with these technologies. One major reason to be concerned is the amount of energy and computing power required to train and execute LLMs. These models also raise significant ethical concerns since the information they

generate is prone to prejudice, stereotypes, and the development of bad habits.

6.3.1 Resource Consumption and Environmental Impact

The process of training LLMs requires a vast amount of resources. Pre-training models like GPT-4, for example, require a significant amount of computational ressources. Development expenses are estimated to be between $2 and $3 million USD. In addition to being expensive, this process has a significant carbon footprint, which raises environmental issues. Even the later fine-tuning process requires a substantial amount of GPU resources, although being less demanding than pre-training.

Despite the fact that optimization techniques like the ones employed in this work lower these resource requirements, using LLMs is still a costly and resource-intensive undertaking. Therefore, when using these models, it's important to apply best practices designed to prevent resource waste and to consider the associated costs and environmental effects while employing these models.

6.3.2 Ethical Implications

In addition to environmental issues, LLMs bring up ethical concerns. Because these models produce information that is based on the training data which can include preconceptions or damaging narratives, they have the potential to produce content that reinforces social biases. While LLMs offer new automated solutions, their lack of accountability, transparency, and human labor displacement pose risks.

Given these considerations, it is essential to evaluate the application of LLMs critically, particularly in circumstances where ethical integrity is important. Furthermore, lighter models may sometimes produce results that are easier to interpret and have adequate performance. As the field evolves, it is important to find a balance between harnessing the incredible potential of LLMs and fostering ethical, sustainable, responsible and explainable AI practices.

6.4 Conclusions

The goal of this project was to benchmark different large language models on an event extraction dataset. We wanted to explore Phi-2, Phi-3, and Mistral 7B, considering these are some of the most recent models and provided a good size split, ranging from 2.7 billion to 7 billion parameters. We compare the fine tuning process on each of these models to the base models with 2-shot prompts on both sentence-level and document-level event extraction. We demonstrate that the fine-tuning process improves the results for every model on both tasks, especially on the document-level event extraction where the base models perform very poorly. The fine-tuning process made the models less prone to hallucinations and verbosity and also helped in ensuring a stable format of output.

On the document-level, the performances are modest but show some potential, suggesting a bigger dataset would help the models to learn better representations, especially for the

argument extraction. For the sentence level, while the base models show a superior comprehension of the problem at the phrase level compared to document-level, the results were improved through fine-tuning, especially for the argument extraction part. We noticed that the Phi-3 and Mistral 7B base models, for which we utilized the instruct versions, demonstrated better performances compared to Phi-2, which didn't undergo an instruction phase. As a result, the instruct models show less of an improvement after being fine-tuned in sentence-level event extraction.

This Master's thesis also provided the chance to investigate optimization techniques like LoRA and QLoRA, which lessen the need for resources and the expense of training these kinds of models. In order to make generative AI-based solutions profitable, this is an essential component for companies looking to develop such solutions. For Phi-2, a single NVIDIA T4 GPU can be used thanks to these optimization techniques, which are relatively cheap and can be accessed for free with Kaggle or Google Colab.

With these results in mind we will go over the research questions initially defined for this project.

1. **How effectively do large language models perform when fine-tuned for event extraction tasks?**
   All three models show improvement after being fine-tuned on both sentence-level and document-level extraction. The argument extraction task shows the most improvement, as it is the most challenging task at hand. On sentence-level extraction, the models demonstrate promising results, while for document-level extraction, the results are modest but still show improvement after fine-tuning.

2. **To what extent does the size of language models influence their performance in event extraction?**
   We will address this question with our empirical results, as it heavily depends on the models employed. Phi-3 shows clear improvement compared to its predecessor Phi-2, with both the base and fine-tuned models demonstrating better performance. Since these two models share a very similar architecture, we can confidently hypothesize that model size plays a role. However, it is important to consider the additional instruction phase Phi-3 received, as well as its improved training dataset. Mistral 7B overall demonstrated superior performance, suggesting that model size impacts results to some extent.

3. **What is the effect of dataset constraints on the performance of fine-tuned models in event extraction?**
   The models definitely benefited from the fine-tuning process; however, the results are quite modest especially for document-level extraction. The base models performances as well as the fine-tuned models results suggests that the task is quite challenging, especially the argument extraction and that the models have the potential to perform better with a bigger dataset.

6.5 Future Work

To extend our work, we propose several ideas to explore. Firstly, our study might be expanded by adding data from the ACE dataset (Walker et al., 2005), which would enrich the data with

more event types and a larger dataset. Since we used three model SLMs in this thesis, it could be interesting to include more SLMs, such as Phi-3 small (Microsoft, 2024), Gemma 7B (Google, 2024), or Llama 3 8B (Meta AI, 2024), in this benchmark.

Adding a first extraction step to extract event mentions and then using the fine-tuned models to do sentence event extraction could be one future strategy to improve the outcomes. This step could probably be done with a simple encoder model with sufficient data or try zero shot classification models that have recently improved a lot, especially for classification tasks (Hugging Face, 2024d).

Finally, this Master's thesis is applied to an event extraction dataset, hence this methodology can be applied to other event extraction datasets. Notably, as LLMs make use of the self attention mechanism, we hypothesize that they could perform well on cross-document events.

References

Ahn, D. (2006). The stages of event extraction. In *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*.

Bera, D., Pratap, R., and Verma, B. D. (2021). Dimensionality reduction for categorical data. *arXiv preprint arXiv:2112.00362*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*.

Chinchor, N. and Sundheim, B. (1993). Muc-5 evaluation metrics. In *Proceedings of the 5th conference on Message understanding*.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Google (2024). Gemma: Google introduces new state-of-the-art open models. `https://blog.google/technology/developers/gemma-open-models/`.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR.

Hsu, I.-H., Huang, K.-W., Boschee, E., Miller, S., Natarajan, P., Chang, K.-W., and Peng, N. (2022). Zero and few-shot event extraction via large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*.

Hugging Face (2024a). Peft: Parameter-efficient fine-tuning. `https://huggingface.co/docs/peft/en/index`.

Hugging Face (2024b). Quantization - accelerate. `https://huggingface.co/docs/accelerate/usage_guides/quantization`.

Hugging Face (2024c). Transformers. `https://huggingface.co/docs/transformers/index`.

Hugging Face (2024d). What is zero-shot classification? `https://huggingface.co/tasks/zero-shot-classification`.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de Las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Le Scao, T., Lavril, T., Wang, T., Lacroix, T., and El Sayed, W. (2023). Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Li, M., Zareian, A., Zeng, Q., Whitehead, S., Lu, D., Ji, H., and Chang, S.-F. (2020). Cross-media structured common space for multimedia event extraction. In *Proceedings of The 58th Annual Meeting of the Association for Computational Linguistics*.

Li, Q., Li, J., Sheng, J., Cui, S., Wu, J., Hei, Y., Peng, H., Guo, S., Wang, L., Beheshti, A., and Yu, P. S. (2022). A survey on deep learning event extraction: Approaches and applications. *arXiv preprint arXiv:2107.02126*.

Liao, B., Meng, Y., and Monz, C. (2023). Parameter-efficient fine-tuning without introducing new latency. *arXiv preprint arXiv:2305.16742*.

Liu, J., Min, L., and Huang, X. (2021). An overview of event extraction and its applications. *arXiv preprint arXiv:2111.03212*.

Liu, X., Luo, Z., and Huang, H. (2018). Jointly multiple events extraction via attention-based graph information aggregation. *ACL anthology*.

Lu, W., Jain, N., Thayaparan, M., Suresh, V., and Surdeanu, M. (2021). Text2event: Controllable sequence-to-structure generation for end-to-end event extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*.

Meta AI (2024). Introducing meta llama 3: The most capable openly available llm to date. `https://ai.meta.com/blog/meta-llama-3/`.

Microsoft (2024). The phi-3 small language models with big potential. `https://news.microsoft.com/source/features/ai/the-phi-3-small-language-models-with-big-potential/`.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Narang, S., Diamos, G., Elsen, E., Micikevicius, P., Alben, J., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. (2018). Mixed precision training. In *International Conference on Learning Representations*.

Nguyen, T. H., Cho, K., and Grishman, R. (2016). Joint event extraction via recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365v2*.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. Technical report, OpenAI.

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., and Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.

Singh, A., Pandey, N., Shirgaonkar, A., Manoj, P., and Aski, V. (2024). A study of optimizations for fine-tuning large language models. *arXiv preprint arXiv:2406.02290*.

Singh, S. (2018). Natural language processing for information extraction. *arXiv preprint arXiv:1807.02383*.

Tunstall, L., von Werra, L., and Wolf, T. (2022). *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, Inc.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Walker, C., Strassel, S., Medero, J., and Maeda, K. (2005). Ace 2005 multilingual training corpus. *Linguistic Data Consortium*.

Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. (2022a). Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2022b). Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Yang, S., Feng, D., Qiao, L., Kan, Z., and Li, D. (2019). Exploring pre-trained language models for event extraction and generation. *ACL anthology*.

Zhang, S., Dong, L., Li, X., Zhang, S., Sun, X., Wang, S., Li, J., Hu, R., Zhang, T., Wu, F., and Wang, G. (2024). Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792v5*.

Appendices

Appendix A
Abbreviations

ACE	Automatic Content Extraction
AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CR	Coreference Resolution
EBS	Elastic Block Store
EE	Event Extraction
FFN	Feed Forward Network
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
IE	Information Extraction
LLM	Large Language Models
LoRA	Low-Rank Adaptation
LSTM	Long Short-Term Memory
ML	Machine Learning

MLM Masked Language Modeling

MLP Multi Layer Perceptron

NEL Named Entity Linking

NER Named Entity Recognition

NVMe Non-Volatile Memory Express

NLP Natural Language Processing

PEFT Parameter-Efficient Fine-Tuning

PML Pre-Trained Model

POST Parts-of-Speech Tagging

QLoRA Quantized Low-Rank Adaptation

RNN Recurrent Neural Network

RE Relation Extraction

ReLU Rectified Linear Unit

SFTTrainer Supervised Fine-Tuning Trainer

SLM Small Language Model

SOTA State of the Art

SSD Solid State Drive

TRL Transformer Reinforcement Learning

VRAM Video Random Access Memory

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-08-29

EXAMENSARBETE Leveraging Large Language Models for Event Extraction
STUDENT Jonathan Desnoyer
HANDLEDARE Pierre Nugues (LTH)
EXAMINATOR Jacek Malec (LTH)

Empowering Event Extraction Through Next-Gen Language Models

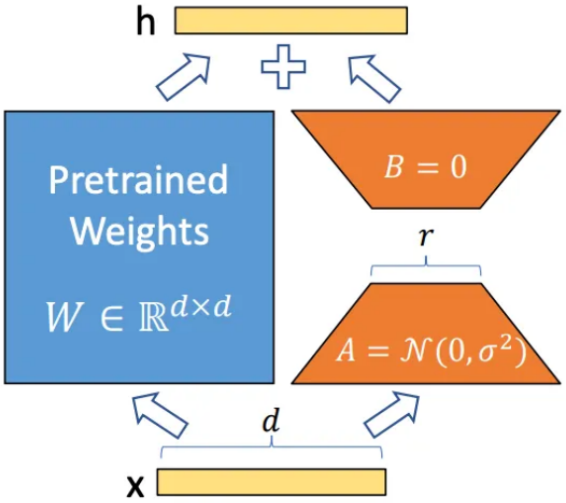
POPULÄRVETENSKAPLIG SAMMANFATTNING **Jonathan Desnoyer**

In this thesis, we fine-tune three large language models, Mistral 7B, Phi-2 and Phi-3 on an event extraction dataset and compare their performance to the base models using two-shot prompts.

Event extraction is a task that aims to identify and extract event triggers and their associated arguments from text. Most recent approaches have focused on deep neural networks and the transformer architecture. However, the emergence of large language models (LLMs) and innovative optimization techniques like low-rank adaptation (LoRA) present new opportunities for advancing event extraction.

With the recent advancements in generative AI, the fine-tuning process has appeared as a way to adapt LLMs on specific tasks. Rather than retraining the entire model, parameter-efficient fine-tuning (PEFT) methods have been developed to be more resource efficient. These methods typically keep most of the model layers frozen and modify a subset of layers. Nowadays, LoRA is the most widely used approach, which approximates weight matrices with low-rank matrices and only alters a subset of the model's parameters.

We have fine-tuned three different LLMs on an event extraction dataset that consist of new articles with intra-sentence events. We use LoRA and compare the results with the base models. The three models have sizes varying from 2.7 billion parameters to 7 billion to evaluate the impact of size on the performances. The models were fine-tuned and evaluated on both document and sentence-level to get a better understanding of the capabilities of these new large language models.



We show that fine-tuning these models enhances the results on both document and sentence-level event extraction, with Mistral 7B outperforming Phi-2 and Phi-3 on both levels. Document-level extraction being a challenging task, reaching 0.329 and 0.089 F1-score against 0.576 and 0.316 for sentence-level extraction on trigger and argument extraction, respectively.