

Bachelor's Thesis

Title of Bachelor's Thesis (English)	
Title of Bachelor's Thesis (German)	
Author (last name, first name):	
Student ID number:	
Degree program:	
Examiner (degree, first name, last name):	

I hereby declare that:

1. I have written this Bachelor's thesis myself, independently and without the aid of unfair or unauthorized resources. Whenever content has been taken directly or indirectly from other sources, this has been indicated and the source referenced.
2. This Bachelor's Thesis has not been previously presented as an examination paper in this or any other form in Austria or abroad.
3. This Bachelor's Thesis is identical with the thesis assessed by the examiner.
4. (Only applicable if the thesis was written by more than one author): this Bachelor's thesis was written together with

The individual contributions of each writer as well as the co-written passages have been indicated.

Date



Signature

Bachelor Thesis

Evaluating the Impact of Task Adaptive Pre-training (TAPT) in Financial Context: A Comparison of BERT with and without Task Adaptive Pre-training for Named Entity Recognition on the FiNER-139 Dataset

Tobias Hundsberger

Date of Birth: 30.01.2001

Student ID: 12042646

Subject Area: Information Business

Studienkennzahl: UJ033561

Supervisor: ao.Univ.Prof. Dr., Johann, Mitlöhner

Date of Submission: 03.September 2025

Department of Information Systems & Operations Management, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

Contents

1	Introduction	7
2	Related Work	8
3	The BERT model	9
3.1	Architecture	9
3.1.1	Tokenization	10
3.1.2	Embedding Layer	11
3.1.3	Transformer Layers & Multi-Head Attention	12
3.1.4	Model Output: Contextual Token-Embeddings	14
3.1.5	Output Head	15
3.1.6	The CLS-Token	15
3.2	Training Stages of BERT	16
3.2.1	Unlabeled Pre-Training	16
3.2.2	Labeled Fine-Tuning	16
4	Named Entity Recognition and the FiNER-139 Dataset	17
4.1	Named Entity Recognition Overview	17
4.1.1	Introducing Named Entity Recognition	17
4.1.2	The IO2 Tagging Systems	18
4.1.3	NER Evaluation	18
4.2	Dataset Structure and Tags	20
5	Pre-training BERT	21
5.1	Initial Pre-Training	21
5.2	Domain Adaptive Pre-training (DAPT)	23
5.3	Task Adaptive Pre-training (TAPT)	24
6	Fine-tuning BERT	24
6.1	Downstream Tasks	25
6.2	Output-Head	25
6.3	Labeled Training Process	26
6.4	NER-Specific Fine-Tuning	27
7	Methodology: Code Implementation of the Training Pipeline	28
7.1	Frameworks and Tools	29
7.2	The Model: bert-base-uncased	30
7.3	Dataset Preparation	30
7.3.1	The FiNER 139 Dataset	30
7.3.2	Label Reduction	31

7.3.3 Text Extraction for TAPT	32
7.4 Devices and Environments	32
7.5 TAPT Implementation	32
7.6 Labeled Fine-Tuning Implementation	35
7.7 Metrics	36
8 Results	37
8.1 Overall Performance	37
8.2 TAPT vs. Classic Fine-tuning	38
8.3 Comparison and Interpretation	39
9 Conclusion	39
9.1 Future Work	40
A Appendix: Code	41

List of Figures

1	High-level schematic diagram of BERT [7]	10
2	BERT's three types of embeddings for input representation [7]	12
3	Encoder-only attention [11]	14
4	Overall pre-training procedures for BERT. [7]	23
5	Overview of the training setup. Own illustration.	29

List of Tables

1	Different Pretraining Phases	24
2	Token length distribution obtained through own dataset analysis in the 00b_FiNER139-Text-Extraction-Analysis.ipynb Jupyter Notebook available in the GitHub Repository	33
3	Evaluation metrics for Base and TAPT models	37
4	Precision, Recall, and F1 scores per entity type for Base and TAPT models	38

Abstract

The extremely high computational costs associated with pre-training language models on large general text corpora constitute a significant barrier to training large language models, particularly given that computational requirements increase exponentially with linear model growth, in an era where ever-larger models are being pursued. A possible solution lies in finding a compromise in continued pre-training. Instead of continued Pre-Training (Domain Adaptive Pre-training (DAPT)) on huge domain dataset which require huge computational resources, the base model only receives training on the specific task data via Masked Language Modeling (MLM) by removing labels from the dataset. This process is known as Task-Adaptive Pre-training (TAPT). After this pre-training step, supervised fine-tuning for the downstream task (Named Entity Recognition, NER) can be carried out by setting up and training a linear output layer on top of the pre-trained BERT body. Due to computational constraints, this work does not include a direct comparison between TAPT and DAPT. Instead TAPT, as a far more resource efficient continued pre-training than DAPT, will be compared to pure supervised fine-tuning in order to evaluate whether Task Adaptive Pre-training (TAPT) improves performance in financial NER tasks compared to standard fine-tuning.

1 Introduction

"Can machines think? [1]" This fundamental question was posed by Alan Turing as early as 1950 in his seminal essay "Computing Machinery and Intelligence." In that work, he also introduced the Turing Test, a benchmark that still plays a role in discussions about artificial intelligence (AI) today. This highlights the long-standing history of the AI field, which many people are unaware of. Most only became interested in AI during the recent wave of attention sparked by the release of OpenAI's ChatGPT at the end of 2022. However, ideas and attempts to create what we now call artificial intelligence began long before that. Early approaches aimed to mimic human thinking through logical, rule-based systems. These systems are now known as symbolic AI. While effective for simple tasks, symbolic AI struggled with more complex problems because it lacked the ability to generalize. In the 1990s, a shift occurred toward data-driven and statistical methods, which became known as machine learning. These new approaches benefited greatly from the rise of the internet, which provided large amounts of data, as well as from improvements in computational power. This combination laid the groundwork for the development of deep learning, with neural networks as its core technology. This progress enabled the creation of more sophisticated model architectures such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks. Both architectures already showed strong performance in natural language processing (NLP) tasks. However, they had a major drawback: their recurrent nature. Each word had to be processed in sequence, which made training slow and limited their ability to handle long text contexts. A new type of architecture was needed to overcome these issues. The 2017 publication of "Attention is All You Need" by Vaswani *et al.* [2] marked a turning point. This paper introduced the Transformer, an architecture that replaced recurrence with a self-attention mechanism. This change made it possible to process text in parallel and better understand longer contexts, greatly improving the generation and comprehension of natural language. The Transformer became the foundation for models like BERT and GPT, which learned language by being pre-trained on large collections of text. These models quickly became state-of-the-art in many NLP tasks, including applications in the financial domain. However, even these powerful models face challenges. Since their architecture allows them to be trained on massive datasets, they also require significant computational resources, especially during the pre-training stage. As the demand for customized solutions grows, there is increasing interest in more efficient fine-tuning methods. One such approach involves avoiding general domain pre-training (known as Domain-Adaptive Pre-training, or DAPT) and in-

stead focusing on Task-Adaptive Pre-training (TAPT). TAPT uses smaller but more targeted datasets to adapt a model to a specific task, which could reduce computational needs while maintaining or even improving performance. This leads to the central research question of this work: Can Task-Adaptive Pre-training (TAPT) improve performance in financial Named Entity Recognition (NER) compared to direct fine-tuning of a base model, without relying on costly Domain-Adaptive Pre-training? To answer this, the study focuses on the performance of TAPT on the FiNER-139 dataset, a benchmark for financial NER. Only TAPT will be evaluated, avoiding the high computational cost of DAPT. The goal is to determine whether TAPT alone can provide a meaningful performance boost over using a base model with standard fine-tuning.

2 Related Work

The BERT transformer model uses an unsupervised learning technique called Masked Language Modeling (MLM) during pre-training to adapt its internal model weights for general text comprehension. To improve performance in specialized domains, additional pre-training on domain-specific text corpora has been incorporated into the training process. This approach is known as Domain-Adaptive Pre-Training (DAPT). Another, more task-focused method is called Task-Adaptive Pre-Training (TAPT), which uses the unlabeled version of the fine-tuning data. By removing the labels, the data becomes suitable for the unsupervised nature of MLM. TAPT is more resource-efficient and more specialized for the target task. Both DAPT and TAPT have been studied and shown to be effective, especially in the work of Gururangan *et al.* [3]. In this work, TAPT is explored further in the financial domain for Named Entity Recognition (NER), using an adapted version of the FiNER-139 dataset.

Gururangan *et al.* [3]: Don’t Stop Pre-training investigated both DAPT and TAPT across various combinations and domains. Their experiments focused on classification tasks and demonstrated that the combination of DAPT and TAPT yielded the best performance. However, DAPT alone or TAPT alone also led to significant improvements compared to the base pre-trained BERT model. TAPT, in particular, stood out as a cost-efficient and effective standalone method. These findings lay the groundwork for further exploration of TAPT in new domain-specific tasks such as financial NER.

Yang *et al.* [4]: FinBERT applied DAPT on top of the BERT model using financial texts like SEC filings, earnings calls, and analyst reports. The resulting FinBERT model showed significantly better performance on

financial sentiment analysis tasks. Although the domain is similar to the one in this work, TAPT was not explored in FinBERT, leaving room for further investigation within financial applications.

Peng *et al.* [5]: "Is Domain Adaptation Worth Your Investment?" compared DAPT on top of the base BERT model with training a BERT model from scratch using domain-specific data. Their findings showed that pre-training from scratch did not yield significantly better performance and required much more computational resources. Therefore, continued pre-training in the form of DAPT was confirmed to be more practical and effective. Despite existing work on DAPT and TAPT, there remains substantial potential for further research on TAPT in specific domains such as finance. This work aims to address the research gap by directly evaluating TAPT in the financial domain on the downstream task of Named Entity Recognition. The experiment involves fine-tuning BERT with TAPT only (excluding DAPT), using an adapted version of the FiNER-139 dataset, and comparing the results with standard fine-tuning without either TAPT or DAPT.

3 The BERT model

In this chapter, the BERT-base variant of the Bidirectional Encoder Representation from Transformers (BERT) model will be thoroughly analyzed and explained. Since BERT forms the core of this work, a solid understanding of its functionality and architecture is essential for following the line of reasoning throughout this thesis. In particular, Chapter 5: Pre-training BERT with TAPT and Chapter 6: Fine-tuning BERT for NER will build upon Section 3.2 of this chapter in greater depth, preparing the groundwork for the Coding Methodology presented later.

This chapter is structured to begin with input preprocessing, followed by a detailed explanation of BERT's internal components, and concluding with an overview of the model's output format. Finally, the model's head, which utilizes the output of the body to perform the downstream task of Named Entity Recognition (NER), will be explained in detail.

3.1 Architecture

The BERT model is structured in multiple layers. The original input to the model is standard user-provided text. This raw text must first be processed into manageable units for the model, a step known as tokenization. Through tokenization, the text is broken down into smaller components called tokens. Based on these tokens, vector representations are created, mapping the tex-

tual information into numerical values that machine learning algorithms can process.

This transformation, from tokens to embeddings represented by their embedding IDs, and their subsequent input into the BERT body is illustrated in the figure below. This process takes place within the embeddings layer. These vectorized representations are then passed through the core of the BERT model: the Transformer layers, which utilize the Multi-Head Attention mechanism introduced by Vaswani *et al.* [2]. In the case of the original BERT architecture, this consists of 12 bidirectional Transformer encoder layers, as described in Devlin *et al.* [6].

The output of these Transformer layers is a sequence of vectors that mirrors the structure of the input but now contains contextualized representations for each individual token. These enriched token representations are visualized in the figure below. They serve as the foundation for diverse downstream tasks, such as Named Entity Recognition (NER).

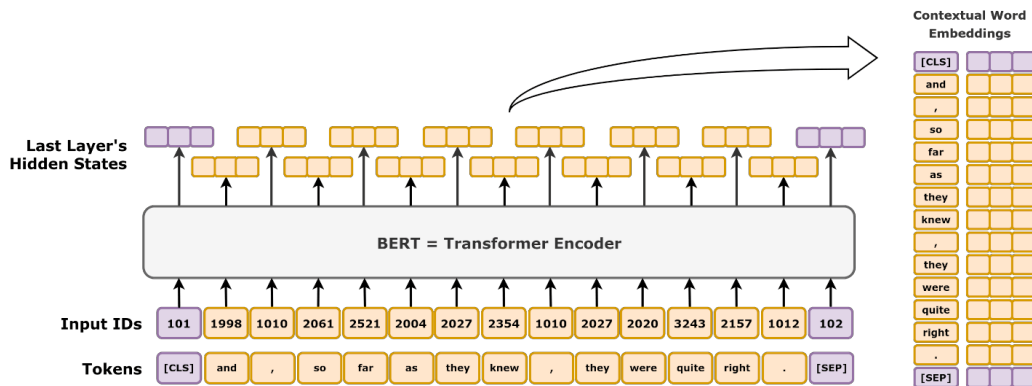


Figure 1: High-level schematic diagram of BERT [7]

3.1.1 Tokenization

For every tokenizer, there are two phases: a training phase and an inference phase. During the training phase, the tokenizer's algorithm is applied to a fixed text corpus. Based on this corpus, the tokenizer generates its own set of tokens, resulting in a fixed vocabulary that is capable of covering the entire corpus.

The BERT model uses its own tokenizer, known as the WordPiece tokenizer. For this segmentation process, the WordPiece tokenizer employs, like many other tokenizers, greedy algorithms. "As a result, it merges subwords into individual tokens if such sequences occur with high relative frequency [8]." For a proper understanding of this process, it is important to note that

each word in the corpus is initially broken down into individual letters and characters, which are also referred to as subwords. BERT's WordPiece tokenizer ultimately produces a token vocabulary of 30,522 tokens based on the model's pre-training data.

Once the tokenizer has been trained, it can be used with any BERT variant without the need for retraining, as it is not part of BERT itself but merely utilized by it.

At this point, only the vocabulary is fixed. The text itself, as well as any future input to the model, still needs to be tokenized based on this vocabulary. During the inference phase, "WordPiece used a greedy longest-match-first strategy: iteratively pick the longest prefix of the remaining text that matches a vocabulary token [9]." This means that each word is tokenized from left to right, matching the largest possible fitting token from the vocabulary. The remaining portion of the word is then again matched against the vocabulary until the entire text has been transformed into tokens. In the rare case where no matching token exists within the vocabulary, the special [UNK] token is used instead.

3.1.2 Embedding Layer

Based on the tokens, every single token receives its own input embedding representation, including special tokens such as the [UNK] token. As already mentioned, an embedding is a numerical vector representation of a literal token. BERT uses 768-dimensional embeddings, meaning each token is represented by a vector of 768 floating-point values.

"For a given token, its input representation is constructed by summing the corresponding token, segment and position embeddings [6]", also illustrated in the figure below. These three types of embeddings are all pre-defined in separate embedding matrices located in the first layer of the BERT model, the embedding layer.

The first type is the token embedding, which provide a unique embedding for each token in the tokenizer's vocabulary of 30,522 tokens. Each embedding consists of 768 values, resulting in a 30,522 x 768 lookup table.

The second type is the position embedding, which encode positional information for each token within a sequence. Unlike sequential models such as LSTMs, transformers have no inherent sense of token order. The sequence length for the BERT model is fixed at 512, meaning that each input sequence can contain up to 512 tokens. If fewer than 512 tokens are present, the [PAD] token is used to fill the sequence to maintain the required format. Sequences are fundamental units within BERT, as they are always processed as a whole, which will become clearer in later sections. The position embed-

dings are stored in a 512 x 768 lookup table.

The third type is the segment embedding, also referred as token type embedding, which distinguish between two consecutive segments, referred to as segments A and B, to help the model learn sentence order during training. For this purpose, only two segment identifiers are needed, resulting in a lookup table of size 2 x 768.

Thus, every token in a sequence is assigned values from all three embedding matrices. These three vectors are summed to produce the final input embedding that is passed to the BERT body.

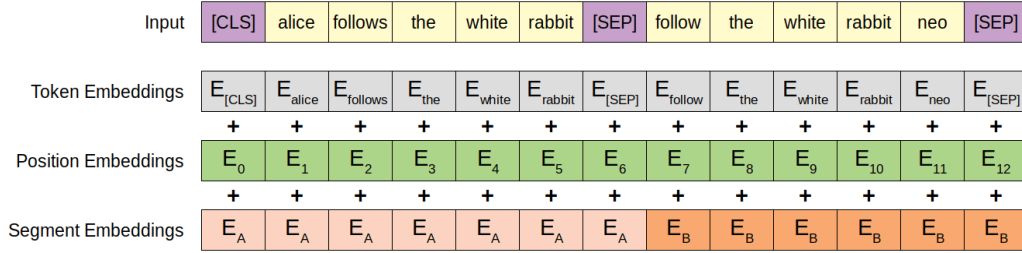


Figure 2: BERT’s three types of embeddings for input representation [7]

All three embedding matrices are initialized randomly at the beginning of BERT’s pre-training. At this initial stage, the embeddings carry no inherent meaning. However, since these matrices are part of the model’s parameters, as shown in the code snippet below, they are updated during the training process. Through continuous updates, these embeddings become increasingly refined and provide better initial representations for the tokens before further contextualization occurs within the Transformer layers.

These embedding parameters can also be examined within BERT’s code-base, which will be referenced in the experimental part of this work.

Trainable Parameter within the BERT-Model:

```
bert.embeddings.word_embeddings.weight      | shape: (30522, 768)
bert.embeddings.position_embeddings.weight  | shape: (512, 768)
bert.embeddings.token_type_embeddings.weight | shape: (2, 768)
```

3.1.3 Transformer Layers & Multi-Head Attention

The Multi-Head Attention mechanism forms the core of all Transformer models and has enabled the development of large language models through its capacity for massive parallel processing. In this section, this algorithm will be explained in the specific context of its implementation within BERT.

At the heart of the attention mechanism are three different matrices: Key (K), Query (Q), and Value (V). Each of these matrices has the dimensions 768 x 768 and is randomly initialized at the beginning of training. Similar to the embeddings layer, the meaningful representations within these matrices are learned and refined through repeated training iterations.

Each of these matrices is split into 12 separate attention heads, each with dimensions of 768 x 64. This division is what is referred to as Multi-Head Attention. The processing within each of these heads follows the so called Scaled Dot-Product Attention mechanism. As Vaswani *et al.* [2] describe, "Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions."

By splitting into multiple heads, the Transformer model gains the flexibility to detect a variety of features through the backpropagation process, effectively encoding these learned characteristics into the model's parameters. Research by Clark *et al.* [10] demonstrated that individual attention heads can specialize in capturing specific linguistic phenomena, such as coreference resolution, syntactic dependencies, or the relationship between prepositions and their objects, without explicit supervision for these tasks. "We find that particular heads correspond remarkably well to particular relations [10]."

As an intermediate result of the scaled dot-product attention mechanism, the following formula is applied to compute the attention scores:

$$\text{Attention Scores} = \frac{QK^T}{\sqrt{d_k}} \in \mathbb{R}^{512 \times 512}$$

This computation results in an attention matrix of dimensions 512 x 512 (as illustrated below), obtained by dividing the dot product of the query (Q) and the transposed key K^T matrices by the square root of the dimensionality of the key vectors, which is $\sqrt{64}$. Each entry in this matrix represents an attention score, indicating its relationship to every other token in the sequence.

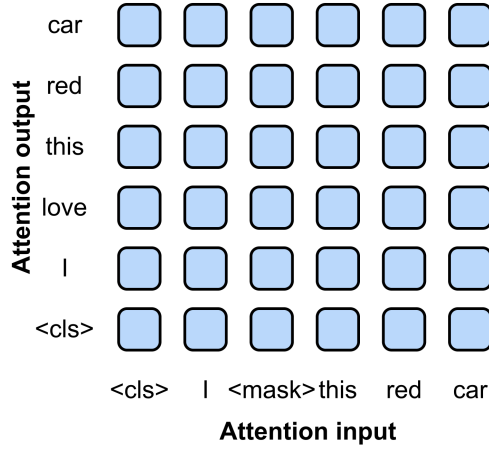


Figure 3: Encoder-only attention [11]

Each of the 12 attention heads described above produces such an attention matrix of size 512 x 512, with each head capturing different aspects of contextual relationships within the input sequence. In the subsequent transformation step, these attention matrices are applied to the corresponding value (V) matrices. The resulting outputs from all 12 heads are then concatenated and merged, forming a single output matrix of size 512 x 768. While this output retains the same dimensional format as the original input, it now contains contextualized embeddings that incorporate information from the entire sequence.

At the same time, the self-attention mechanism operates bidirectionally, capturing context from both directions simultaneously. This bidirectional nature contributes significantly to the model’s parallel processing capabilities, described by Devlin *et al.* [6].

As the final component of the Transformer layer, a Layer Normalization (LayerNorm) is applied to normalize the inputs for the subsequent Transformer layer. This normalization step improves the model’s stability during backpropagation by mitigating issues such as vanishing or exploding gradients.

3.1.4 Model Output: Contextual Token-Embeddings

After each Transformer layer, a contextualized sequence embedding matrix of shape 512 x 768 is produced. In the BERT model, this output is passed through the next Transformer layer, which shares the same architecture but operates with different learned weights. In total, BERT employs 12 such Transformer layers.

With each successive layer, the embeddings become increasingly enriched with deeper semantic information and contextual relationships, as information from all tokens in the sequence is progressively fused and distributed across every other token, outlined by Devlin *et al.* [6]. As a result, the final model outputs contain highly contextualized knowledge representations.

3.1.5 Output Head

According to Devlin *et al.* [6], the output head utilizes these contextualized token embeddings for so-called downstream tasks, such as Masked Language Modeling (MLM), Next Sentence Prediction (NSP), or Named Entity Recognition (NER), as previously mentioned. Other typical applications include sequence tagging and question answering.

For these tasks, the output embeddings are passed into an additional layer on top of the BERT body, commonly referred to as the BERT head. This head typically consists of a feedforward neural network, adapted to the specific requirements of the task. For example, in the case of the pre-training MLM step, the MLM layer contains a neural layer of dimensions $768 \times 30,522$, mapping one embedding to one of the vocabulary tokens. This output layer also contains learnable weights, which are updated during the respective training processes. The MLM layer will be discussed in more detail in Chapter 5.

3.1.6 The CLS-Token

Although the [CLS] token is not relevant for this work and will not be needed for the NER task, it is an integral part of the BERT model and should be mentioned here for the sake of completeness.

Every input sequence in BERT begins with the [CLS] token, which stands for classification. This token has the same 768-dimensional embedding representation as any other token in the sequence. However, during training, it is specifically designed to capture a generalized representation of the entire input sequence, depending on the nature of the training objectives.

In BERT’s pre-training phase, the [CLS] token is primarily used in the Next Sentence Prediction (NSP) task. During this process, the model receives loss values based on sentence order predictions, encouraging the [CLS] token to develop a representation that reflects the overall meaning of the entire sequence. In further developments of BERT, such as Sentence Transformers, the [CLS] token can be refined and utilized through specialized pooling layers to generate sentence-level embeddings. However, these applications are beyond the scope of this thesis.

3.2 Training Stages of BERT

In summary, encoder-based Transformer models like BERT can be structured into two architectural components: the body and the head. While the body constitutes the majority of the model, the head builds on top of the body’s contextualized embeddings. However, training the model is only possible through the presence of this head, as it connects the body to a specific task, which is necessary to generate a loss function. This loss is essential for enabling backpropagation and forward propagation within Transformer models, allowing the model’s weights to be adjusted in order to capture a comprehensive understanding of language. This process is referred to as the training of the BERT model and can be divided into two distinct phases: pre-training and fine-tuning.

For the sake of completeness in explaining the BERT model, these two methods will be briefly introduced in the following subchapters. However, a more detailed discussion of these training methods will follow in the respective Chapter 5: Pre-training and Chapter 6: Fine-tuning.

3.2.1 Unlabeled Pre-Training

During the pre-training stage, large general text corpora are used to train the weights of the BERT body to develop general language comprehension capabilities. To achieve this, tasks such as Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) are employed to guide the model in learning to represent linguistic structures within its parameters.

Additionally, there is the option to use domain-specific text corpora during pre-training to enhance the model’s understanding within a particular field. This process is referred to as Domain-Adaptive Pre-Training (DAPT).

3.2.2 Labeled Fine-Tuning

The fine-tuning phase builds upon the results of the pre-training phase. In this stage, the MLM head of the pretraining phase is being swapped to a task-specific head, which will be implemented on top of the BERT body, tailored to the specific downstream application, such as Named Entity Recognition (NER). Fine-tuning requires a labeled dataset for both training the task-specific head and for further updating, refining, and adapting the parameters of the already pre-trained BERT body.

4 Named Entity Recognition and the FiNER-139 Dataset

Recognizing Named Entities within financial literature can save a significant amount of manual work. It allows for much faster processing by automatically extracting relevant entities from financial texts. This includes documents such as financial reports and corporate announcements. According to Reyes *et al.* [12], such automation can support both personal and corporate economic decision making by relating to the classification of Named Entities within the economic and financial context.

4.1 Named Entity Recognition Overview

Named Entity Recognition does not only identify entity spans within text but also categorizes these entities into predefined categories known as NER tags. Each NER dataset can define labels that are appropriate for its specific context. To implement NER classification effectively, additional rules must be considered and applied to ensure smooth and accurate classification. Furthermore, evaluating the performance of NER classifiers requires carefully selected metrics that can accurately capture the efficacy of NER models. These evaluation rules and metrics also need to be clearly defined before introducing the FiNER-139 dataset used later in this thesis experiment.

4.1.1 Introducing Named Entity Recognition

Named Entity Recognition (NER) identifies entities within text. These entities can include persons, organizations, locations, dates and many others according to Pakhale [13].

While there are many models that can perform NER tasks, Transformer models like BERT "achieves superior performance and hence stands as the new state-of-the-art [14]." NER classification represents one type of downstream task built on the output embeddings of the BERT model, based on the BERT architecture described in Chapter 3.

While BERT relies on a self-supervised learning paradigm during pre-training, NER heads apply supervised learning. This requires specifically labeled datasets to train the NER head to generate tag sequences for all tokens in the input.

The BERT model learns to classify these labels as provided by the dataset. For this purpose, the output head performs token classification for each token within BERT's output span. Each token can be assigned to one of the NER tags or to none of them.

The most common approach is to implement a feedforward neural network on top of BERT’s output layer. This neural network must be adapted to the task and dataset it is trained on. The final layer must match the total number of NER tags, consisting of all possible combinations of entity tags and positional tags, which will be explained in more detail later. The linear layer with softmax projects each token’s 768-dimensional embedding to probabilities for all tags. The tag with the highest probability will be assigned to the token. The O token, which marks tokens that do not belong to any entity, is treated as a default tag.

4.1.2 The IO2 Tagging Systems

NER uses tagging systems that define how entity categories are structured, especially when an entity consists of more than one word within a text. The IOB2 tagging system, also commonly referred to as BIO, divides each tagging category into two subtypes: Beginning tokens and Inside tokens. Tokens that are not part of any entity are labeled with O, meaning Outside of entities as outlined by Lample *et al.* [15].

These positional tags are combined with the entity tag, indicating both the order within the entity and the entity type. Although there are variations of this scheme, such as IOB1 and IOB2, IOB2 has become the current standard. This standard will also be used in this work.

4.1.3 NER Evaluation

To measure the performance of NER classification, specific evaluation metrics need to be defined. Due to the nature of NER, the distribution of tokens is often highly imbalanced. A classifier that only predicts the majority token for every token might achieve a deceptively high accuracy while providing no meaningful explanatory power.

Therefore, alternative metrics must be used that are better suited to the specific characteristics of NER tasks. These metrics must also account for a variety of different scenarios, which will be discussed in the following subsections.

Token-Level vs Entity-Level Evaluation

In token-level evaluation, each token is assessed by comparing its predicted tag to its true tag. However, this approach is problematic because the majority of tokens are often labeled as O. A classifier can easily exploit this imbalance by simply predicting O for most tokens, leading to inflated accuracy without demonstrating real understanding of entities.

For this reason, Entity-Level Evaluation has become the standard in NER. This method evaluates whether entire entities are correctly identified, considering both the exact span and the correct entity label. Entity-level evaluation provides a more meaningful measure of a model’s true performance in recognizing and classifying entities within text.

Exact Match vs Partial Match

For NER classification the paradigms Exact Match and Partial Match can be used to measure the classification performance. The Exact Match method only counts entities as correct if the entire entity span is correctly classified, from the first token to the last token of the entity. In contrast, the Partial Match method is less strict and also rewards partially correct predictions of entity spans.

For example, if only the first token of a two-token entity span is tagged correctly as part of an entity while the second token is incorrectly labeled as O, the partial match method would still count this as a success, whereas the exact match method would not.

Additionally, the Intersection over Union (IoU) method can be applied to partial matches. This allows for continuous scoring between zero and one, such as assigning a score of 0.5 for partially correct classifications. This provides a more nuanced evaluation compared to binary outcomes like those used in exact match or plain partial match without IoU.

Standard Metric for NER

In contrast to the unreliable accuracy metric, the F1 score has become the standard metric for evaluating NER performance. The F1 score combines both precision and recall into a single measure, capturing two desirable characteristics of a classifier in one value. Vickers *et al.* [16] highlight this in their work, emphasizing the importance of metrics that account for the skewed distribution of tags in NER tasks.

To compute the F1 score meaningfully, it must be based on clear definitions of correct binary classifications, as discussed in the previous sections. Commonly, exact match combined with the entity-level evaluation paradigm is used as the input for calculating the F1 score. This approach will also be applied in the experiments conducted in this work.

Micro vs Macro Averaging

When using the F1 score as the overall evaluation metric, it is important to define how the different NER categories are aggregated. Macro averaging

treats all NER categories equally, giving each category the same weight in the final measurement, regardless of how many instances each category contains. This makes macro averaging particularly suitable for evaluating performance on unbalanced datasets.

In contrast, micro averaging considers all classified entities equally, giving more weight to categories with a larger number of examples. Micro averaging aggregates the contributions of all categories before calculating precision, recall and F1 score.

Both averaging methods can be used simultaneously, and there is no need to choose only one. Reporting both provides a more comprehensive understanding of model performance, especially in the context of imbalanced data.

4.2 Dataset Structure and Tags

The FiNER-139 dataset is a specialized NER dataset that provides predefined tag labels for each word. However, since BERT does not operate at the word level but at the token level, the dataset must first be converted to tokens as part of the preprocessing step. This tokenization and label alignment process is necessary for most NER datasets because annotations are typically provided at the word level in natural text.

When a NER-tagged word is split into multiple tokens during tokenization, the original label must be aligned with the resulting token sequence. For example, the word "HuggingFace" may be labeled as B-Org (Beginning of Organization) in its original form. After tokenization, "HuggingFace" might be divided into two tokens, "Hugging" and "Face", requiring an adjustment to the labeling strategy.

A common practice in such cases is to retain the original label only for the first token. The subsequent sub-token is excluded from the labeling process by assigning it a special placeholder label during preprocessing. This placeholder is later ignored during model training. This alignment step can be automated using rule-based algorithms and has become a standard procedure in the preparation of NER datasets for models such as BERT.

The FiNER-139 dataset consists of approximately 1.1 million sentences annotated with XBRL tags. XBRL, short for Extensible Business Reporting Language, reflects the dataset's origin in the domain of financial reporting. XBRL is a standardized annotation language required by financial authorities to enable more automated processing and submission of financial reports. This connection to regulatory practice gives the FiNER-139 dataset strong real-world relevance for NER applications outlined by Loukas *et al.* [17].

Unlike typical NER datasets, which often contain no more than five entity

tags, FiNER-139 provides a far more granular label set with 139 entity types. However, for the experiments conducted in this thesis, these 139 labels will be grouped and reduced to 10 entity types for the sake of simplification.

5 Pre-training BERT

This chapter provides a detailed explanation of the first training phase of the BERT model: pre-training. Following the examination of the BERT architecture in Chapter 3, which outlined the 110 million parameters of the BERT base variant, it is important to note that these parameters, comprising weights and biases, are ineffective without proper training. The pre-training phase enables BERT to develop an initial understanding of language. This chapter also distinguishes between the various approaches to pre-training.

5.1 Initial Pre-Training

In the first training phase of BERT, two large general-purpose text corpora were used: BookCorpus and English Wikipedia, comprising approximately 2.5 billion words according to Devlin *et al.* [6]. These corpora served as the basis for training within a self-supervised learning paradigm. Unlike supervised learning, where labeled data is required, or unsupervised learning, where no labels are present, self-supervised learning generates labels automatically from unlabeled data. As Araci [18] states, "Language models don't require any labels, since the task is predicting the next word. They can learn how to represent the semantic information." Although this quote specifically refers to decoder models, the underlying principle is similar for encoder-based models, where tokens are predicted based on their surrounding context.

The most widely used approach in encoder models is the Masked Language Modeling (MLM) task. This task is applied to the BERT architecture as a pre-training objective. The MLM head automatically masks individual tokens within sequences in the batched BERT input. In total, 15 percent of all tokens are selected for masking. Of these, 80 percent are replaced with the special [MASK] token, 10 percent are replaced with a random token, and the remaining 10 percent are left unchanged. This distribution improves the robustness of the token prediction task. The [MASK] token is a special token within BERT's vocabulary matrix, similar to tokens such as [UNK] or [SEP], and is assigned its own 768-dimensional embedding in the lookup table. This embedding, like all others, is contextualized throughout the BERT layers and results in a specific representation at the output layer.

From this final output embeddings for each token of a sequence, probabilities over the entire token vocabulary for the [MASK] tokens can be computed using a fully connected linear layer with dimensions 768×30522 . This linear output layer serves as the MLM head on top of the BERT bodies architecture. The most likely token is then selected and compared with the original (unmasked) token. The probability of the correct token is evaluated using the cross-entropy loss function, and the resulting loss is used to update the model via backpropagation.

While MLM is the primary task for learning textual comprehension, a second pre-training objective, Next Sentence Prediction (NSP), was also introduced during the initial training of BERT. In this task, sentences within the corpus were designated as Sentence A and Sentence B in alternating order. The model was then fitted to predict whether Sentence B logically followed Sentence A. This task aimed to enhance the model’s ability to capture inter-sentential relationships using the [CLS] token as a mechanism for sentence-level context learning. However, subsequent research, such as RoBERTa by Liu *et al.* [19], demonstrated that NSP provided limited benefit and could even impair performance. Consequently, more robust BERT variants have omitted NSP entirely, relying solely on the MLM objective. In this work, only MLM will be considered as the pre-training task.

Through the interplay of MLM (and, in the original version, NSP) with these large-scale text corpora, BERT’s parameters were trained to capture general-purpose language representations and comprehension capabilities. The following figures provide a more illustrative overview of this process.

As Devlin *et al.* [6] describe, "BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers," since masked tokens receive information from both their preceding and following context.

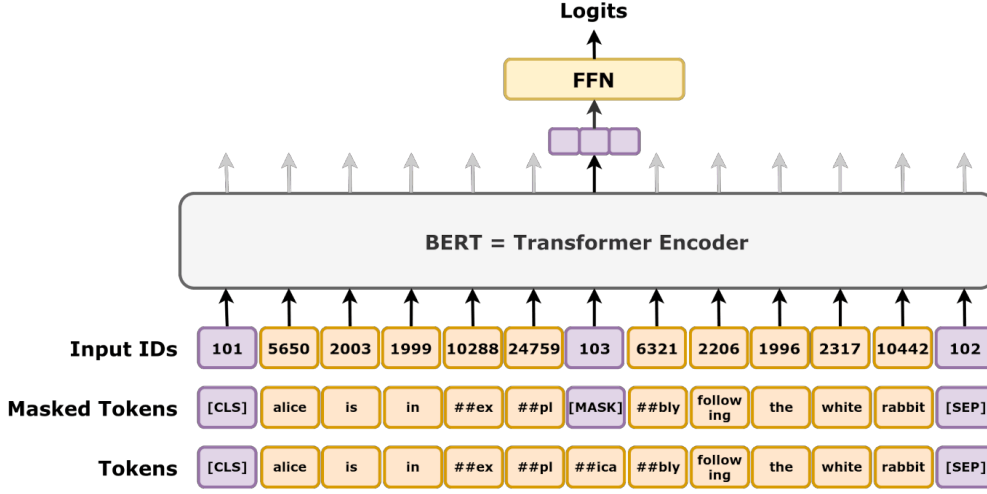


Figure 4: Overall pre-training procedures for BERT. [7]

5.2 Domain Adaptive Pre-training (DAPT)

While the fundamental pre-training process, including Masked Language Modeling (MLM), remains unchanged, different variants of pre-training can be distinguished based on the type of data used. The initial pre-training phase relies on large, general-purpose text corpora. In contrast, methods such as Domain-Adaptive Pre-training (DAPT) and Task-Adaptive Pre-training (TAPT) specialize the training data to better align with the model's downstream task. Although pre-trained transformers have shown success in a range of interdisciplinary domains, they do "not always transfer adequately to the [technical] domain due to its highly specialized language [20]." This limitation has led to the development of domain-specific transformer models. DAPT builds upon the initial general-domain pre-training, also referred to as continual pre-training, by further training the model on domain-specific corpora using the same MLM objective. This process enhances the model's adaptation to and understanding of the target domain, leading to improved performance shown by Singhal *et al.* [21].

In the context of this work, the target domain is the financial sector.

Gururangan *et al.* [3], also discussed in Chapter 2, emphasize that "a second phase of pretraining in-domain (domain-adaptive pretraining) leads to performance gains, under both high- and low-resource settings. [3]" Similarly, Han *et al.* [22] report "substantial improvements over strong BERT baselines, with particularly impressive results on out-of-vocabulary words. [22]"

5.3 Task Adaptive Pre-training (TAPT)

This domain-specific pre-training approach can be further refined to target not only a specific domain but also the particular labeled dataset used for the final fine-tuning task. Task-Adaptive Pre-training (TAPT), like Domain-Adaptive Pre-training (DAPT), builds upon the initial general-domain pre-training. However, TAPT focuses exclusively on the unlabeled version of the task-specific dataset that will later be used for supervised fine-tuning.

Gururangan *et al.* [3] demonstrated that "adapting to the task's unlabeled data (task-adaptive pretraining) improves performance even after domain-adaptive pretraining." In this process, all labels are removed from the dataset. For example, in the case of the FiNER-139 dataset used in this work, the Named Entity Recognition (NER) labels are stripped away. The individual words, originally separated to match NER tags one by one, are then concatenated into continuous text. This flowing text can be treated as standard input for MLM-based pre-training, but is now tailored specifically to the task the model will be fine-tuned on.

This targeted pre-training strategy enables the model to better align with the structure and characteristics of the final task dataset, potentially leading to improved downstream performance.

A summarized comparison of the three pre-training variants is provided in the table below to support the logic structure in the coding section.

Name	Phase	Goal	Data Source
Initial Pretraining	1st Phase	General language comprehension	Large open-domain corpora (Wikipedia, BooksCorpus)
DAPT	2nd Phase	Adaptation to a specific domain	Domain-specific unlabeled texts
TAPT	3rd Phase	Adaptation to task-specific language patterns	Task-related unlabeled texts

Table 1: Different Pretraining Phases [23]

6 Fine-tuning BERT

This chapter provides a detailed examination of the fine-tuning process of BERT, which builds upon the pre-training phase. Typical downstream tasks

are introduced, along with their technical implementation within the BERT model through various output heads. The labeled training process is discussed in greater detail, with a particular focus on Named Entity Recognition (NER), as it represents the core component of this study’s research question and will also be addressed in the coding section later on.

6.1 Downstream Tasks

The contextualized embeddings generated by the BERT body contain valuable information for real-world applications. However, since these embeddings are represented as floating-point numbers, they require further processing before they can be used effectively. Through appropriate processing, these embeddings can be utilized in various downstream tasks. These tasks are implemented on top of the BERT body, in a manner similar to the masked language modeling (MLM) layer used during pre-training.

There are numerous downstream tasks, such as sentiment analysis, which evaluates the emotional tone of sentences, and question answering, which extracts specific information by predicting the start and end positions of the answer within a given sequence. Named Entity Recognition (NER) is another prominent example and will be discussed in greater detail in Chapter 6.2.3, as it constitutes the central focus of this thesis.

6.2 Output-Head

Downstream tasks require a dedicated output layer, similar in function to the masked language modeling (MLM) head used during the pre-training phase. As Devlin *et al.* [6] state, "Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters." [6]

For token-level tasks such as Named Entity Recognition (NER), the output embeddings produced by the BERT body are passed into a task-specific output layer. In contrast to single token classification like NER, the [CLS] token embedding is typically used for sentence-level classification tasks such as sentiment classification as shown in Devlin *et al.* [6].

The standard architecture for an output head consists of a simple linear classification layer, with the number of output neurons determined by the specific downstream task. For example, an NER task with ten possible entity tags would require ten output neurons. Each token then receives one so called logit score for each of the ten entity tags. Each token then has ten logits values corresponding to the ten NER tags. These logits are subsequently converted into probabilities for every NER tag using the Softmax function.

More complex architectures can also be employed. In particular, Conditional Random Fields (CRF) have been used for NER tasks. CRFs apply the Viterbi algorithm to assign tags while also validating the sequence-level consistency of tag transitions. This ensures, for example, that an entity begins with a "Beginning" tag and not an "Inside" or "End" tag. Despite these advantages, linear classification layers have shown greater efficiency in BERT-based models. Therefore, this thesis will also rely on a linear classification approach in the experimental implementation.

Training this output head is referred to as fine-tuning. It requires labeled data specific to the downstream task for which the model is being adapted.

6.3 Labeled Training Process

During pre-training, labels were generated automatically by masking tokens in existing sentences and predicting those tokens for training purposes. In contrast, the fine-tuning process for downstream tasks requires manually annotated labels specific to the task. Each output head must be trained using labeled datasets, where the model’s predictions can be compared against the true outcomes.

A loss function is used to quantify the difference between the model’s predictions and the actual labels. Similar to the masked language modeling (MLM) head in pre-training, this loss value is then used to guide the backpropagation process. An optimizer, configured with a learning rate, adjusts the model’s weights based on the computed loss. The learning rate determines the magnitude of these weight updates at each step of backpropagation.

According to Devlin *et al.* [6], during fine-tuning, not only the parameters of the output head are updated, but the parameters of the BERT body are also adjusted to fit the new downstream task. Although according to Merchant *et al.* [24], fine-tuning affects all layers, the top layers of the BERT model are typically modified more substantially, while the lower layers undergo minimal changes. In most BERT implementations, it is possible to configure which layers should remain trainable. Layers that are not intended to change can be frozen during fine-tuning. It is common to freeze the earlier layers, as they capture general language features that usually do not need to be modified for specific tasks. However, in practice, it is also common to keep all layers trainable.

As outlined by Devlin *et al.* [6], this flexibility in training makes fine-tuning significantly less computationally expensive than full pre-training, which is a key element in the research question addressed in this thesis .

Although fine-tuning can lead to substantial performance improvements

for a specific downstream task, it also binds the BERT model to that task. Once fine-tuned, the model parameters are adapted specifically for one downstream application. Applying the same fine-tuned model to a different task, such as using an NER-optimized model for sentiment classification, can result in poor performance by conflicting weight adaptations. This phenomenon is known as “catastrophic forgetting” (McCloskey & Cohen, 1989).

6.4 NER-Specific Fine-Tuning

The Named Entity Recognition (NER) downstream task is a token-level classification task. Each input token sequence is mapped to a corresponding sequence of tags that identify the entity type of each token. The tags are defined by the labeling scheme used in the dataset, such as BIO (Beginning, Inside, Outside), and combined with the entity types (e.g., person, location, organization). These labels are predetermined by the dataset and fixed during fine-tuning.

As previously mentioned, the most common approach for NER using BERT involves a simple linear classification layer. The number of output neurons in this layer matches the number of NER tags in the labeled dataset. Each output neuron produces a numeric score (logits) for every token in the sequence. By applying the Softmax function, these logits are normalized into probabilities. The tag with the highest probability is then selected as the predicted label for each token.

Fine-tuned BERT models are particularly effective for information extraction tasks such as NER. As Zhao *et al.* [14] note, "Fine-tuned BERT-based models are capable of capturing both syntactic and semantic context, making them well-suited for information extraction tasks such as NER. [14]"

While pre-training enables BERT to generate general-purpose contextual embeddings, fine-tuning adapts the model to perform a specific task. For NER, this involves modifying the embedding space so that tokens associated with different labels become more distinguishable. As Zhou *et al.* [25] explain, fine-tuning "increases the distances between examples associated with different labels, [25]" thus facilitating clearer separations between entity classes in the embedding space.

Although using the same BERT model for different types of downstream tasks tends to degrade performance, since model parameters can only be optimized for one task at a time, fine-tuning the same task across different domains can be beneficial. Transfer learning allows models trained on a source domain to serve as base models for another domain. As Francis *et al.* [26] explain, "NER models trained on labeled data from a source domain can be used as base models and then be fine-tuned with few labeled data [26]"

from a target domain.

7 Methodology: Code Implementation of the Training Pipeline

The aim of this chapter is to provide a clear understanding of the process used to create the training pipeline. This includes explanations of the selected hyperparameters, the implementation methods, and the rationale behind the choice of toolchains and computational devices.

This experiment analyzed whether TAPT without prior DAPT alone can have a significant impact on a model’s performance compared to a base model without any continued pre-training beyond its initial pre-training. For this purpose, two identical BERT base uncased models were used. One model received TAPT, while the other did not. After this step, fine-tuning was performed on both model variants, including the initialization of the linear NER layer with the head size corresponding to each tag, which in total amounts to 21 (a detailed breakdown is provided later in this chapter).

The labeled FiNER 139 dataset served as the starting point. For the TAPT process, labels were ignored, and all tokens were merged back into their original text so that MLM could be applied in the TAPT training pipeline. Separately, this dataset was manually processed to reduce the 139 different tags to 10 categories that summarize all of the original labels. The dataset is already split into three parts: train, validation, and test.

While the extracted text was used as a separate training dataset for the TAPT pipeline, the train and validation splits were used for the training process, and the test split was reserved solely for benchmarking. In this final step, the two fully trained and fine-tuned models were compared against each other using the same test split.

The figure below presents a visual conceptualization of the entire training pipeline.

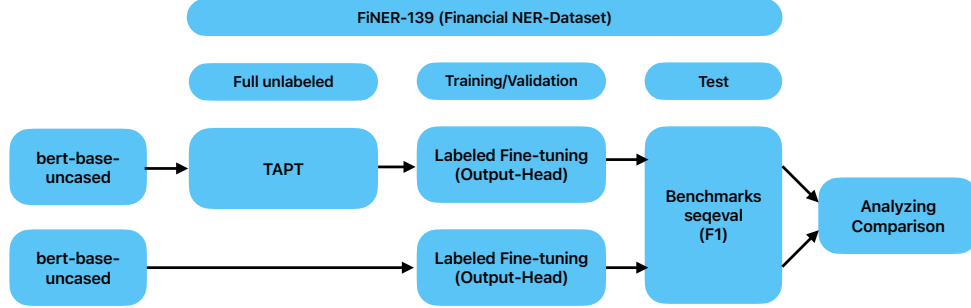


Figure 5: Overview of the training setup. Own illustration.

7.1 Frameworks and Tools

Both the retrained BERT base uncased model and the FiNER 139 dataset were imported from Hugging Face. Hugging Face has established itself as the leading platform for Natural Language Processing by providing a vast collection of transformer-based models through its Transformers library for Python. This library includes the so-called Trainer API, which offers a production-ready template for training different types of transformers with a unified syntax. Hyperparameters can be specified directly without the need for complex workarounds. The transformers library for the codebase was used in version 4.55.0.

Hugging Face also provides the Python library datasets, which grants access to a large collection of datasets for training, many of which are available directly on the Hugging Face Hub. This is also the case for the FiNER 139 dataset.

However, Hugging Face itself only provides models, datasets, and APIs for integrating them into Python. To work with these models and datasets efficiently, a lower level deep learning framework, which provides the foundation for this Hugging Face API is required. The most common frameworks for NLP tasks in deep learning are PyTorch, TensorFlow, and JAX/Flax. These frameworks enable computations to be executed on GPUs instead of solely on CPUs, allowing for massive parallelization, which is essential for training deep learning models such as transformers.

This experiment used PyTorch version 2.2.0. PyTorch was selected because it is the standard implementation most compatible with Hugging Face APIs and is widely adopted as the preferred framework in research.

7.2 The Model: bert-base-uncased

BERT base uncased is the most commonly used BERT implementation on Hugging Face. While it follows the BERT architecture, this version uses the base configuration, which includes 12 transformer layers, 12 attention heads, 768 dimensional embeddings, and a total of 110 million parameters. The model also comes with its own tokenizer, which operates exclusively on uncased text. All letters are converted to lowercase, and no distinction is made between upper and lower case characters during the tokenization process.

7.3 Dataset Preparation

This section explains in detail the dataset preparation process that was outlined in the introduction to this chapter.

7.3.1 The FiNER 139 Dataset

The FiNER 139 dataset, short for Fine-grained Named Entity Recognition Dataset, is notable for its exceptionally high number of entity categories, a total of 139. In comparison, most NER datasets contain fewer than 10 entity types. While the large number of categories provides fine-grained detail, it also poses a challenge: the risk that general patterns cannot be effectively learned, as there may be no clear separations between token embedding representations. This can increase the danger of overfitting in the weights of a fine-tuned BERT model.

In addition, the dataset uses the BIO tagging scheme, assigning two variants to each original label: B for the beginning of an entity and I for tokens inside the entity. Since the fine-granularity problem is not part of the research question addressed in this thesis, a manual simplification was carried out to create the final dataset used for model training. Specifically, the 139 entity categories were mapped to 10 broader, more general categories through a manually defined mapping. This mapping table is provided by the mappings implementation in the codebase.

Both the TAPT-enhanced model and the baseline model in this study were fine-tuned on this merged-category dataset, which contains 21 NER tags when accounting for the BIO scheme. The 10 general categories become 20 tags due to the B and I labels, and the outside tag (O) is included as the twenty-first label.

The dataset is stored in the DatasetDict format, a common structure in the Hugging Face ecosystem. A DatasetDict is composed of multiple Dataset

objects with identical formats but different purposes: training, validation, and testing.

Each split contains three columns: `id`, `tokens`, and `ner_tags`. The train split is used for parameter optimization during training. The validation split normally is used for hyperparameter tuning, ensuring that the model is evaluated on different data from the training set to avoid data leakage, the phenomenon where a model is overfitted to data it has already seen. However, in this work hyperparameters were chosen manually. Finally, the test split is reserved exclusively for final evaluation, providing a completely new and independent dataset that the model has never encountered during parameter training or hyperparameter validation.

7.3.2 Label Reduction

Following the manual mapping process, all 139 entity labels were merged into 10 general categories. These will be used as the new labels:

- **Debt**
- **Equity**
- **Compensation**
- **Revenue**
- **Tax**
- **Lease**
- **Acquisition**
- **Contingency**
- **Assets**
- **Other**

These 10 categories generalize the original fine-grained labels. With the BIO scheme applied, this results in 20 labels, and with the outside tag (O), the total becomes 21. The mapping was applied to all splits of the dataset, train, validation, and test, within the `00_FiNER139-TagMerging` Jupyter Notebook. O tokens remained unchanged.

The resulting dataset format is consistent across all splits, with each row containing between 2 and 1,260 tokens.

7.3.3 Text Extraction for TAPT

In the 00_FiNER139-Text-Extraction for TAPT Jupyter Notebook, most tokens from the train split were extracted and reconstructed into their original running-text format. This version of the dataset contained only one feature per row, namely the reconstructed sentence, and was used for the TAPT model’s continued pre-training with the MLM objective.

During preprocessing, approximately 10,000 rows containing short boilerplate text were removed, as they were unlikely to contribute to meaningful language comprehension. This resulted in over 890,000 rows remaining for TAPT training.

This dataset was then saved in Arrow format. Arrow is widely used within the Hugging Face ecosystem and stores data in an efficient columnar binary format, allowing faster access to specific columns and enabling more efficient computation than formats such as JSON or CSV. This is particularly important for large-scale data processing required by transformer models.

7.4 Devices and Environments

Since both TAPT as a form of continued pre-training and the subsequent fine-tuning tasks are highly computationally intensive, the experiments in this thesis were carried out using a single NVIDIA SXM A100 GPU with 80 GB of VRAM. This resource was provided by RunPod, with the hourly cost for the NVIDIA SXM A100 (80 GB) set at \$1.74. This high-performance hardware was essential for efficiently processing the large dataset and training transformer-based models within reasonable time frames.

Due to the long and uninterrupted runtimes required for these tasks, all training procedures were executed as standalone Python scripts rather than in Jupyter Notebooks. While Jupyter Notebooks are advantageous for interactive experimentation and visualization, their higher susceptibility to interruptions makes them less suitable for extended training processes where only the final trained model is required as output.

All training scripts utilized the Hugging Face Trainer API, which provides a stable and unified interface for model training while allowing direct integration of hyperparameters without complex workarounds. This ensured reproducibility and consistency across the TAPT and fine-tuning stages.

7.5 TAPT Implementation

The TAPT training process was implemented using two splits (train and validation) of the datasets within the Python training script. The train split

contained approximately 890,000 unlabeled sequences, as described earlier in Section 7.3.3. Since all hyperparameters were set manually, no automated hyperparameter search was conducted. However, performance metrics were calculated during the training process in order to monitor improvements throughout the run directly in the terminal. For these interim evaluations, as well as for the final evaluation of the fully trained model, the validation split was used.

With regard to the hyperparameters, a maximum sequence length of 192 tokens was selected. While BERT can process sequences up to 512 tokens, this smaller length was chosen for two reasons: first, the computational cost increases quadratically with sequence length, and second, most sequences in the dataset do not exceed 192 tokens. The distribution of token lengths in the revised FiNER 139 dataset is shown in Table 2, illustrating that the vast majority of samples fall well below the maximum allowed by BERT.

Token Range	Count
> 512	379
256–512	2.985
192–256	5.398
128–192	24.995
64–128	181.523
48–64	146.765
10–48	505.454
< 10	9.889
10–192	858.737

Table 2: Token length distribution obtained through own dataset analysis in the `00b_FiNER139-Text-Extraction-Analysis.ipynb` Jupyter Notebook available in the GitHub Repository

Tokens beyond a length of 192 were truncated, as they are not required for TAPT. Since MLM generates its own labels, no important label information is lost compared to fixed-label training setups.

In MLM, a standard masking rate of 15 percent of all tokens was applied. Masked tokens were selected randomly and differed in each epoch. This process was implemented via the `DataLoader`, which is responsible for creating batches from the dataset, together with the `DataCollator`, which applies the masking. Over several epochs, different tokens within the same sentences are masked, resulting in more robust learning and reducing the risk of overfitting.

In Gururangan *et al.* [3], smaller datasets were used for TAPT, but training was carried out for up to 100 epochs. Overfitting was avoided because

the masked labels changed in each epoch. Based on this, the present work pursued a different approach: instead of using a high number of epochs, only two epochs were performed, but on a larger dataset than in Gururangan *et al.* [3]. This was intended to increase domain diversity while still closely matching the target task.

Batch size in a training pipeline must be chosen according to both the GPU’s capabilities and the desired robustness of training, as it involves trade-offs. MLM requires stronger generalization to adapt effectively to a domain, while fine-tuning tasks such as NER require less generalization, as excessive fitting to the downstream task can degrade the model’s broader text comprehension, a phenomenon known as catastrophic forgetting.

Batch size refers to the number of sequences processed on the GPU in one forward and one backward propagation step, together referred to as a "step." For each step, the loss for the entire batch is calculated as the average over all sequences, and the model parameters are updated accordingly. Forward and backward propagation are highly memory-intensive, requiring sufficient GPU VRAM. Insufficient memory leads to out-of-memory (OOM) errors.

Batch size also strongly influences generalization. Gradients used to update model parameters are derived from the loss landscape. Greater variance in the loss landscape across batches leads to higher gradient noise. Larger batch sizes yield more accurate approximations of the overall loss landscape, but this can make the optimizer more prone to becoming trapped in local minima. Smaller batch sizes produce noisier, less precise gradients, which can help the optimizer explore the loss landscape more widely and potentially find better global minima. Smaller batches also increase the number of parameter updates per epoch.

In MLM, the reduced generalization ability of large batch sizes can be mitigated by using a higher learning rate, which allows the optimizer to make larger jumps in the loss landscape. Since MLM continues to improve general text comprehension, catastrophic forgetting is less of a concern. Nevertheless, it is best practice to gradually reduce the learning rate over time, allowing training to converge more stably with smaller parameter updates.

In this work, the computational capacity of the A100 GPU allowed for a relatively large batch size of 384, combined with a comparatively high initial learning rate of 7.5×10^{-5} and the FP-16 format for the parameters. A linear scheduler was used to decrease the learning rate steadily to zero by the end of training, ensuring a stable conclusion.

The entire TAPT training process took approximately 30 minutes, utilizing on average 99 percent of the GPU’s computational capacity, as measured by the NVIDIA SMI terminal monitoring tool. During this 30 minutes, the loss calculated by the Hugging Face Trainer API decreased consistently.

Upon completion, the model was saved in a format ready for use in the subsequent fine-tuning scripts.

While 30 minutes is relatively short compared to the duration of the fine-tuning stages, this supports the central thesis of this work regarding the efficiency of additional continued pre-training. More extensive and intensive pre-training remains a subject for future research.

7.6 Labeled Fine-Tuning Implementation

For the core experiments in this thesis, two fine-tuning scripts were implemented, identical in every respect except for the underlying BERT model being fine-tuned. This setup allowed for a direct comparison of their performance. One script used the standard BERT base uncased model, while the other used the TAPT-adapted version of BERT, referred to in the scripts as `tapt-bert`, described in the preceding sections.

In the fine-tuning stage for the Named Entity Recognition (NER) task, both models were initialized with the total number of labels as the size of the output head, using Hugging Face’s predefined `BertForTokenClassification()` function.

In fine-tuning for downstream tasks, the model’s existing text comprehension capabilities are leveraged rather than retrained from scratch. As a result, gradient updates must be applied carefully to avoid degrading the language comprehension features acquired during pre-training. Large batch sizes tend to produce highly precise gradient directions, but when combined with higher learning rates, they increase the risk of overfitting to the downstream task, leading to catastrophic forgetting of general language features. Therefore, higher batch sizes require proportionally lower learning rates.

For this reason, a batch size of 64 was selected, paired with a learning rate of 3×10^{-5} . No learning rate scheduler was used, as the initial learning rate was already relatively low; further reduction would have risked slowing learning progress to the point of diminishing returns.

Regarding the number of epochs, both fine-tuning pipelines were trained for three epochs on the full train split of the dataset. Three epochs are commonly recommended for fine-tuning tasks, as they typically provide sufficient training without a significant risk of overfitting.

The maximum sequence length was set to BERT’s full capacity of 512 tokens. In downstream tasks, the labels are of high value and should be preserved in their entirety wherever possible. Truncating sequences would risk losing crucial labeled information. Nevertheless, 379 sequences in the dataset exceeded 512 tokens and were truncated to this limit to avoid introducing unnecessary complexity into the pipeline by applying a method called

chunking, given their minimal proportion of the overall dataset.

While the standard numerical representation in deep learning is FP32 (32-bit floating point), both fine-tuning pipelines in this work used BF16 (bfloat16). FP refers to floating point, and the number indicates the bit length used for representation. BF16 uses 16 bits for storage, thereby reducing memory requirements by half compared to FP32, while maintaining the same 8-bit exponent as FP32. This allows for a much larger representable range of values than FP16, which has only a 5-bit exponent.

FP16 is known to suffer from underflow and overflow issues due to its limited range, often requiring computationally expensive loss scaling during training. BF16 mitigates these problems by preserving the larger exponent size, eliminating the need for loss scaling, while accepting a modest reduction in precision in the mantissa. This trade-off enables more efficient training without substantial loss of numerical stability.

Both fine-tuning scripts required approximately two hours each, utilizing on average 90 percent of the GPU’s computational capacity, as measured by the NVIDIA SMI terminal monitoring tool.

7.7 Metrics

Model performance was evaluated using the sequeval library, which is short for "sequence evaluation." This library is specifically designed to compute evaluation metrics for sequence labeling tasks such as NER. By providing the predicted labels and the corresponding ground truth labels to its evaluation function, sequeval can easily calculate the most commonly used NER metrics.

These metrics were computed for both models, the base model and the TAPT-pretrained model, to enable a direct, one-to-one comparison. All valid tokens were assigned one of the defined labels. Subwords or special tokens, such as [CLS], [PAD], and [SEP], were assigned the value -100 so that they would be ignored by sequeval during evaluation.

The F1-score, recall, and precision metrics were computed solely on valid entity labels, excluding O-labeled tokens. Only the accuracy metric included O tokens. However, accuracy should not be given excessive weight when interpreting results for NER, as O tokens represent the majority of the dataset (approximately 90 percent). In such cases, a model predicting only O tokens would achieve a deceptively high accuracy score while performing poorly on actual entity recognition. For this reason, accuracy is considered a weak metric for guiding NER optimization.

For the purpose of reproduction, the random seed 42 was used in the evaluation notebook `04_Model_Comparison_Eval.ipynb` within the training arguments.

8 Results

Both models produced reasonable values for all metrics measured across the different tag categories, with a brief summary of the results shown in the table below.

Model	Precision	Recall	F1	F1 Macro	Accuracy
Base	0.8116	0.8928	0.8502	0.8482	0.9981
TAPT	0.8223	0.9004	0.8596	0.8582	0.9982

Table 3: Evaluation metrics for Base and TAPT models

8.1 Overall Performance

Both models achieved reasonable scores for the Named Entity classifications, with F1 micro scores above 0.85. The table below shows that some NER tags reached higher F1 values than others. This can be attributed to the varying difficulty of individual tags due to their inherent characteristics. Some tag categories are more likely to be confused with others because of overlaps between the categories.

It can also be observed that the precision values are consistently lower than the recall values across all tags and models. This indicates that the models produced more false positives than false negatives. In other words, the models tend to assign tokens to tags rather than classify them as the "O" tag, which is not included in the F1 metrics. This behavior is advantageous because NER datasets are mostly made up of "O" tokens, which serve as the default and carry no semantic meaning.

For the NER task, favoring false positives over false negatives can be evaluated positively, since incorrectly labeled tokens are less problematic than completely unrecognized tokens, which are more likely to be overlooked in the large number of majority "O" tokens.

Model	Entity	Precision	Recall	F1
Base	Acquisition	0.7821	0.8531	0.8160
TAPT	Acquisition	0.7731	0.8504	0.8099
Base	Assets	0.8088	0.9008	0.8523
TAPT	Assets	0.8216	0.9090	0.8631
Base	Compensation	0.8419	0.9237	0.8809
TAPT	Compensation	0.8508	0.9330	0.8900
Base	Contingency	0.8122	0.8577	0.8343
TAPT	Contingency	0.8306	0.8647	0.8473
Base	Debt	0.7988	0.9017	0.8471
TAPT	Debt	0.8095	0.9085	0.8562
Base	Equity	0.8159	0.8734	0.8437
TAPT	Equity	0.8247	0.8805	0.8517
Base	Lease	0.8082	0.8755	0.8405
TAPT	Lease	0.8276	0.8785	0.8523
Base	Other	0.7923	0.8599	0.8247
TAPT	Other	0.8015	0.8840	0.8408
Base	Revenue	0.8443	0.9039	0.8731
TAPT	Revenue	0.8693	0.9108	0.8896
Base	Tax	0.8378	0.9029	0.8691
TAPT	Tax	0.8531	0.9116	0.8814

Table 4: Precision, Recall, and F1 scores per entity type for Base and TAPT models

8.2 TAPT vs. Classic Fine-tuning

The most relevant interpretation of these results in relation to the central question of this thesis is the difference between the TAPT fine-tuned model and the baseline fine-tuned model. These differences are presented in the table at the beginning of this chapter. The central metric, F1 micro, shows a difference of 0.0094 in F1 value. All other metrics, including F1 global, precision, and recall, display a similar difference of approximately 0.01. Precision and recall improve with TAPT by 0.0107 and 0.00776 respectively.

This indicates that TAPT reduces the gap between precision and recall. It compensates for the lower performance in classifying false positives while also improving recall, even if to a lesser extent. Continued TAPT pre-training

could potentially maintain or enhance this trend.

8.3 Comparison and Interpretation

While Gururangan *et al.* [3] achieved F1 score improvements of 0.01 to 0.03 across different domains, each using short datasets and one hundred epochs, the measurable performance boost of TAPT in this work, with a F1 score boost of 0.0094, can be evaluated positively, especially considering the low number of only two epochs. This approach may still hold considerable potential for further pre-training by increasing the number of epochs. A higher number of epochs could strengthen the model’s textual understanding of the domain by exposing it repeatedly to the same data, but with varied token masking. This would allow the model to learn task-specific structures more deeply.

Nevertheless, it has been demonstrated that a high variety of data with a low number of epochs can still yield better performance than baseline models, and that DAPT is not a prerequisite for improving the model with TAPT. TAPT could even be considered a better alternative to DAPT, provided that sufficient data is available, particularly in low-resource environments.

9 Conclusion

This work demonstrates that resource efficient TAPT training configurations can yield measurable performance improvements of the F1 score of approximately 0.0094 in F1 value compared to the baseline model. Relevant adaptations of the BERT model for financial NER tasks can be made with TAPT only, without relying on DAPT. Given the relatively small training investment, since TAPT required only about 30 minutes in addition to the normal fine-tuning time of two hours, this additional pre-training can be worthwhile for tasks where even small performance gains are valuable. However, it should be noted that the additional setup costs could overcomplicate the NER classification process and may not be worthwhile for many use cases. A limitation of this work is that no DAPT was performed, so no direct performance comparison between TAPT and DAPT can be made, even though such a comparison would be possible for NER in the financial domain using the FiNER-139 dataset. Furthermore, only one model and one dataset were used for analyzing the NER capabilities of encoders within the financial domain. Future research could explore more diverse approaches and tests. To answer the research question, TAPT can indeed enhance the performance of financial Named Entity Recognition tasks compared to baseline models without

continued pre-training. Although the observed performance gain is relatively modest, the required computational resources were also rather small. Therefore, TAPT represents a valid and efficient approach for improving model performance in scenarios where even small gains are considered valuable.

9.1 Future Work

Future work could aim to improve NER classification by employing more computationally intensive settings and by implementing DAPT prior to TAPT which Gururangan *et al.* [3] already showed is the most precise in F1 terms. The model could also be pre-trained on a broader financial text corpus. Additionally, pre-trained DAPT models for the financial domain, such as FinBERT, could be used instead of training DAPT from scratch, as demonstrated by Peng *et al.* [5], who showed that such domain specific models already provide an improved baseline. Models of this kind can also be found on Hugging Face.




Another potentially valuable direction for future work is to keep the current TAPT training configuration unchanged except for increasing the number of epochs, which could lead to better text representation learning from the TAPT data. Gururangan *et al.* [3] showed that this approach is effective for small TAPT datasets with epoch values as high as 100.

Different encoder model variants could also be explored:

- A) **RoBERTa:** For larger and more robust models as an alternative to BERT, RoBERTa (Robustly Optimized BERT) could be implemented. While `roberta-base` has 125 million parameters, there is also a `roberta-large` variant with around 355 million parameters.
- B) **DistilBERT:** If efficiency remains the goal, DistilBERT could be a better option. It is a compressed version of BERT with only 66 million parameters, sacrificing only 1 to 3% performance in several benchmarks while maintaining robust context representations and making more efficient use of training data with better generalization.
- C) **ELECTRA:** Another promising approach is the ELECTRA model, which offers robust context representations and high efficiency. While `electra-base` also has 110 million parameters, it uses a more efficient training method consisting of a generator and discriminator. This technique could be integrated into the fine-tuning process to allow training on larger datasets with the same computational resources.

A Appendix: Code

The full training pipelines with all training python scripts and preprocessing jupyter notebooks can be found on GitHub. The FiNER-139 Dataset as the bert-base-uncased model are available on Hugging Face:

-  [TobiHD/Bachelor_Thesis on GitHub](https://github.com/TobiHb/Bachelor_Thesis)
https://github.com/TobiHb/Bachelor_Thesis
-  [FiNER-139 on Hugging Face](https://huggingface.co/datasets/nlpauueb/finer-139)
<https://huggingface.co/datasets/nlpauueb/finer-139>
-  [bert-base-uncased Model on Hugging Face](https://huggingface.co/google-bert/bert-base-uncased)
<https://huggingface.co/google-bert/bert-base-uncased>

References

- [1] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, 1950. DOI: [10.1093/mind/LIX.236.433](https://doi.org/10.1093/mind/LIX.236.433). [Online]. Available: <https://doi.org/10.1093/mind/LIX.236.433>.
- [2] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, *et al.*, Eds., 2017, pp. 5998–6008. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>.
- [3] S. Gururangan, A. Marasovic, S. Swayamdipta, *et al.*, “Don’t stop pre-training: Adapt language models to domains and tasks,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, Eds., Association for Computational Linguistics, 2020, pp. 8342–8360. DOI: [10.18653/v1/2020.acl-main.740](https://doi.org/10.18653/v1/2020.acl-main.740). [Online]. Available: <https://doi.org/10.18653/v1/2020.acl-main.740>.

- [4] Y. Yang, M. C. S. Uy, and A. Huang, “Finbert: A pretrained language model for financial communications,” *CoRR*, vol. abs/2006.08097, 2020. arXiv: [2006.08097](https://arxiv.org/abs/2006.08097) [Online]. Available: <https://arxiv.org/abs/2006.08097>.
- [5] B. Peng, E. Chersoni, Y.-Y. Hsu, and C.-R. Huang, “Is domain adaptation worth your investment? comparing BERT and FinBERT on financial tasks,” in *Proceedings of the Third Workshop on Economics and Natural Language Processing*, U. Hahn, V. Hoste, and A. Stent, Eds., Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 37–44. DOI: [10.18653/v1/2021.econlp-1.5](https://doi.org/10.18653/v1/2021.econlp-1.5). [Online]. Available: <https://aclanthology.org/2021.econlp-1.5/>.
- [6] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds., Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: [10.18653/v1/N19-1423](https://doi.org/10.18653/v1/N19-1423). [Online]. Available: <https://doi.org/10.18653/v1/n19-1423>.
- [7] D. V. Godoy, *High-level schematic diagram of bert*, <https://dvgodoy.github.io/dl-visuals/BERT/>, Licensed under CC BY 4.0, n.d.
- [8] V. Sachidananda, J. S. Kessler, and Y. Lai, “Efficient domain adaptation of language models via adaptive tokenization,” in *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing, SustainLP@EMNLP 2021, Virtual, November 10, 2021*, N. S. Moosavi, I. Gurevych, A. Fan, *et al.*, Eds., Association for Computational Linguistics, 2021, pp. 155–165. DOI: [10.18653/v1/2021.SUSTAINLP-1.16](https://doi.org/10.18653/v1/2021.SUSTAINLP-1.16). [Online]. Available: <https://doi.org/10.18653/v1/2021.sustainlp-1.16>.
- [9] X. Song, A. Salcianu, Y. Song, D. Dopson, and D. Zhou, “Fast word-piece tokenization,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, M. Moens, X. Huang, L. Specia, and S. W. Yih, Eds., Association for Computational Linguistics, 2021, pp. 2089–2103. DOI: [10.18653/v1/2021.EMNLP-MAIN.160](https://doi.org/10.18653/v1/2021.EMNLP-MAIN.160). [Online]. Available: <https://doi.org/10.18653/v1/2021.emnlp-main.160>.

- [10] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, “What does BERT look at? an analysis of bert’s attention,” in *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@ACL 2019, Florence, Italy, August 1, 2019*, T. Linzen, G. Chrupala, Y. Belinkov, and D. Hupkes, Eds., Association for Computational Linguistics, 2019, pp. 276–286. DOI: [10.18653/v1/W19-4828](https://doi.org/10.18653/v1/W19-4828). [Online]. Available: <https://doi.org/10.18653/v1/W19-4828>.
- [11] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Encoder-only attention is all-to-all*, https://github.com/d2l-ai/d2l-en/blob/master/chapter_attention-mechanisms-and-transformers/large-pretraining-transformers.md, Licensed under CC BY-SA 4.0, n.d.
- [12] D. D. L. Reyes, A. Barcelos, R. Vieira, and I. H. Manssour, “Related named entities classification in the economic-financial context,” in *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation, EACL 2021, Online, April 19, 2021*, H. Toivonen and M. Boggia, Eds., Association for Computational Linguistics, 2021, pp. 8–15. [Online]. Available: <https://aclanthology.org/2021.hackashop-1.2/>.
- [13] K. Pakhale, “Comprehensive overview of named entity recognition: Models, domain-specific applications and challenges,” *CoRR*, vol. abs/2309.14084, 2023. DOI: [10.48550/ARXIV.2309.14084](https://doi.org/10.48550/ARXIV.2309.14084). arXiv: [2309.14084](https://arxiv.org/abs/2309.14084). [Online]. Available: <https://doi.org/10.48550/arXiv.2309.14084>.
- [14] X. Zhao, J. Greenberg, Y. An, and X. T. Hu, “Fine-tuning BERT model for materials named entity recognition,” in *2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, December 15-18, 2021*, Y. Chen, H. Ludwig, Y. Tu, *et al.*, Eds., IEEE, 2021, pp. 3717–3720. DOI: [10.1109/BIGDATA52589.2021.9671697](https://doi.org/10.1109/BIGDATA52589.2021.9671697). [Online]. Available: <https://doi.org/10.1109/BigData52589.2021.9671697>.
- [15] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, “Neural architectures for named entity recognition,” in *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, K. Knight, A. Nenkova, and O. Rambow, Eds., The Association for Computational Linguistics, 2016, pp. 260–270. DOI: [10.18653/v1/N16-1030](https://doi.org/10.18653/v1/N16-1030). [Online]. Available: <https://doi.org/10.18653/v1/n16-1030>.

- [16] P. Vickers, L. Barrault, E. Monti, and N. Aletras, “We need to talk about classification evaluation metrics in NLP,” *CoRR*, vol. abs/2401.03831, 2024. DOI: [10.48550/ARXIV.2401.03831](https://doi.org/10.48550/ARXIV.2401.03831). arXiv: [2401.03831](https://arxiv.org/abs/2401.03831). [Online]. Available: <https://doi.org/10.48550/arXiv.2401.03831>.
- [17] L. Loukas, M. Fergadiotis, I. Chalkidis, *et al.*, “Finer: Financial numeric entity recognition for XBRL tagging,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, *ACL 2022, Dublin, Ireland, May 22-27, 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Association for Computational Linguistics, 2022, pp. 4419–4431. DOI: [10.18653/V1/2022.ACL-LONG.303](https://doi.org/10.18653/V1/2022.ACL-LONG.303). [Online]. Available: <https://doi.org/10.18653/v1/2022.acl-long.303>.
- [18] D. Araci, “Finbert: Financial sentiment analysis with pre-trained language models,” *CoRR*, vol. abs/1908.10063, 2019. arXiv: [1908.10063](https://arxiv.org/abs/1908.10063). [Online]. Available: <http://arxiv.org/abs/1908.10063>.
- [19] Y. Liu, M. Ott, N. Goyal, *et al.*, “Roberta: A robustly optimized BERT pretraining approach,” *CoRR*, vol. abs/1907.11692, 2019. arXiv: [1907.11692](https://arxiv.org/abs/1907.11692). [Online]. Available: <http://arxiv.org/abs/1907.11692>.
- [20] E. Laparra, A. Mascio, S. Velupillai, and T. Miller, “A review of recent work in transfer learning and domain adaptation for natural language processing of electronic health records,” *Yearbook of Medical Informatics*, vol. 30, pp. 239–244, Aug. 2021. DOI: [10.1055/s-0041-1726522](https://doi.org/10.1055/s-0041-1726522).
- [21] P. Singhal, R. Walambe, S. Ramanna, and K. Kotecha, “Domain adaptation: Challenges, methods, datasets, and applications,” *IEEE Access*, vol. 11, pp. 6973–7020, 2023. DOI: [10.1109/ACCESS.2023.3237025](https://doi.org/10.1109/ACCESS.2023.3237025). [Online]. Available: <https://doi.org/10.1109/ACCESS.2023.3237025>.
- [22] X. Han and J. Eisenstein, “Unsupervised domain adaptation of contextualized embeddings for sequence labeling,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Association for Computational Linguistics, 2019, pp. 4237–4247. DOI: [10.18653/V1/D19-1433](https://doi.org/10.18653/V1/D19-1433). [Online]. Available: <https://doi.org/10.18653/v1/D19-1433>.
- [23] C. (OpenAI), *Bert pretraining phases*, <https://chatgpt.com/share/689df9ab-e8c8-800b-b3fb-9b3417b17887>, Zugriff am 14. August 2025, 2025.

- [24] A. Merchant, E. Rahimtoroghi, E. Pavlick, and I. Tenney, “What happens to BERT embeddings during fine-tuning?” In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2020, Online, November 2020*, A. Alishahi, Y. Belinkov, G. Chrupala, D. Hupkes, Y. Pinter, and H. Sajjad, Eds., Association for Computational Linguistics, 2020, pp. 33–44. DOI: [10 . 18653 / V1 / 2020 . BLACKBOXNLP - 1 . 4](https://doi.org/10.18653/v1/2020.blackboxnlp-1.4). [Online]. Available: <https://doi.org/10.18653/v1/2020.blackboxnlp-1.4>.
- [25] Y. Zhou and V. Srikumar, “A closer look at how fine-tuning changes BERT,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Association for Computational Linguistics, 2022, pp. 1046–1061. DOI: [10 . 18653 / V1 / 2022 . ACL - LONG . 75](https://doi.org/10.18653/v1/2022.acl-long.75). [Online]. Available: <https://doi.org/10.18653/v1/2022.acl-long.75>.
- [26] S. Francis, J. V. Landeghem, and M. Moens, “Transfer learning for named entity recognition in financial and biomedical documents,” *Inf.*, vol. 10, no. 8, p. 248, 2019. DOI: [10 . 3390 / INF010080248](https://doi.org/10.3390/info10080248). [Online]. Available: <https://doi.org/10.3390/info10080248>.