



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Sentiment-Informed Price Prediction in Algorithmic Trading: An Emoji-Based Approach

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Paolo Pietro Bucci**

Student ID: 10964818

Advisor: Prof. Simone Formentin

Co-advisors: Prof. Daniele Mazzarina, Dr. Raffaele Giuseppe Cestari

Academic Year: 2024-25

Abstract

This thesis presents a Large Language Model-based approach to the task of financial market sentiment classification. We propose an innovative methodology to extend, train, and fine-tune existing BERT-based models in order to enhance their sentiment classification performance in the context of emoji-rich online content such as social media posts. By extending the original vocabulary and layers of a base model, the network is made capable of exploiting emojis and non-verbal cues, thereby improving its ability to correctly extract market sentiment for a target financial asset.

The extracted sentiment is then used as an additional feature, alongside classical technical indicators, as input to a Recurrent Neural Network for price movement forecasting. In this way, we aim to verify the importance of online investor sentiment in driving market movements and trends. Furthermore, we quantify the impact that the ability of the language model to accurately capture sentiment has on the overall performance of price forecasting.

Keywords: Large Language Model, LLM, Emojis, Sentiment Analysis, Time Series Forecasting

Abstract in lingua italiana

Questa tesi di ricerca propone un approccio basato sui moderni Large Language Model per affrontare la classificazione del sentiment di mercato relativo a un asset finanziario. In particolare, verrà proposto un metodo innovativo che consiste nell'estendere, addestrare e finetunare un modello preesistente basato sull'architettura BERT, al fine di migliorarne le prestazioni originali nel compito di classificazione del sentiment di mercato, concentrandosi soprattutto sul sentiment espresso tramite contenuti provenienti da fonti online, come i social media, ricchi di emoji. Estendendo il vocabolario e layer originali del modello, la rete diventa in grado di sfruttare il contenuto espressivo delle emoji nella classificazione del sentiment.

Il sentiment estratto dai post verrà poi utilizzato come caratteristica aggiuntiva, insieme ai classici indicatori tecnici, come input per una rete neurale ricorrente per prevedere i movimenti di prezzo del medesimo asset finanziario di cui è stato classificato il sentiment. Con questa strategia l'obiettivo è valutare l'importanza del sentiment degli investitori nel guidare i movimenti di prezzo del mercato. Inoltre, è possibile quantificare l'impatto che l'accuratezza del modello linguistico nel catturare il sentiment ha sulle prestazioni della previsione dei prezzi.

Contents

| | |
|---|------------|
| Abstract | i |
| Abstract in lingua italiana | iii |
| Contents | v |
| | |
| 1 Introduction | 1 |
| 2 Objectives | 3 |
| 2.1 Classifying Market Sentiment from Social Media Posts: The Challenge of Emoji-Rich Text | 3 |
| 2.2 Predicting Financial Asset Returns Using Market Sentiment | 4 |
| 3 Literature Review | 7 |
| 3.1 Natural Language Processing Techniques | 7 |
| 3.1.1 Tokenization | 7 |
| 3.1.2 Encoding | 8 |
| 3.1.3 The Transformer Architecture: A Deep Dive | 9 |
| 3.2 Emojis in Natural Language Processing | 16 |
| 3.3 Market Sentiment in Financial Markets | 17 |
| 3.4 Deep Neural Networks For Price Forecasting | 18 |
| 4 Model Design and Methods | 21 |
| 4.1 Dataset | 21 |
| 4.1.1 Data Collection | 21 |
| 4.1.2 StockTweets Dataset | 22 |
| 4.1.3 Stocktwits-Emoji Dataset | 24 |
| 4.2 Base model: FinBERT/RobERTa | 26 |
| 4.2.1 FinBERT | 26 |

| | | |
|----------|--|-----------|
| 4.2.2 | twitter-roberta-base-sentiment-latest | 27 |
| 4.3 | exBERT | 28 |
| 4.4 | Pretraining | 30 |
| 4.4.1 | Data Pre-Processing For Masked Language Modeling | 30 |
| 4.4.2 | Pretraining | 31 |
| 4.5 | Finetuning | 32 |
| 4.5.1 | Finetuning results | 33 |
| 4.6 | Model Evaluations | 36 |
| 4.6.1 | FinBERT vs exBERT FinBERT-based | 37 |
| 4.6.2 | ROBERTA vs exBERT ROBERTA-based | 42 |
| 4.6.3 | Finbert fine-tuned | 46 |
| 4.7 | Models Comparison | 49 |
| 5 | Price Prediction Model | 55 |
| 5.1 | Price Forecasting | 55 |
| 5.2 | LSTM Architecture | 56 |
| 5.3 | Trading Strategy | 57 |
| 5.3.1 | Experiments | 58 |
| 5.3.2 | Results Evaluations | 60 |
| 6 | Conclusions and Future Work | 65 |
| 6.1 | Future Work | 66 |
| | Bibliography | 69 |
| | List of Figures | 73 |
| | List of Tables | 75 |
| | Ringraziamenti | 77 |

1 | Introduction

To fully capture financial markets dynamics has always been a complex but fascinating goal for researchers and financial institution. The task have recently become increasingly complex and influenced not only by traditional economic indicators but also by the rapid diffusion of news and information with digital communication. The advent of online platforms, such as Twitter, StockTweets, and financial forums, has created a new environment in which investors, both retail and institutional, actively share opinions, expectations, and emotions about the marker, heavily influencing its behavior. This vast amount of unstructured textual data present online provides a unique opportunity to capture the collective mood of the market and to assess whether sentiment can a reliable signal for asset price movements. Traditional financial forecasting approaches rely heavily on historical quantitative data such as returns, volumes, and volatility. While these indicators are fundamental, they do not capture the psychological and behavioral dimensions that often drive short-term price fluctuations. A clear example is the massive price surge of GameStop in early 2021, largely fueled by discussions on the Reddit forum *r/WallStreetBets*, where collective retail investor sentiment and coordinated action drove the stock far beyond levels justified by traditional financial fundamentals. Behavioral finance has long emphasized the role of investor sentiment in shaping market outcomes. Thanks to the recent rise of have Natural Language Processing, nowadays why have tools and methods to make it feasible to systematically extract sentiment from large corpus of text and incorporate it into predictive models. The objective of this thesis is to investigate the contribution of sentiment information to financial time-series forecasting, with a particular focus on extracting sentiment from unstructured, colloquial and emoji-rich online content. To this end, we develop and evaluate sentiment classification models specifically adapted to financial texts enriched with emojis, which are increasingly common in online investor discussions. The extracted sentiment signals are then integrated into a price prediction model based on a Long Short-Term Memory (LSTM) architecture, a deep learning method well suited for time series forecasting. By comparing models trained with and without sentiment features, and by assessing how variations in sentiment prediction quality affect price forecasting, we aim to quantify the added value that sentiment analysis

provides in anticipating future market movements. The research also explores the effect of sentiment model quality on downstream forecasting accuracy. Specifically, we compare baseline language models with extended architectures that include enriched vocabularies for financial and emoji-rich content. This allows us to assess not only whether sentiment matters, but also whether improvements in sentiment classification accuracy translate into measurable gains in financial prediction tasks. Beyond methodological contributions, this study has practical implications for algorithmic trading. If sentiment proves to be a reliable feature for price prediction, it could be integrated into automated trading systems to enhance decision-making and improve portfolio performance. At the same time, understanding the limitations of sentiment-based models is equally important, as financial markets are influenced by a wide range of factors, many of which remain unpredictable. The remainder of the thesis is structured as follows: Chapter 2 delivers more clearly the objectives of this research, Chapter 3 provides a review of the relevant literature on financial forecasting, sentiment analysis, and deep learning methods. Chapter 4 describes the development of Large Language Model based sentiment classification methods and their evaluation. Chapter 5 presents the integration of sentiment features into the LSTM-based price prediction framework. Finally, Chapter 6 concludes the thesis by summarizing the main findings and outlining directions for future works.

2 | Objectives

2.1. Classifying Market Sentiment from Social Media Posts: The Challenge of Emoji-Rich Text

The primary objective of this thesis is to extract **market sentiment related to a financial asset** (e.g., stocks, cryptocurrencies) from online and social media sources, with a focus on **emoji-rich text**. Once extracted, this sentiment is quantified and used as a novel input feature in predictive models for forecasting future asset prices and returns. This research adopts a large language model (LLM)-based approach for sentiment classification, addressing the complex challenge posed by the subtle interpretation of emojis and informal textual content rich of irony and sarcasm as well as typos and slang expressions.

In today's web-centric world, financial news and opinions are generated and disseminated at high frequency across diverse platforms. These include traditional sources such as financial institutions, government agencies and newspapers, as well as an always increasing number of social media communities, forums, and retail investor groups. Online posts, differently from headlines, often contain slang, abbreviations and emojis, making them difficult to interpret with conventional **Natural Language Processing (NLP)** tools.

By compounding content from these heterogeneous sources we have an overview over market sentiment surrounding an asset. Capturing and understanding the sentiment is potentially valuable, as investor expectations and emotions are known drivers of market behavior. The ultimate aim of this work is therefore to design an effective methodology to extract, interpret, and leverage sentiment signals from online discourse in order to support **financial forecasting tasks**.

Therefore, the goal is to develop a BERT-based classifier capable of handling heterogeneous textual sources and interpreting **emojis**, which nowadays are a key driver of sentiment in social media and online vocabulary.

The classifier is designed to assign each post to one of the following sentiment categories:

- **Bullish:** posts expressing optimism or positive expectations about an asset's future

performance (e.g., price increases, growth, strong fundamentals, trust).

- **Bearish:** posts reflecting pessimism or negative expectations regarding an asset (e.g., price drops, risk, weakness, geopolitical uncertainty).
- **Neutral:** posts that are informational or express no clear sentiment toward the asset.








| Sentence | Sentiment |
|--|-----------|
| time to start prosecuting this pyramid scheme   | Bearish |
| this ain't the stock market   | Neutral |
| we have bottomed    | Bullish |

Table 2.1: Examples of posts labeled with sentiment.

To aggregate sentiment at scale and transform it in a quantitative measure, we compute a *Sentiment Score* defined as the difference between the number of bullish and bearish posts:

$$\text{Sentiment Score} = \#\text{Bullish} - \#\text{Bearish}$$

This score serves as a synthetic indicator of market sentiment over a selected time window and can be used as a predictive feature for asset price modeling.

2.2. Predicting Financial Asset Returns Using Market Sentiment

The second core objective of this thesis is to exploit the previously computed Sentiment Score as a predictive feature in forecasting future asset prices and returns. The goal is to design an automatic trading agent that integrates traditional financial indicators, such as past prices and traded volumes, with market sentiment signals to enhance predictive accuracy and guide trading decisions.

In formal terms, at each time step t , the model aims to predict the future return or price y_{t+1} based on a set of lagged features:

$$X_t = \{P_{t-k}, \dots, P_t, V_{t-k}, \dots, V_t, S_{t-k}, \dots, S_t\}$$

where:

- P_t : asset price at time t ,
- V_t : traded volume at time t ,
- S_t : sentiment score at time t ,
- k : number of past time steps used as input (rollback window).

The predictive model is based on a **Long Short-Term Memory (LSTM)** neural network, a recurrent architecture well-suited for time-series data due to its ability to capture long-range temporal dependencies and mitigate the vanishing gradient problem. The LSTM processes sequences of the form $X_t \in \mathbb{R}^{k \times d}$, where d is the number of features. In our case three features will be used : price, volume and sentiment score.

Once the model outputs the predicted price \hat{P}_{t+1} , a simple trading logic can be applied:

- If $\hat{P}_{t+1} > P_t$: open a **long position**,
- If $\hat{P}_{t+1} < P_t$: open a **short position**.

This decision-making process forms the basis of an *end-to-end trading pipeline*, combining sentiment-aware forecasting with algorithmic execution.

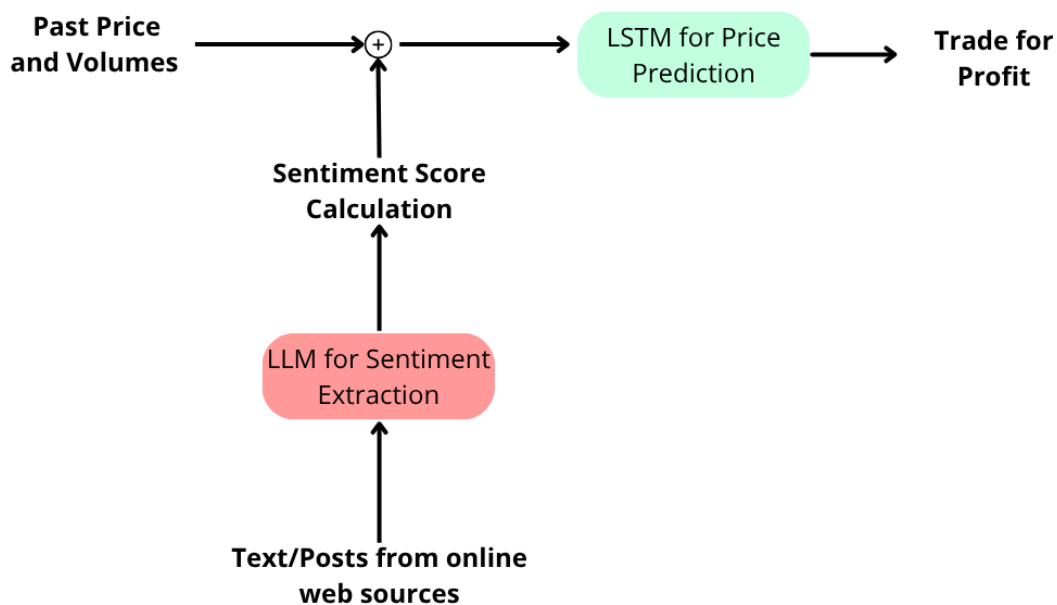


Figure 2.1: Overview of the sentiment-enhanced trading agent architecture.

3 | Literature Review

This chapter provides an overview of **Natural Language Processing** (NLP) techniques and how to use them to determine one of the key drivers of financial markets: **Market Sentiment**. Moreover, a set of publications concerning the effectiveness of analyzing market sentiment in order to forecast market behavior is collected.

3.1. Natural Language Processing Techniques

The very first step to perform sentiment analysis through a machine is to transpose words into something a machine can run computations on. In this sense the main approach is to encode words in multidimensional vectors [18]. In order to do so a textual document is split into words or sub words, commonly called **Tokens**. These tokens serve as the atomic elements upon which further linguistic processing is performed. Through a machine learning algorithm, words' representation in the target vector multidimensional space is learned. Once we have transposed tokens into vectors, we can manipulate them to perform a wide range of tasks such as text generation, translation, or sentiment analysis. The more accurate the vector encoding, the more effective downstream tasks can be.

3.1.1. Tokenization

Tokenization is the foundational pre-processing step in any NLP pipeline.

Traditional tokenization methods often rely on whitespace and punctuation to segment text into words. While simple and intuitive, these methods can be brittle in handling edge cases such as contractions (e.g., *don't* vs. *do not*), compound words, or multilingual content [18]. To improve robustness and reduce vocabulary size, modern NLP models frequently use **sub-word tokenization**. This approach segments words into smaller meaningful units (e.g., *play + ing* instead of *playing*) and is particularly useful for handling rare or out-of-vocabulary words [27]. Two widely used subword tokenization techniques are: **Byte Pair Encoding (BPE)** [27] and **WordPiece** [32]. The first was originally a data compression technique that iteratively merges the most frequent pairs of characters or

character sequences into sub-word units, balancing between full words and characters. The latter, developed for Google’s Neural Machine Translation system, uses a likelihood-based objective to determine the best sub-word splits. It is the method used in BERT and other Transformer-based models. More recently, **SentencePiece** [20] has been introduced to allow unsupervised tokenization directly on raw text without requiring whitespace-based pre-tokenization. It is particularly effective in multilingual and noisy text settings, as it treats text as a sequence of unicode characters.

Tokenization is crucial because it determines the granularity of input representation and directly impacts vocabulary size, model efficiency, and performance. Incorrect or inconsistent tokenization may introduce semantic noise, hinder model learning, or inflate computational costs.

3.1.2. Encoding

Early sentiment classification approaches relied on traditional machine learning models to encode words into vectors.

Among the most used algorithms we should mention **Support Vector Machines (SVM)** [7], often trained on text representations like **Bag-of-Words (BoW)** or **TF-IDF**. These methods represent a document as an unordered collection of words, ignoring grammar, syntax, and word order. As a result, BoW-based models fail to capture the meaning of phrases where context and word dependencies are essential, such as negation (“not good”) or sarcasm.

In the BoW model [33], a document d is represented by a vector $\mathbf{x}_d = [x_1, x_2, \dots, x_{|V|}]$, where x_i is the frequency of the word w_i in d . In the TF-IDF variant, the term weight is adjusted as:

$$\text{TF-IDF}(w_i, d) = \text{tf}(w_i, d) \cdot \log\left(\frac{N}{\text{df}(w_i)}\right)$$

where N is the total number of documents and $\text{df}(w_i)$ is the number of documents containing w_i .

To overcome these limitations, distributed representations like **Word2Vec** [23] were introduced. Word2Vec maps words to dense, fixed-size vectors that preserve semantic relationships, enabling models to capture word similarity (e.g., “king” - “man” + “woman” = “queen”). The Skip-gram model maximizes:

$$\sum_{t=1}^T \sum_{-k \leq j \leq k, j \neq 0} \log P(w_{t+j} | w_t)$$

with the probability:

$$P(w_O | w_I) = \frac{\exp(\mathbf{v}_{w_O}^\top \cdot \mathbf{v}_{w_I})}{\sum_{w=1}^{|V|} \exp(\mathbf{v}_w^\top \cdot \mathbf{v}_{w_I})}$$

Typically approximated using negative sampling:

$$\log \sigma(\mathbf{v}_{w_O}^\top \cdot \mathbf{v}_{w_I}) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-\mathbf{v}_{w_i}^\top \cdot \mathbf{v}_{w_I})]$$

However, Word2Vec embeddings are still context-independent: the word “bank” will have the same vector whether referring to a riverbank or a financial institution. For these reasons using the embedding resulting from Cbow or Word2Vec as feature to predict text sentiment at the state-of-arts is not very common.

The boundaries of research in NLP tasks were pushed forward after the introduction of the attention mechanism [31], which allowed models to dynamically focus on relevant parts of a sequence during processing. This marked a fundamental shift from fixed-size context windows (as in Word2Vec) toward models capable of learning richer contextual dependencies.

3.1.3. The Transformer Architecture: A Deep Dive

The Transformer architecture, introduced by Vaswani et al. [31], is built upon an **Encoder-Decoder** structure, and was designed for sequence-to-sequence (Seq2Seq) tasks. A sequence-to-sequence task aims at taking an input sequence of words and output another sequence of words. Encoder-Decoder were designed to translate text in one language into another language. However, after this first model was published researchers realized that both the Encoder and the Decoder could work just fine on their own. It turned out it was possible to generate text, including translations of text, with just a Decoder: these models, which form the basis for ChatGPT, are named **Decoder-Only Transformers**. Likewise, models based entirely on the encoder started to be very useful on their own and these models which form the basis for **BERT** and many other models, were named **Encoder-Only Transformers**. However, over the years, the models based on Decoder-Only Transformers, such as ChatGPT or Gemini, kind of stole the show although models like BERT utilize only the encoder. This section details the core layers of the encoder, which underpin many state-of-the-art NLP models.

Each encoder layer is composed of the following sub-layers:

- Encoding Layer

- Positional Encoding
- Multi-Head Self-Attention
- Add & Norm (Residual Connection + Layer Normalization)
- Feed Forward Network
- Add & Norm (again)

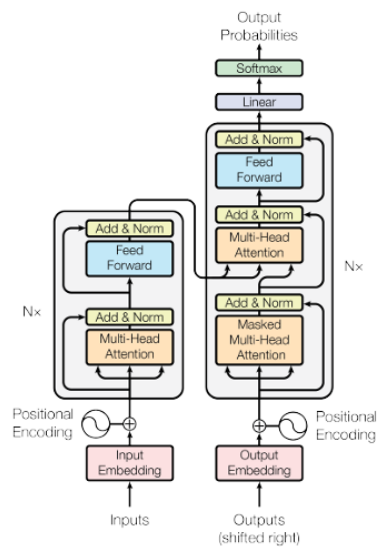


Figure 3.1: Encoder-Decoder Transformer Architecture. Vaswani et al. [31]

Input Embedding

Each token is first mapped to a dense vector using a learned embedding matrix. Similarly to other sequence embedding models mentioned above, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model} . This is the first set of weights of the Transformer, and the embedding matrix is learned using the backpropagation algorithm [26], which is standard in neural network training.

Positional Encoding

In order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add *positional encoding* to the input embeddings at the bottoms of the encoder stacks. The positional encoding have the same dimension d_{model} as the embeddings, so that the two can be summed.

There are many choices of positional encoding, both learned and fixed [15]. In literature, sine and cosine functions of different frequencies are used:

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

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than those encountered during training.

Multi-Head Self-Attention

The original idea of attention was to add a bunch of new paths from the encoder to the decoder, one per input value, so that each step of the decoder can directly access input values. Since modern transformers only have an encoder or a decoder, a mechanism similar to Attention, called Self-Attention, is used. In general terms, Self-Attention works by seeing how *"similar"* each word is to all of the words in the input sentence, including itself. If we take the sentence *"The pizza came out of the oven and it tasted good"*, the pronoun *"it"* could be referring to both the word *"pizza"* or the word *"oven"*. The Self-Attention mechanism enables the Transformer to correctly associate the word *"pizza"* with the word *"it"*. After the positional encoding layer a self-attention layer is added: each embedded vector is mapped, through a linear projection, onto three new vectors called **Query(Q)**, **Key(K)** and **Value(V)** of the same size of the original embedding vector. Afterward the Query value of each word is used to calculate the similarity with each Key value of all the word in the sentence. To calculate the similarity score between vectors exists two main approaches. The dot product between two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ is defined as:

$$\text{DotProduct}(\mathbf{a}, \mathbf{b}) = \mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i$$

The way to calculate similarity is the cosine similarity that normalizes the dot product by the magnitudes of the vectors:

$$\text{CosineSim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n a_i^2} \cdot \sqrt{\sum_{i=1}^n b_i^2}}$$

While the dot product is influenced by both the direction and magnitude of the vectors, cosine similarity measures only the angle, or the orientation, between the two vectors. This makes cosine similarity particularly useful in comparing semantic similarity between text

embeddings, where the scale of the vector is less important than its direction, therefore the latter is used for Self-Attention's similarity score calculation. Now that we have similarities scores we would like that a word which is more similar to another word has more impact on its embedding than a word less similar. For this purpose, similarities scores are run through a softmax layer. This layer outputs the percentage of each input word that is used to encode the queried word. At this point the previously calculated Value vector is used to compute the final embedding for the word: each Value, including the Value vector of the queried word itself, contributes to the final representation based on the percentage output by the softmax function. In this way each Value vector of the input sentence contributes, with its weighted percentage, to the final representation of the queried word. The main advantages of Self-Attention are that we use the same set of weights to calculate Queries, Keys and Values for each input word, regardless of how many words are in the sentence. This allows the model to calculate the Queries, Keys and Values for each word at the same time. In other words we don't have to calculate the Query, Key and Value for the first word first, before moving on the second word and because we can run all the computation at the same time, Transformers can take advantage of parallel computing and run fast.

The new embedding coming from the Self-Attention layer, which is again of the same dimension of the vector coming out the positional embedding layer, contains input from all other words: this helps give each word context and establish how each word in the input is related to others. Moreover if we think of this unit with its weights for calculating Queries, Keys and Values as Self-Attention cell then in order to correctly establish how words are related in complicated sentences and paragraphs we can create a stack of Self-Attention cells, each with its own set of weights, that we apply to the position encoded values for each word, to capture different relationships among the words. In the manuscript that first described Transformers [31], they stacked 8 self attention cells, and called this **Multi-Head attention**. Each head has dimension:

$$\frac{\mathbf{d}_{\text{model}}}{\text{number of cells}}$$

Self-attention enables each token to attend to all other tokens in the sequence. More formally, each attention head computes:

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

where Q , K , and V are learned linear projections of the input.

The Multi-Head mechanism allows the model to jointly attend to information from different representation subspaces:

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

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

The projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W^O \in \mathbb{R}^{h \cdot d_v \times d_{\text{model}}}$. These set of weights, used to create Q, K and V are learned during training.

In Vaswani et al. [31], $h = 8$ parallel attention layers where employed, meaning they used eight *heads*. For each of these, the dimension of each head was $d_k = d_v = \frac{d_{\text{model}}}{h} = 64$. Due to the reduced dimensionality of each head, the total computational cost remains similar to that of single-head attention with the full model dimensionality.

Residual Connections and Layer Normalization

After the Self-Attention layer the values coming from the positional encoding layer and the Self-Attention layer are added together. These bypasses are called **Residual Connections** and they make it easier to train complex neural networks by allowing the self-attention layer to establish relationships among the input words without also having to preserve the original word embedding and position encoding information. The use of residual connections is a standard for all deep neural networks design ever since [16] showed how it helps to address the *degradation problem*, where adding more layers to a deep network leads to higher training error. By allowing gradients to flow more directly through the network, residual connections improve convergence and make optimization more stable, even in very deep architectures. This mechanism enables the model to learn identity mappings more easily when deeper transformations are not needed. In addition, each sub-layer in the Transformer (such as self-attention and feed-forward layers) is followed by **Layer Normalization** [2], which normalizes the output across the hidden dimension for each token independently. This helps stabilize training by reducing internal covariate shift and ensures consistent scaling of the input to each layer, improving training speed and model generalization.

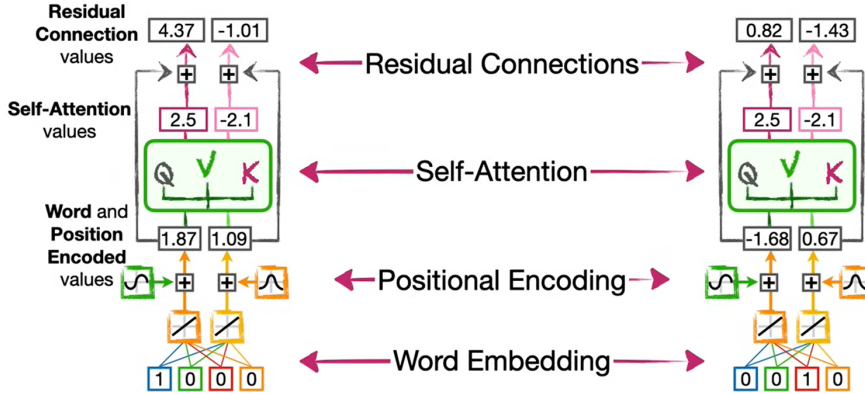


Figure 3.2: Overview of the sentiment-enhanced trading agent architecture.

That is all is needed by a Transformer to encode the input: word embedding, positional encoding, self-attention and residual connections. These four features allow the Transformer to encode words into numbers, encode the positions of the words, encode the relationships among the words and relatively easily and quickly train in parallel. Combining all four layers creates a new kind of embedding for each token that takes position and relationships among words into account. This new type of embedding is sometimes called **Context Aware Embedding** or **Contextualized Embedding** [21]. Context Aware Embedding can be finally used to perform downstream tasks such as clustering text or document or for text sentiment classification.

Feedforward Layers

On top of the sub-layer composed by Self-Attention plus Residual Connections and Layer Normalization, is placed a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. These two new sets of weights are also learned during training

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

BERT: Bidirectional Encoder Representations from Transformers

If we consider all the encoder's layers as an atomic unit, we can stack multiple units in series and build a Deep Neural Network, consisting of any number N of Encoder layers. The model is composed of N identical layers. The final output encodes rich, contextualized representations of input tokens. Indeed, BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described above [9]. Contextualized representation can be used for a wide variety of down-stream tasks:

- Natural Language Inference (NLI): is a task where the model determines the logical relationship between two sentences.
- Question Answering (QA): is a task where given a context passage and a question, the model extract the correct answer span from the passage
- Named Entity Recognition (NER): is a task where the model Assign a label to each token in a sentence (e.g., person, location, organization)
- **Sentence Classification or Text Classification:** a task where the model assign a label to a chunk of text or an entire document.

Many of the aforementioned tasks have as input a pair of sentences, logically separated and are based on understanding the relationship between two sentences. To capture contextualized meaning of a sentence (or a couple of sentences) the first token of every sequence is always a special classification token [**CLS**]. The final representation coming out all the Encoder layers corresponding to this token is used as the aggregate sequence representation for classification tasks. Sentence pairs are packed together into a single sequence. We differentiate the sentences in two ways. First, we separate them with a special token ([SEP]). Second, we add a learned embedding to every token indicating whether it belongs to sentence A or sentence B. The model is trained in two different steps **Pre-Training** and **Finetuning**. The Pre-Training phase, performed over a unlabeled dataset, is independent from the downstream task we want to perform and it is the step in which the model learns how to correctly produce the contextualized embedding. Since Pre-Training is the common denominator among all the classification task performed by BERT, to train the model two "general tasks" are used: **Masked Language Modeling (MLM)** and **Next Sentence Prediction(NSP)**. The first simply masks some percentage of the input tokens at random, and then predict those masked tokens. In order to do so, each token in a sentence is replaced with the [**MASK**] token with probability of 15% and the final hidden vectors corresponding to the mask tokens are fed into an output softmax over

the vocabulary to predict the masked word. The second is performed since many important downstream tasks such as QA and NLI are based on understanding the relationship between two sentences, which is not directly captured by language modeling. Specifically, when choosing the sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext)

During the Fine-Tuning phase the model is trained for the specific task we want to perform. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. At the output, the token representations are fed into an output layer for token level tasks, such as sequence tagging or question answering, and the [CLS] representation is fed into an output layer for classification, such as entailment or sentiment analysis. Compared to pre-training, fine-tuning is relatively unexpensive.

3.2. Emojis in Natural Language Processing

Nowadays, online content, especially coming from social media platforms, is rich of diverse and heterogeneous data. By analyzing the main platforms such as Twitter or Instagram, it is clear that contents with short text messages including hashtags and emojis [10] are dominant. Nearly half of the content on Instagram includes emojis [10], and Facebook sees the use of 5 billion emojis every day. The first usage of emojis in online text was in 1997, in Japan, as it was the first attempt to carry emotions through nonverbal cues. Currently, emojis serve as intentional carriers of emotional states and are widely used in online communications. Approximately 20 billion tweets are posted daily on platforms such as Twitter, and with each new Unicode version, new emojis are introduced, making them increasingly relevant to sentiment analysis tasks. [14]. Understanding their meaning and their interaction with traditional written form (i.e. words and punctuation) is a must to comprehend the actual sentiment of a post, since an emoji could reinforce the emotional state of author's text or it could completely revert the meaning of a sentence by introducing sarcasm or irony. Emoji-based sentiment analysis methods are divided into three main categories: dictionary-based methods, machine learning-based methods, and deep learning-based methods[29]. Dictionary-based methods focus on building emoji sentiment dictionaries to support text sentiment analysis tasks. The first attempt was to create a first emoji sentiment dictionary [19]. Then, researchers employed the skip-gram neural embedding model introduced by Mikolov to address the problem [3]. Subsequently,

the Emoji2Vec [13] pretrained embedding method was developed. In the paper, embeddings for emoji Unicode symbols were learned from their textual description. The model was trained using a simple method: for every training example consisting of an emoji and a sequence of words w_1, \dots, w_N describing that emoji, the sum of the individual word vectors in the descriptive phrase as found in the Google News word2vec embeddings was computed. Then, a trainable vector \mathbf{x}_i is defined for every emoji in the training set, and they modeled the probability of a match between the emoji representation \mathbf{x}_i and its description representation \mathbf{v}_j using the sigmoid of the dot product of the two representations:

$$P(\text{match}) = \sigma(\mathbf{x}_i^\top \mathbf{v}_j)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function, \mathbf{x}_i is the trainable embedding for emoji i , and \mathbf{v}_j is the summed word2vec representation of its description.

The performances of these methods were outperformed by the most recent **deep learning techniques**. Examples include models such as BiLSTM, deep neural networks, symbiotic graph networks and recurrent neural network models. Finally, the modern approach with attention-based Transformers model such as BERT and GPT [22] has the best performance on the task. Unfortunately, open-source BERT pre-trained models for classification task that can be found online and downloaded (i.e. from HuggingFace) don't have both words and emojis in their vocabulary. This means that when the Tokenizer finds a token corresponding to an emoji, instead of feeding the network with the correct embedding for that emoji, it will replace it with [UNK] token (which has its own embedding) and therefore the models can not distinguish between different emojis.

3.3. Market Sentiment in Financial Markets

Sentiment analysis is a subfield of natural language processing (NLP) that aims to determine the emotional tone of text and classify it as *positive*, *neutral*, or *negative*. Social media platforms are rich sources of such textual data, and in recent years, many studies have leveraged these sources to understand and predict financial behavior.

In the literature we find popular traditional methods to stock price prediction that fall in two broad categories: **fundamental analysis** and **technical analysis**. Fundamental analysis is a technique used to assess or predict the price of a stock using publicly available information such as financial statements or economic indicators [11]. On the other hand Technical analysis involves using charts, trading data such as price and volume as input features with very little subjectivity on the analysis to evaluate investment opportunities. This is done by analyzing the statistical trends of the collected data. Differently from

fundamental analysis, technical analysis aims at determining future price movement of a stock with a statistical analysis of the past price and volume data as well as *technical indicators*(e.g. RSI, MA) with the belief that historical prices tend to repeat themselves [25].

Given the hard nature of the task of predicting future behavior of financial markets, researchers have proposed a number of methods aimed at using language features to predict market variables to be used jointly with fundamental and technical analysis. A growing body of literature suggests that stock market returns are influenced not only by sentiment embedded in firm-specific news and financial reports, but also by sentiment expressed on social media platforms. These signals may reflect institutional intentions (e.g., Federal Reserve or central banks) or the positioning of retail investors. Through analyzing these text data from online news websites and other sources, Mo et al. [24] investigate the effects of sentiment on stock price returns and provide empirical evidence of a bidirectional feedback mechanism between news sentiment and U.S. financial market returns (S&P 500, NASDAQ, Dow Jones, and Russell 3000). Their findings reveal that news sentiment influences market returns with a lag of five days. Moreover the authors highlight the value of sentiment-return interdependence for predictive modeling and propose future directions including intraday analysis and the development of trading indicators based on these dynamics.

Adams et al. [1] developed the Twitter Financial Sentiment Index (TFSI), showing that Twitter-based sentiment correlates strongly with traditional market indicators and helps predict daily stock returns. Their findings emphasize that user engagement—rather than average sentiment—drives most of the variation in sentiment signals.

Other studies, such as Deveikyte et al. [8], confirm the expected positive correlation between sentiment and stock market returns, and a negative correlation with volatility. A foundational study by Tetlock [30] examined media pessimism using Wall Street Journal content and found that high pessimism contributes to downward pressure on market prices.

3.4. Deep Neural Networks For Price Forecasting

Before the advent of deep learning methods, time series forecasting was mainly based on statistical and classical machine learning approaches. The most widely used class of models were **Auto-Regressive Models** such as AR, the Moving Average (MA) model, and Auto-Regressive Integrated Moving Average (ARIMA) model [4]. This paradigms are suited for stationary data and model linear dependencies between past observations.

In an **AR(p)** model, the value of a time series at time t , denoted y_t , is modeled as a linear combination of its p previous values, plus an error term:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t,$$

where c is a constant, ϕ_i are the Auto-Regressive coefficients, and ϵ_t is a white noise. AR models are suitable when past values have a significant influence on future values.

The **MA(q)** model, on the other hand, models y_t as a function of the current and past error terms:

$$y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q},$$

where θ_j are the moving average coefficients.

The **ARIMA(p,d,q)** model combines AR and MA components and introduces differencing to handle non-stationary processes. The integer d indicates how many times the data is differentiated to bring a non-stationary process to a stationary one. The general ARIMA model can be written as:

$$\Delta^d y_t = c + \phi_1 \Delta^d y_{t-1} + \cdots + \phi_p \Delta^d y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q},$$

where $\Delta^d y_t$ represents the differenced series.

Extensions such as **SARIMA** (Seasonal ARIMA) and **ARIMAX** (ARIMA with exogenous variables) allow these models to handle seasonal patterns and incorporate external regressors, respectively. SARIMA introduces seasonal autoregressive and moving average terms, while ARIMAX includes other time series as inputs.

This family of model is able to shape only linear dependencies among data at different time steps, and linearity is not suited for financial and market data. For this reason, more recent Machine Learning techniques tried to overcome this limitation by modeling non-linear dependencies among features. Techniques such as **Support Vector Regression (SVR)** [12] and **Random Forests** [5] have been applied to time series forecasting, especially for non-linear patterns. These methods typically require handcrafted feature extraction (e.g., lag features, rolling statistics) and do not inherently model sequential dependencies. The limitations of these traditional methods in capturing complex temporal dynamics have led to the increasing adoption of deep neural networks, such as LSTMs and GRUs, which can learn such dependencies directly from raw sequences.

LSTMs [17] are powerful networks which not only have the capability of carrying information from one time step to the other but can also store or establish short-term dependencies

and translate them into long-term dependencies. They have internal mechanisms called gates that can regulate the flow of information. This information will then go through gates to inform whether it will be sufficiently important to recall or not.

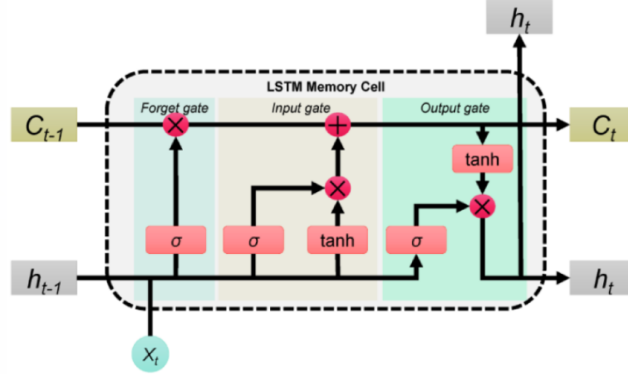


Figure 3.3: Architecture of an LSTM cell showing the input gate, forget gate, and output gate that control the flow of information.

The computations inside an LSTM cell at time step t are governed by the following equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (\text{forget gate})$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (\text{input gate})$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (\text{candidate cell state})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (\text{cell state update})$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (\text{output gate})$$

$$h_t = o_t \odot \tanh(c_t) \quad (\text{hidden state output})$$

4 | Model Design and Methods

In this chapter it will be shown how to address the above challenges.

The main idea is to leverage an open-weight BERT model, which parameters can be found and downloaded online, and **extend** it's original vocabulary in order to make it able to tokenize and embed emojis. More specifically, the new model, which will be named as **exBERT** in the following paragraphs, extends a BERT-Based model by augmenting its embeddings for the original vocabulary with new embeddings for the domain-specific vocabulary via a learned "*extension*" module. The output of the original and extension modules are combined via a trainable weighted sum operation. This methodology could be applied for every sort of new domain application, from the biomedical field to the chemical one, or any field that relies on specific and contextualized expressions and uncommon words which are used only in that domain. These words may not be included in the original BERT vocabulary or, even worst, could have different connotation and shade of meaning depending on the domain. In the specific case of this research, the new domain in which we want to operate is **short-text and emoji rich online content, concerning financial assets**. Since the main objective is to extract market sentiment from these posts, the model must be able to correctly interpreter emojis and the sentiment they carry, therefore the extension module will focus on extending the pretrained model, tokenizer and vocabulary with **emojis**. This approach will allow us to reuse the original pretrained model's weights as they are to reduce required computation and training data.

4.1. Dataset

4.1.1. Data Collection

Two datasets will be used to train and test the model: **The SPY StockTweets Dataset** and the **Stocktwits-Emoji Dataset**.

4.1.2. StockTweets Dataset

This dataset is a collection of posts from the Stocktwits website, one of the leading social platforms for investors and traders. With over 10 million users, Stocktwits provides real-time sentiment and trending insights across various financial instruments, including stocks, ETFs, and cryptocurrencies. Since Stocktwits covers a wide range of assets, the dataset used in this study has been filtered to retain only the posts related to the **SPY** ticker, which represents the SPDR S&P 500 ETF Trust—a widely traded exchange-traded fund tracking the performance of the S&P 500 index. Focusing on SPY allows the research to analyze sentiment and trends with respect to a single, high-volume asset, reducing noise and ensuring consistency in the financial context of the analysis. One of the upside of using this dataset is that users may share a message and label it as *bullish* or *bearish* (*neutral* labels are absent), so that posts are already labeled and suitable for supervised learning. In the dataset there are three data columns: **Sentence**, **DateTime**, and **Sentiment**.

| DateTime | Sentence | Sentiment |
|---------------------|--|-----------|
| 2020/03/23 07:50:31 | \$SPY how many times do I gotta tell you guys, tech hasn't even sold off yet. You really think bottom is in? 🙄 you goofballs are going to call bottom until 170 and then say you called it 🤡 | Bearish |
| 2020/08/14 12:05:17 | \$SPY lol the bears go bye bye 😊 the SPY just smoked some crack and is higher than pookie 😊 | Bullish |
| 2021/01/03 15:42:09 | \$SPY 🤡 dumb ass bears. Bulls hit flu about to get ended | Bullish |
| 2021/06/27 09:12:45 | \$SPY it's actually gonna be rigged now 😬 | Bullish |
| 2020/12/10 18:33:02 | \$SPY I am so glad we can pump the market when we have 30% unemployment and I live in a ghost town. This bubble will bust sooo hard 😊 | Bearish |
| 2021/09/04 11:55:10 | \$SPY lmao did shorts think they were getting paid today??? 🤡🤡🤡 | Bullish |
| 2022/02/28 08:21:56 | \$SPY a healthy 12\$+ gain in PH... just gave me the confidence and the certainty to go in big on puts at open. Thank you fed, cheaper puts every time I need them. 😊 | Bearish |

Table 4.1: Example posts from the StockTwits Dataset with timestamp and sentiment label.

The dataset consists of **1.419.711** labeled posts of which **210.154** contain at least **one emoji**. All posts refer to the **SPY** ticker and they cover 498 business days, spanning over a period from 23-03-2020 to 13-03-2022. The Sentiment label column is slightly unbalanced toward Bullish posts: **663.476** Bearish posts against **756.235** Bullish posts.

The dataset contains 1219 (each used at least one time) while the most used Emoji is 😊, used 48.298 times.

The most frequent sentence is the word \$SPY followed by a several emojis (338 occurrences), clearly showing how sometimes people express their sentiment just through nonverbal cues.

The longest sentence is made of 223 words (whitespace sentence breakdown) while the shortest sentence, made of just one word which is \$SPY. The average post size is 14.75

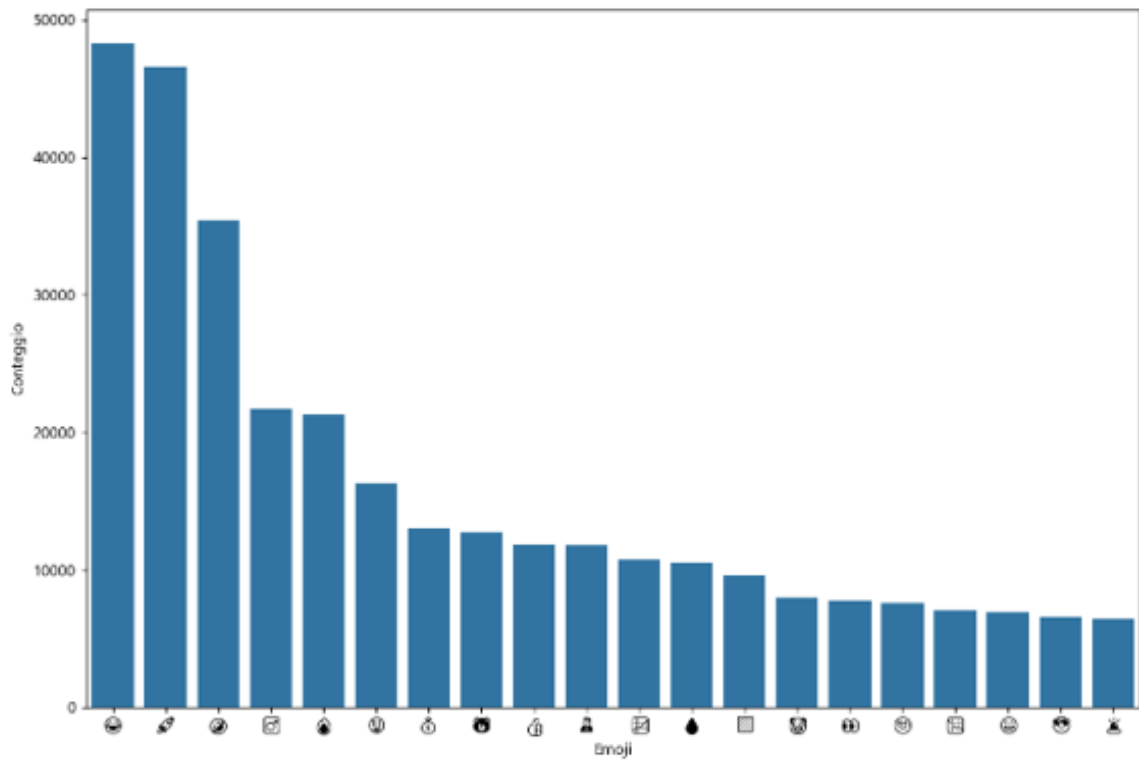


Figure 4.1: Top 10 Most Frequent Emojis Distribution

| Sentence |
|-------------------------------------|
| \$SPY 🇺🇸 📈 |
| \$SPY 🎉 🎉 🎉 🍊 🍊 🍊 🚀 🚀 🚀 🎉 🎉 🎉 🙌 🙌 🙌 |
| \$SPY 🍷 🍷 |
| \$SPY 🙄 🙄 |
| \$SPY 📈 📈 📈 📈 |
| \$SPY 📈 📈 |
| \$SPY 🎉 🎉 🎉 |
| \$SPY 😊 |
| \$SPY 🤩 🐮 🤩 🐮 🤩 🐮 |
| \$SPY 📈 📈 |
| \$SPY 🐮 🐮 |
| \$SPY 🐮 🐮 🐮 |
| \$SPY 🙄 🙄 🙄 🙄 |

Table 4.2: Example one-word posts from the StockTwits dataset.

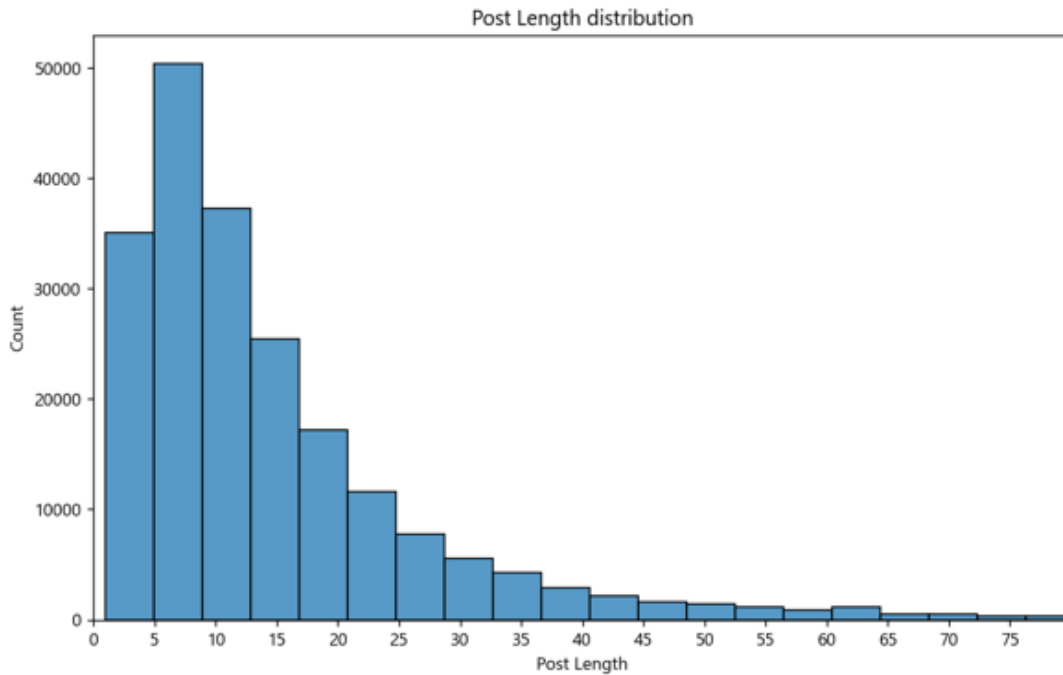


Figure 4.2: Post Length Distribution in number of words(whitespace separation)

words/post.

This dataset provides rich textual and symbolic (emoji) content for sentiment analysis.

4.1.3. Stocktwits-Emoji Dataset

This dataset is open-source and can be downloaded from HuggingFace. It contains Stock-Twits posts for Bitcoin (BTC.X), Ethereum (ETH.X) and Shiba Inu (SHIB.X). The full set ranges from 01-11-2021 to 30-04-2022, consists of 91.758 observations with three labels: Bullish, Bearish and also **Neutral**. The dataset is **strongly unbalance** and it includes 57.932 bullish, 26.516 neutral, and 7.310 bearish posts. All posts are labelled.

| Sentence | Senti-ment |
|---|------------|
| -x this weekend is going to 🔥 | Neutral |
| spot on. howto. 🍷🌱👉✅ | Neutral |
| bears have been posting the same dumb shit since the 30k's this summer. smh 🙄 | Neutral |
| let's skip the moon and go to mars 🔥🚀🚀🚀🚀 | Neutral |
| i knew it. just when i buy, it starts trending down. 😞 | Neutral |

Table 4.3: Example posts from the dataset with corresponding sentiment labels

The dataset contains 1072 (each used at least one time) while the most used Emoji is 🚀, used 29.539 times. The minimum post length of this set is four words, while the longest

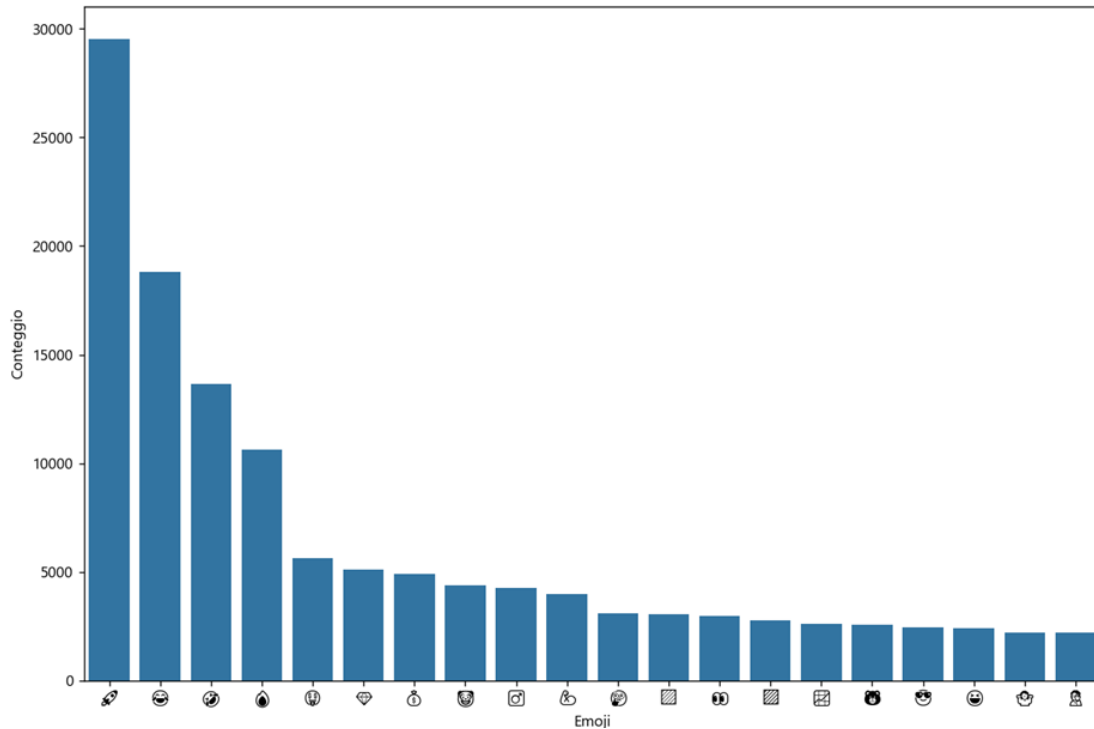


Figure 4.3: Top 10 Most Frequent Emojis Distribution

sentence is made of 199 words (whitespace sentence breakdown). The average post size is 13.31 words/post. Referring to Cao et al. before feeding the posts into the model, the following text pre-processing steps have been performed on text:

1. Remove all web links from the text, including those that start with “http://”, “https://”, or “www.”.
2. Replace the HTML character for single quotation marks with standard single quotation marks.
3. Identify and replace hashtags (#) and cashtags (\$) in the text. Find all words that start with “#” or “\$”, such as \$AAPL, and substitute them with terms like “cashtag_AAPL”. This change is made to preserve the original symbol information while making it more suitable for text processing.
4. Replace Twitter usernames with a term that begins with “mention_”, followed by the username. This modification is also intended to preserve the original information while rendering it more suitable for text processing.

Moreover, this dataset contains a **test set** that will be used later for model evaluation. The test set ranges from 16.06.2022 to 30.06.2022 and consists of 5172 bullish, 4199

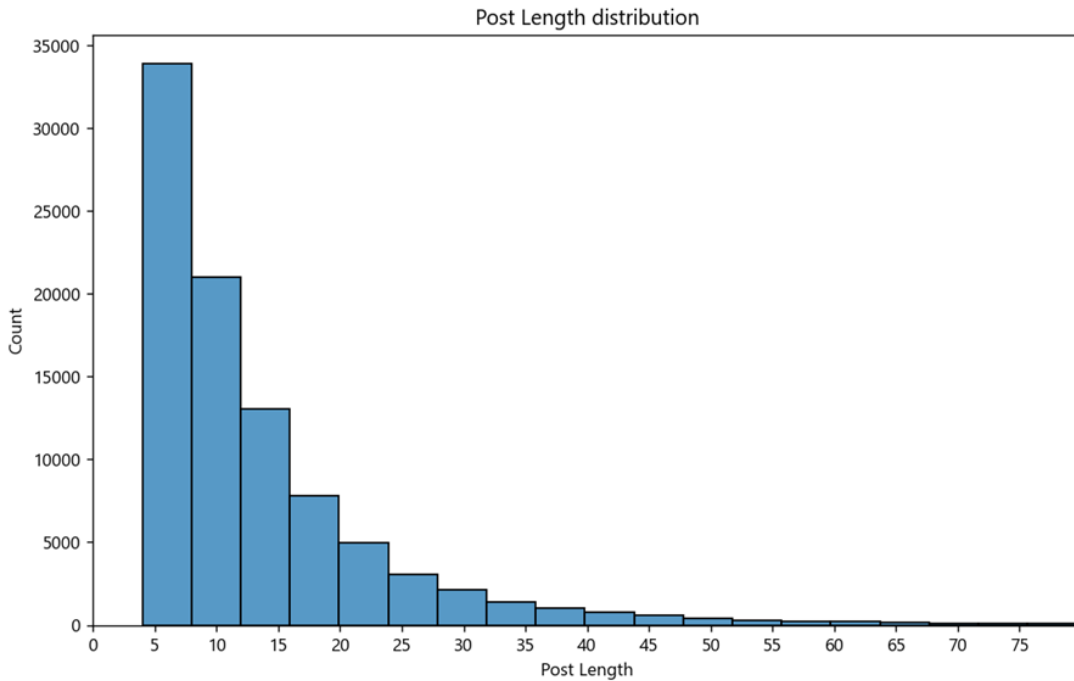


Figure 4.4: Post Length Distribution in number of words(whitespace separation)

neutral, and 2613 bearish posts, having 11,984 observations in total.

4.2. Base model: FinBERT/Roberta

For exBERT, we augment the original embedding of a BERT-based model.

For this purpose two off-the-shelf models were chosen: **ProsusAI/FinBERT** and **cardiffnlp/Twitter-roBERTa-base**.

4.2.1. FinBERT

FinBERT is a pretrained NLP model to analyze sentiment of financial text. It is built by further training the BERT language model in the finance domain, using a large financial corpus and thereby fine-tuning it for financial sentiment classification. Sentiment classification is conducted by adding a dense layer after the last hidden state of the [CLS] token. This is the recommended practice for using BERT for any classification task. Then, the classifier network is trained on a brand new labeled sentiment dataset. This is done since researches show how further pretraining a language model on a target domain corpus improves the eventual classification performance over that specific domain. However, the methodology adopted for FinBERT present some limitation for the purpose of

this research. First, the vocabulary, as well as the embedding layer, is not extended and therefore the model is **not yet able to correctly encode and weight emojis**. Second, the corpus of financial text used in the experiment is filled with technical and rigorous language expressions, which is not the case for social media posts which are rich of slang and informal vocabulary. Anyways, since our ultimate goal is to analyze sentiment in financial markets, extending a model that is already trained in that domain surely represents a better starting point than a plain BERT model that is trained for general purpose NLP tasks. Below the full configuration of this model is showed.

| Parameter | Value |
|-----------------------------------|--|
| Model Architecture | |
| Model Type | bert |
| Hidden Size | 768 |
| Intermediate Size | 3072 |
| Hidden Activation | gelu |
| Num Hidden Layers | 12 |
| Num Attention Heads | 12 |
| Max Position Embeddings | 512 |
| Type Vocab Size | 2 |
| Position Embedding Type | absolute |
| Dropout and Initialization | |
| Hidden Dropout Prob | 0.1 |
| Attention Probs Dropout Prob | 0.1 |
| Classifier Dropout | null |
| Initializer Range | 0.02 |
| Training Details | |
| Gradient Checkpointing | false |
| Use Cache | true |
| Pad Token ID | 0 |
| Layer Norm Epsilon | 1×10^{-12} |
| Torch Dtype | float32 |
| Transformers Version | 4.49.0 |
| Vocab Size | 30522 |
| Label Mapping | |
| Label2ID | {negative: 1, neutral: 2, positive: 0} |
| ID2Label | {0: positive, 1: negative, 2: neutral} |

Table 4.4: Configuration of the BERT-based sentiment analysis model.

4.2.2. twitter-roberta-base-sentiment-latest

This is a RoBERTa-base model trained on 124 million tweets from January 2018 to December 2021, and finetuned for sentiment analysis. This model specifically addresses the challenge as sentiment analysis on Twitter data among others, recognizing it as an hard task due to limited amount of contextual cues available in short texts. Starting from such a model instead of starting from the Finbert model represent a different starting point to address the problem: Finbert is trained on a financial specific vocabulary and should be able to leverage the shades of meaning of that specific domain to classify posts, while twitter-roberta is already trained on posts without a focus on the domain-specific vocabulary. For this reason twitter-roberta should be able to leverage only slang and to better

process short-text information. Below the full configuration of this model is showed.

| Parameter | Value |
|-----------------------------------|--|
| Model Architecture | |
| Model Type | roberta |
| Architecture | RobertaForSequenceClassification |
| Hidden Size | 768 |
| Intermediate Size | 3072 |
| Hidden Activation | gelu |
| Num Hidden Layers | 12 |
| Num Attention Heads | 12 |
| Max Position Embeddings | 514 |
| Type Vocab Size | 1 |
| Position Embedding Type | absolute |
| Dropout and Initialization | |
| Hidden Dropout Prob | 0.1 |
| Attention Probs Dropout Prob | 0.1 |
| Classifier Dropout | null |
| Initializer Range | 0.02 |
| Training Details | |
| Gradient Checkpointing | false |
| Use Cache | (not specified) |
| Pad Token ID | 1 |
| Layer Norm Epsilon | 1×10^{-5} |
| Torch Dtype | float32 |
| Transformers Version | 4.13.0.dev0 |
| Vocab Size | 50265 |
| Label Mapping | |
| Label2ID | {negative: 0, neutral: 1, positive: 2} |
| ID2Label | {0: negative, 1: neutral, 2: positive} |

Table 4.5: Configuration of the RoBERTa-based sentiment analysis model.

4.3. exBERT

The following section will describe exBERT implementation starting from FinBERT as base model.

For exBERT, we augment the original BERT’s embedding layer with an extension embedding layer and corresponding emojis extension vocabulary, and **add an extension module to each transformer layer**. First, we derive an extension vocabulary from our pretraining dataset, by extracting all the emojis used in the post(1219 distinct emojis) while keeping the original general vocabulary used by BERT unchanged. Since in the original Finbert vocabulary(size **30522 tokens**), there are not any emojis, we do not have the problem of having the same token repeated in the vocabulary. In standard BERT-based tokenizers, any emoji or symbol not included in the original vocabulary is replaced with the special token [UNK] (unknown token), meaning the model cannot learn a dedicated embedding for that symbol. For example, in the original FinBERT tokenizer, emojis such as 🐘, 🐻, 🔥, or 🚀 are not present in the vocabulary. Originally, emojis are mapped to [UNK] during tokenization, resulting in a loss of important semantic information. After extending the vocabulary, as done in exBERT, frequently used emojis are explicitly added

as new tokens. Consequently, during tokenization, these emojis are correctly mapped to their unique token IDs. For instance, 🐻 is tokenized as [31616] instead of [UNK], allowing the model to learn specific embeddings that capture their meaning in social media and financial discourse.

Then a corresponding embedding layer for the extension vocabulary is added, which is randomly initialized at the beginning and can be optimized during pretraining. Moreover, exBERT augments each layer of the original FinBERT model, except the classification and pooler layer, by adding an extension module to its side depicted in Figure 5.5. To combine the output from the off-the-shelf module $T_{\text{ofs}}(\cdot)$ and the extension module $T_{\text{ext}}(\cdot)$, a weighted sum mechanism is used as follows:

$$H^{(l+1)} = T_{\text{ofs}}(H^{(l)}) \cdot w(H^{(l)}) + T_{\text{ext}}(H^{(l)}) \cdot (1 - w(H^{(l)})) \quad (4.1)$$

where $H^{(l)}$ is the output of the l -th layer and w is the weighting block, a fully-connected layer of size 768×1 that outputs the weight used to compute the weighted sum of embedding vectors from the two modules. To ensure numerical stability, the output magnitudes of the weighting block is made consistent across layers by applying a sigmoid function $\sigma(\cdot)$ to constrain the output weights within the range $[0, 1]$. Specifically, the final weighting function becomes:

$$w(H^{(l)}) = \sigma(WH^{(l)} + b)$$

where $W \in \mathbb{R}^{768 \times 1}$ and $b \in \mathbb{R}$ are the learnable parameters of the fully-connected layer. The size of the extension module is flexible, provided that its output shape matches that of the off-the-shelf module. This ensures that the weighted sum in Equation (4.1) is valid and the integration between the two modules is seamless. In the end, exBERT’s architecture consist of the off the-shelf-module, which parameters have already been learned, and the new extension embedding layer, the extension module with the weighting block, which parameters need to be learned from scratch with a new pretraining phase followed by a finetuning for post sentiment classification task stage. The original model size of the base model (FinBERT) is **109.484.547 parameters** while the **new parameters** to be learned are **86.599.692**, for a **total of 196.084.239 parameters**. The size of the model is considerably big, this will require a precise managing of overfitting and powerful computational resources, with extensive memory, to fit the model itself and the data batches.

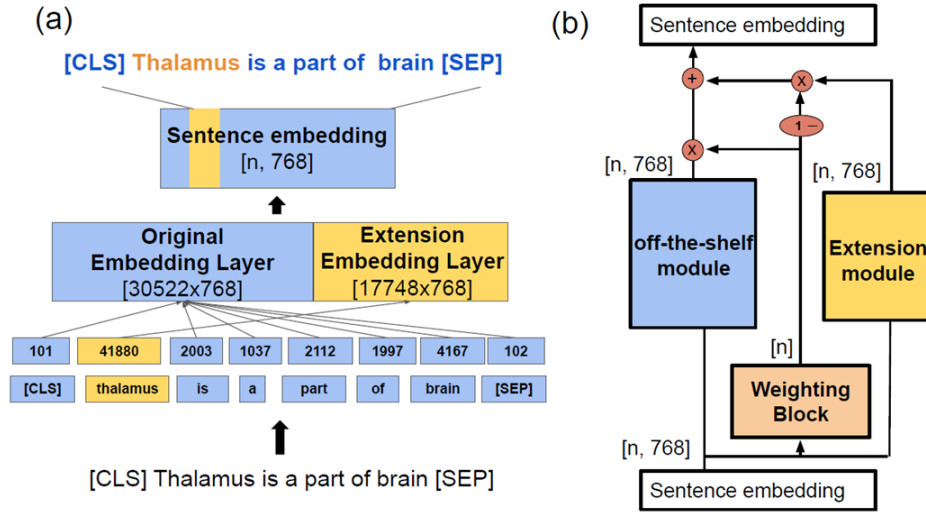


Figure 4.5: exBert model architecture Tai et al. [28]

4.4. Pretraining

Pretraining is used to allow the new model to learn *contextualized embeddings* of the new tokens (emojis) added to the extension vocabulary. During the pretraining phase, the **original BERT model remains completely frozen**, and only the **extension module** and the **weighting block** are updated.

4.4.1. Data Pre-Processing For Masked Language Modeling

To start pretraining the SPY Stocktweets Dataset, used to pretrain the model, needs to be preprocessed. The purpose of this step is to clean the input text, preserve the emojis (which are the focus of the vocabulary extension), and convert each post into a format suitable for BERT-based training. Since BERT models have a maximum input length, long posts are divided into smaller *chunks* of at most 256 characters (minus space for special tokens). Each chunk is then wrapped with the special tokens [CLS] at the beginning and [SEP] at the end, resulting in the format: [CLS] chunk of text [SEP]. These are the expected input tokens for the MLM pretraining task in BERT-like architectures. After constructing the list of formatted MLM sentences, each emojis present in the dataset is collected. This step is important to identify the emoji vocabulary used in the dataset, which will later be added to the tokenizer. The output is a list of MLM-ready sentences and a set of unique emojis used throughout the dataset. This preprocessing enables efficient training of the extension module on domain-specific emoji usage.

4.4.2. Pretraining

The training procedure uses the *Adam* optimizer with the following hyperparameters: learning rate = 1×10^{-4} , $\beta_1 = 0.9$, and $\beta_2 = 0.999$. Training is performed using 2 NVIDIA L40S (48GB GDDR6). An 80%-20% split is used for training and validation set over the pretraining dataset. The model is trained up to 20 epochs with **early stopping** technique based on the validation loss. In particular, training stops if the validation loss does not improve for 5 consecutive epochs. Finally, the model with the best validation loss is retained, to use it for further finetuning. During training, a **learning rate scheduler** is used to adjust the learning rate dynamically for each batch. In particular, the method applied is called *linear warmup with decay*, commonly used in transformer-based models like BERT. At the beginning of the training, the learning rate increases linearly from 0 to a peak value over a number of steps defined as **warm-up steps**. After this initial phase, the learning rate is decreased linearly back to zero over the remaining training steps. Formally, the learning rate at step $t \in [0, T]$ is computed as:

$$\eta_t = \begin{cases} \frac{t}{t_{\text{warmup}}} \cdot \eta_{\text{max}}, & \text{if } t \leq t_{\text{warmup}} \\ \left(1 - \frac{t-t_{\text{warmup}}}{T-t_{\text{warmup}}}\right) \cdot \eta_{\text{max}}, & \text{if } t > t_{\text{warmup}} \end{cases}$$

where:

- η_t is the learning rate at step t
- η_{max} is the peak learning rate
- t_{warmup} is the number of warm-up steps
- T is the total number of training steps

This scheduling strategy serves two main purposes: first, it stabilizes training in the early stages by avoiding large parameter updates. Second it allows **gradual convergence** by reducing the learning rate as the model approaches a minimum in the loss landscape. In the implementation, the total number of batches T is calculated as:

$$T = \left\lceil \frac{N}{B} \right\rceil$$

where N is the size of the training dataset and B is the batch size. The learning rate is updated *at every batch step*, ensuring fine-grained control over the optimization process. The **batch size** is set to 32 (because of memory capacity reasons), and the **maximum input sequence length** is 256 tokens. To conduct this new pretraining stage a custom

Masked Language Modeling task was used. Since we want to model to focus it’s learning on the new tokens, the masking probability was shaped to prioritize the learning of emoji representations, therefore significantly higher masking probability of 80% is assigned to emoji tokens compared to non-emoji tokens, which are masked at the standard 15% rate. Moreover, since the downstream of task will be classification, during pretraining NSP is not used since we will not give the model pair of sentences, but only single sentences, and therefore sentence-level relationships is not particularly relevant. The dataset used for pre-training is extracted from the **SPY StockTweets Dataset**. Each post containing an emoji is used as an observation, as input to the model that has to guess the correct masked tokens, which are mainly emojis. The configuration of the extension module is shown below.

| Parameter | Value |
|-----------------------------------|-------|
| Model Architecture | |
| Hidden Size | 768 |
| Intermediate Size | 3072 |
| Hidden Activation | gelu |
| Num Hidden Layers | 12 |
| Num Attention Heads | 12 |
| Max Position Embeddings | 256 |
| Type Vocab Size | 2 |
| Dropout and Initialization | |
| Hidden Dropout Prob | 0.1 |
| Attention Probs Dropout Prob | 0.1 |
| Initializer Range | 0.02 |
| Training Details | |
| Vocab Size | 1261 |

Table 4.6: Configuration of the extension module.

Notice that the vocabulary size of this module matches the number of tokens (emojis) we are adding to the original’s model vocabulary.

4.5. Finetuning

The following step in the training process is to finetune the model over the sentiment classification task.

For this purpose the Stocktweets Datataset from HugginFace will be used. Since the dataset is strongly unbalanced, with the majority of posts labeled as *Bullish* and a smaller proportion labeled as *Bearish* or *Neutral*. Such imbalance can bias the model toward predicting the majority class, leading to inflated accuracy but poor generalization to under-represented sentiments. To address this issue, **undersampling** will be applied, reducing the number of majority-class samples to match the minority class size. This approach balances the dataset, helping the model learn equally from each sentiment category and improving its ability to detect less frequent but potentially significant patterns. The minority class is **Bearish**, with 7.310 posts, hence 7.310 Bullish posts and 7.310 Neutral

post will be randomly sampled from the original dataset, to build a new balanced fine-tuning dataset consisting of a total of 21.930 posts. For finetuning a 90%-10% training and validation split is used. The maximum sequence length is 256 tokens, while the batch size is 32: the size of this parameter is strictly limited by the memory capacity of the GPUs used: If the total memory required to fit all the posts in one batch exceeds the GPU's available VRAM, this will lead to out-of-memory errors. The model weights are initialized starting from the best model obtained during pretraining. The optimizations leverages the *AdamW optimizer* from the popular PyTorch library. The learning rate is set to 5×10^{-5} (recommended by the literature).

As it was pointed out by Howard and Ruder (2018), catastrophic forgetting is a significant danger with this fine-tuning approach. Because the fine-tuning procedure can quickly cause model to "forget" the information from language modeling task as it tries to adapt to the new task. In order to deal with this phenomenon, we apply a commonly used technique called progressive unfreezing. With progressing unfreezing, we start training with all layers but the new embedding layer as frozen. During training we gradually unfreeze all of the layers starting from the highest one, so that the lower level features become the least fine-tuned ones. Hence, during the initial stages of training it is prevented for model to "forget" low-level language information that it learned from pre-training. In particular the model is **trained for 35 epochs whit the following unfreezing approach:**

- Epochs 0–9: only `word_embeddings_ADD` trained.
- From epoch 10: classifier and pooler layers unfrozen.
- From epoch 25: full model unfrozen.

After the completion of the finetuning phase for all the 35 epochs, since we want to select the best model and avoid overfitting, the **weights that guarantee the best validation loss are retained** and used for the evaluation step.

4.5.1. Finetuning results

Below the graphs showing the performance of the exBert FinBERT-based model during finetuning. The graph shows the evolution of training and validation loss across 35 epochs, with vertical markers highlighting key stages of the fine-tuning strategy. Plotting the two losses one against the other is useful to monitor learning stages as well as weights stability during training and it signals the raising of overfitting.

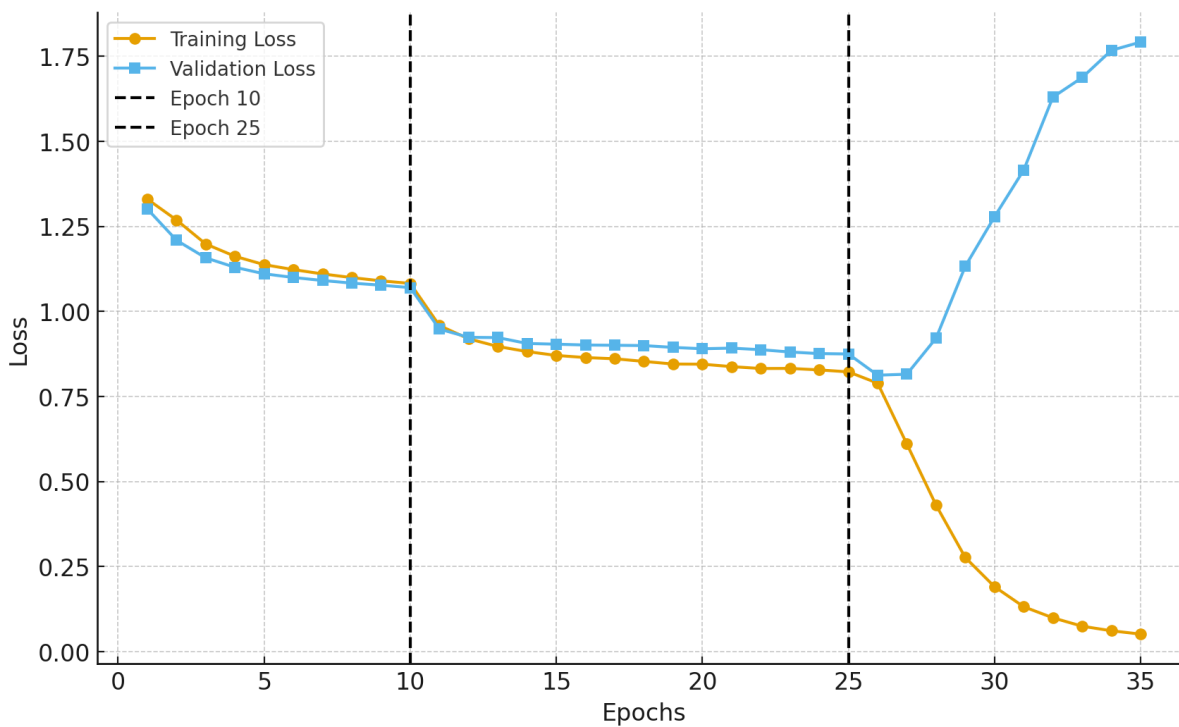


Figure 4.6: Fibert-Based exBert Training Loss vs Validation Loss

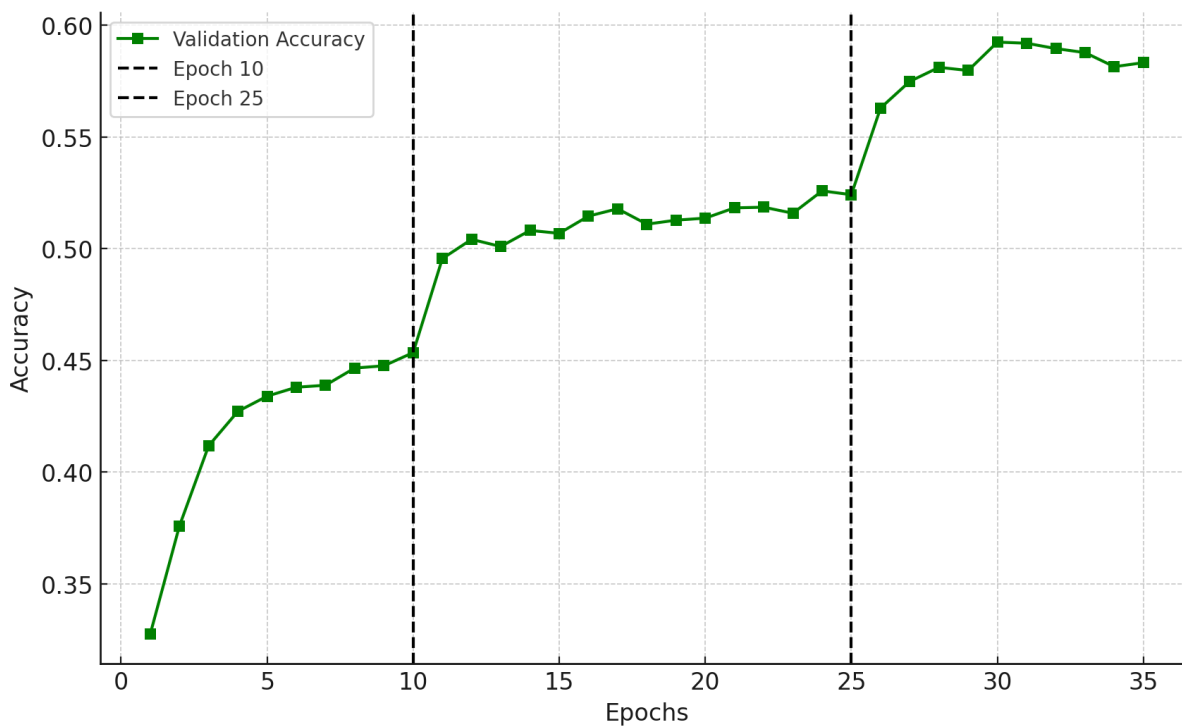


Figure 4.7: FinBERT-Based exBert Validation Accuracy

The training and validation loss graph illustrates the model's learning dynamics. Up to epoch 10, both training and validation loss decrease in parallel, suggesting effective learning without significant overfitting. Between epochs 10 and 25, losses remain closely aligned, with a gradual improvement in training performance and a relatively stable validation curve, indicating that the model maintains good generalization. After epoch 25, when the entire model is unfrozen, training loss continues to decrease sharply, while validation loss begins to rise. This divergence is a typical indicator of overfitting. The point just before epoch 26 represents the optimal bias-variance trade-off point, where validation loss is lowest and model performance is maximized before overfitting starts corrupting model's performance. We can notice a similar behavior in the Validation Accuracy graph in Fig 4.7: the best validation accuracy is reached after epoch 25, when it raises for the last time before stabilizing in a range between 55% and 60%.

As mentioned above a second model starting with a ROBERTaA-based model as the off-the-shelf model is trained with the exact same methodology used to pretrain and finetune the FinBERT-based model showed above. The graphs below shows the performance of the exBERT ROBERTaA-based model during finetuning.



Figure 4.8: ROBERTa-Based exBERT Training Loss vs Validation Loss

In this second the training and validation loss curves show a more stable trend compared to the first graph. In the initial epochs, both training and validation losses decrease grad-

ually and remain closely aligned, showing that the model is learning effectively without overfitting. A more pronounced reduction in loss is observed after epoch 10, which corresponds to the unfreezing of the classifier and pooler layers. At this stage, the model is able to leverage additional trainable parameters to refine its representations, leading to better adaptation to the dataset while maintaining stability in validation performance. Between epochs 10 and 25, the validation loss plateaus around 0.9, suggesting that the model is learning in a controlled way but not substantially improving its generalization. Meanwhile, the training loss continues to decline, signaling that the model is fitting more tightly to the training data. This divergence becomes more evident after epoch 25, when the full model is unfrozen. From this point onward, training loss decreases sharply, reaching very low levels, while validation loss begins to increase significantly. Once again, the best generalization performance is achieved around epoch 25–26, just before validation loss starts to climb.

Overall, we can affirm that both models benefit from the progressive unfreezing strategy up to around epoch 25, preventing overfitting in early stages and allowing the models to learn before the two curves start to diverge. Especially during the first epochs of training the models clearly learn how to classify posts, as it is highlighted by the decrease in validation loss and increasing validation accuracy. Unfortunately, when models' weights are fully unfrozen it happens that the training rapidly falls into overfit, signaling unstable learning probably due to the huge size of the models, both counting millions of parameters.

4.6. Model Evaluations

In this section training results will be analyzed through testing and then compared to some benchmarks in order to assess the effectiveness of the method explained above.

The main metrics that will be taken into account are the typical ones used to evaluate machine learning methods for classification: **validation accuracy, precision, recall and F1**. In particular we will use confusion matrices to evaluate performance on the test set. A **confusion matrix** is a tabular representation of the performance of a classification model. It compares the predicted labels with the true labels, showing how many instances were correctly or incorrectly classified for each class. The diagonal elements represent correct predictions, while the off-diagonal elements indicate misclassifications. For example, in a binary classification task (e.g., sentiment analysis as positive vs. negative), the confusion matrix summarizes the model's predictions as True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). This is particularly useful because it provides more detailed insights than overall accuracy alone: it highlights which types of

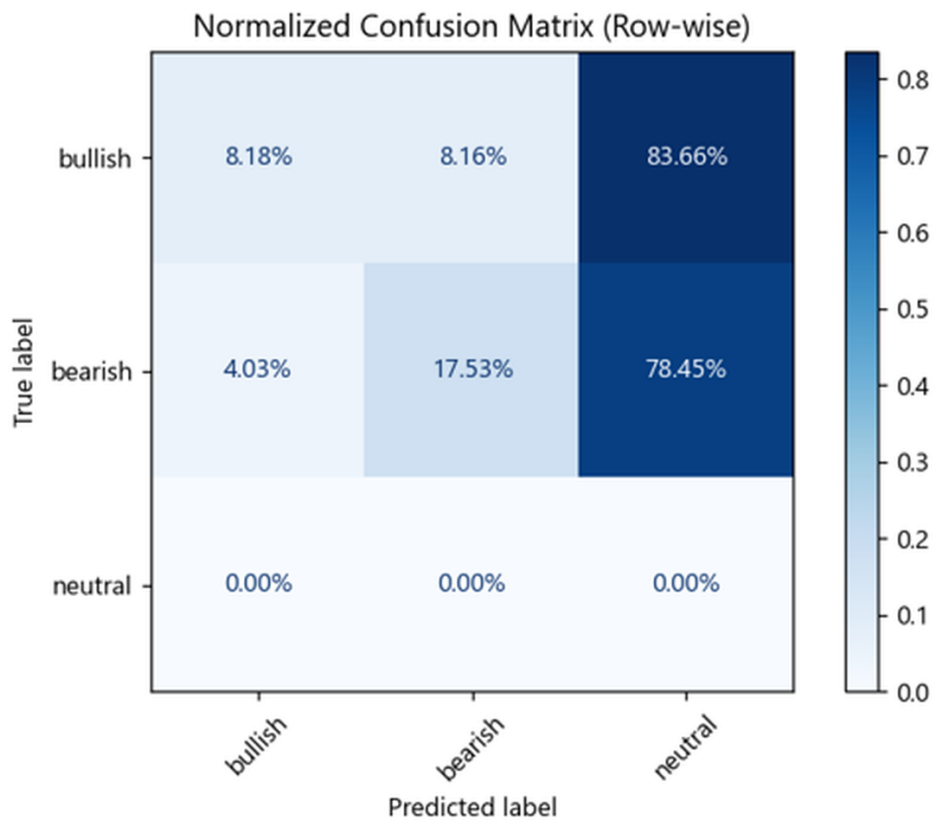
errors the model tends to make, allowing researchers to compute additional metrics such as precision, recall, and F1-score, which are crucial for evaluating model performance in imbalanced datasets.

For evaluation two test sets will be used. The first is the SPY-Stocktweets dataset, which is the same used in pretraining to extend the vocabulary. It contains **1.419.711 labeled posts** with **663.476 Bearish** posts against **756.235 Bullish** posts. This dataset was not used to train the model on the classification task, hence the model is not biased in classifying the posts contained in the dataset. Unfortunately, this dataset contains only two labels (Bullish and Bearish), while the models we trained are designed to predict three labels (Bullish, Bearish and Neutral). As a result, the evaluation metrics of the models will be strongly biased toward the worst-performing cases. Moreover, since only 210,154 posts in the dataset contain an emoji, this limits the correct interpretation of the results when assessing the model's ability to improve its predictions on emoji-rich posts. Nevertheless, the predictions obtained from this dataset will be used in the subsequent part of the research, and are therefore necessary.

For the reasons mentioned above, the second test dataset (the Stocktwits-Emoji test set) will provide more clear result since it contains all the three labels and each post has at least one emoji. The latter consists of **11,984 observations with 5172 bullish, 4199 neutral, and 2613 bearish posts**.

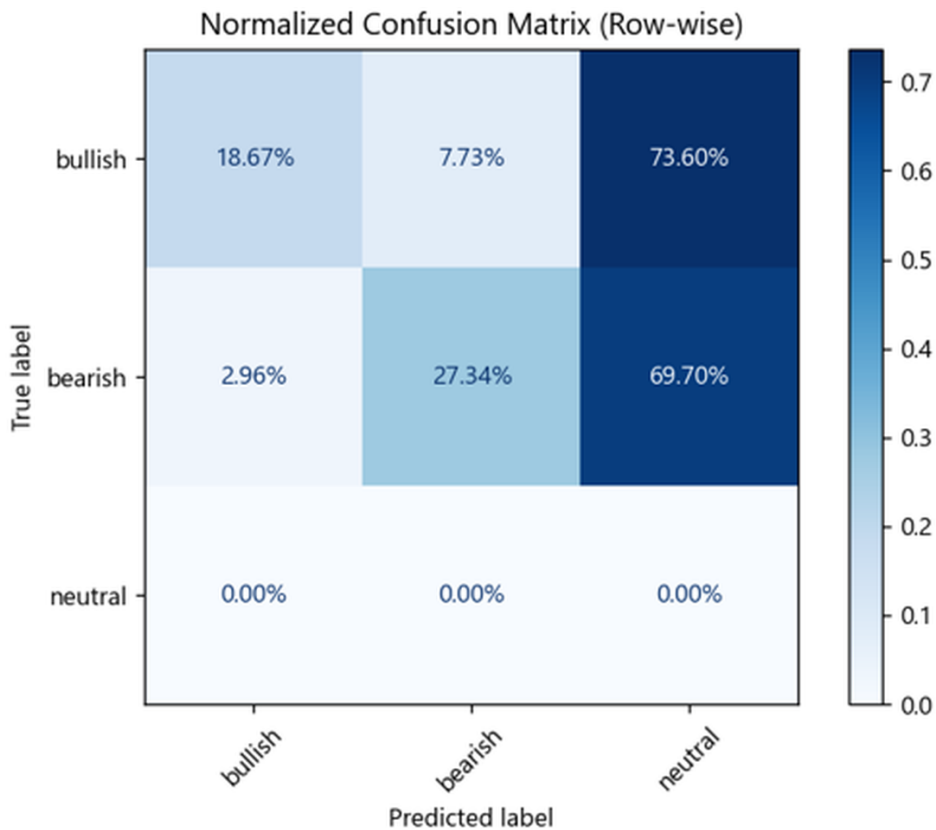
4.6.1. FinBERT vs exBert FinBERT-based

To start the analysis of the results, we first compare the performance of the **base models** with their corresponding **extended versions**. This comparison allows us to quantify the improvements obtained by incorporating the discussed model extensions technique with respect to the original architectures. In particular, we focus on the enhancement in prediction capability and overall classification accuracy.



| Metric | Value |
|-----------|--------|
| Accuracy | 0.1255 |
| Precision | 0.4505 |
| Recall | 0.0857 |
| F1-score | 0.1409 |

Figure 4.9: Confusion matrix and macro-averaged metrics for FinBERT on the SPY-StockTweets dataset.

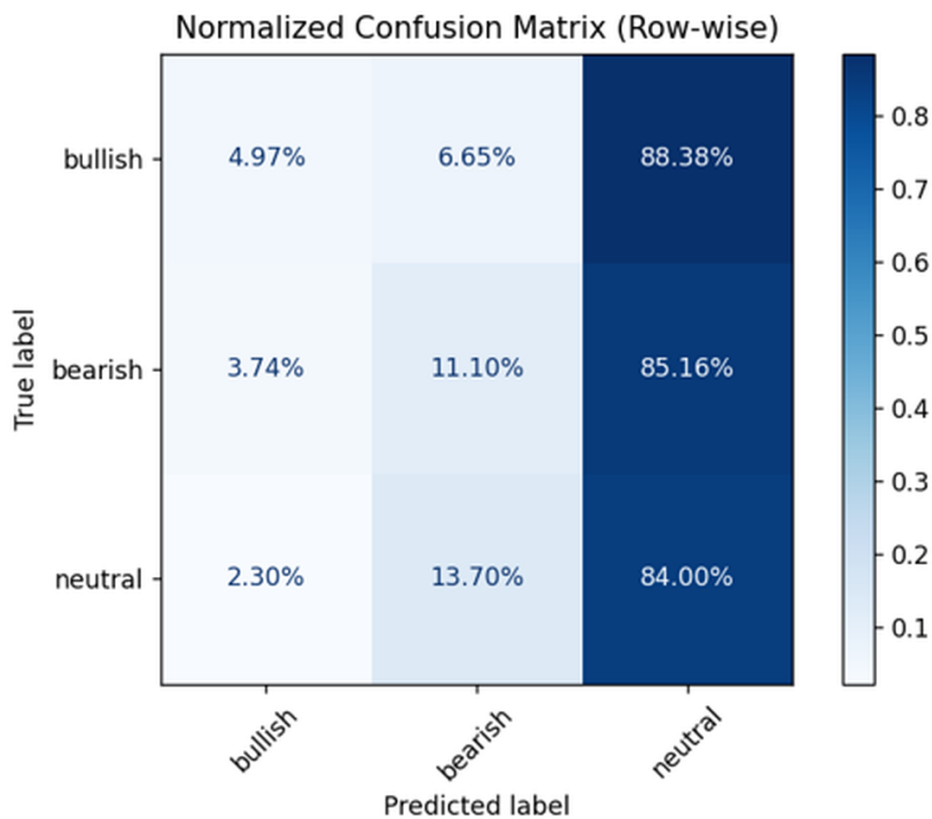


| Metric | Value |
|-----------|--------|
| Accuracy | 0.2272 |
| Precision | 0.5447 |
| Recall | 0.1534 |
| F1-score | 0.2365 |

Figure 4.10: Confusion matrix and macro-averaged metrics for exBert finbert-based on the SPY-StockTweets dataset.

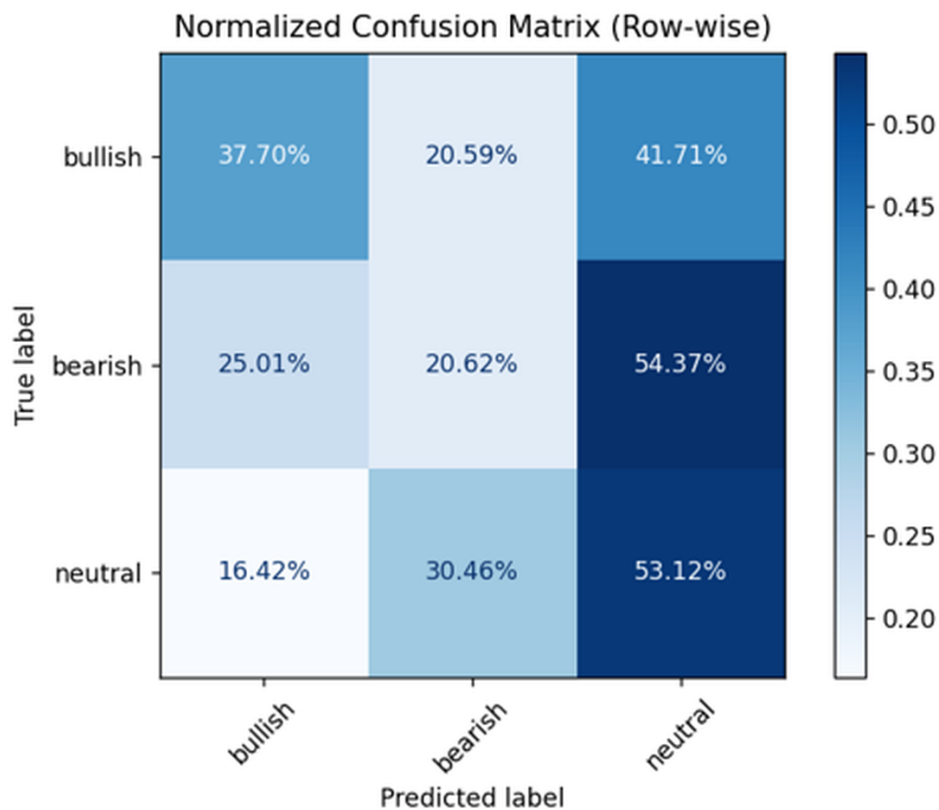
Over the first dataset, FinBERT shows very poor performance, reaching an accuracy of only 12.55%. This indicates that the base model is not well suited for the task and highlights the need for both the extension module and the extended vocabulary. Indeed, exBERT significantly improves predictive performance on this dataset, increasing accuracy by about 10% to a value of 22.72%. The confusion matrices reveal that FinBERT tends to be extremely conservative, showing a reluctance to classify posts as Bullish or Bearish and instead predicting a large proportion of Neutral labels. This behavior is par-

tially mitigated in the fine-tuned exBERT model, which is able to correctly identify more Bullish and Bearish posts. Nevertheless, as discussed above, the absence of Neutral labels in the dataset strongly affects the overall results, biasing them toward lower performance since every Neutral prediction made by the model is automatically counted as an error. For this reason, the analysis conducted on the second dataset provides a more reliable and informative evaluation of model performance.



| Metric | Value |
|-----------|--------|
| Accuracy | 0.2435 |
| Precision | 0.3845 |
| Recall | 0.3336 |
| F1-score | 0.2017 |

Figure 4.11: Confusion matrix and macro-averaged metrics for Finbert on the Stock-Tweets-Emoji test set.



| Metric | Value |
|-----------|--------|
| Accuracy | 0.3508 |
| Precision | 0.3748 |
| Recall | 0.3715 |
| F1-score | 0.3441 |

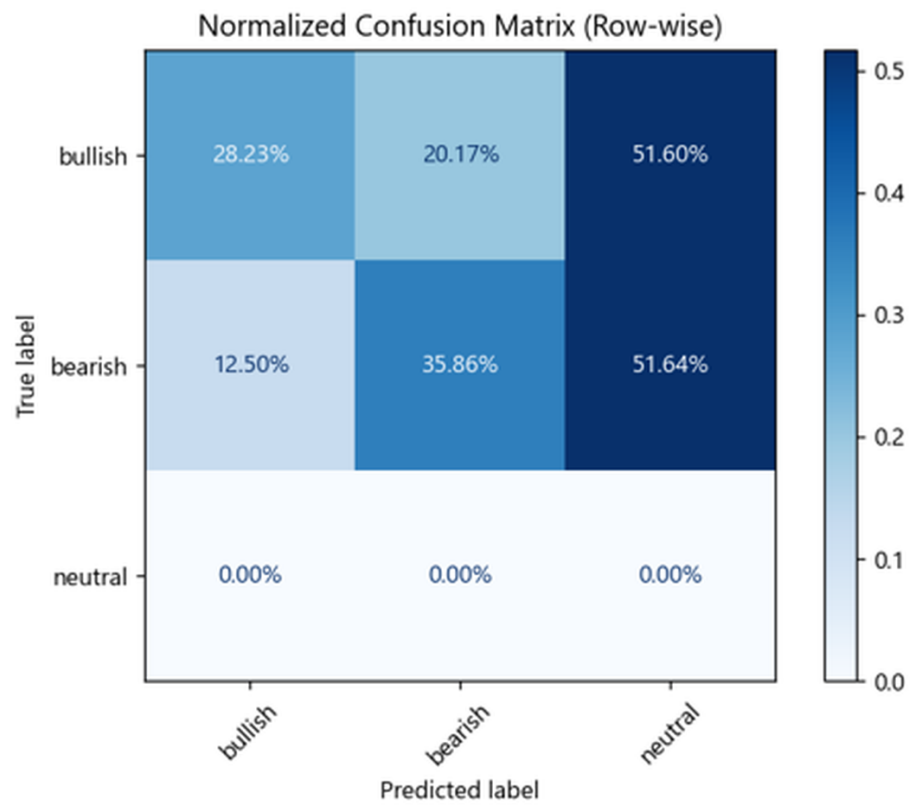
Figure 4.12: Confusion matrix and macro-averaged metrics for exBert FinBERT-based on the StockTweets–Emoji test set.

Once again, the first confusion matrix, corresponding to FinBERT, reveals the model’s strong bias toward predicting the Neutral class. This behavior results in low overall accuracy of about 24%, with the vast majority of predictions concentrated in a single class. The second confusion matrix, corresponding to exBERT, demonstrates a significant improvement: accuracy rises to approximately 35%, which corresponds to an improvement of about 10 percentage points compared to FinBERT. Unlike the base model, exBERT distributes its predictions more evenly across all three classes: it correctly identifies about

38% of Bearish posts, 20% of Bullish posts, and over 53% of Neutral posts. This indicates a substantial gain in recall for the Bullish and Bearish categories, reducing the strong Neutral bias that characterized FinBERT.

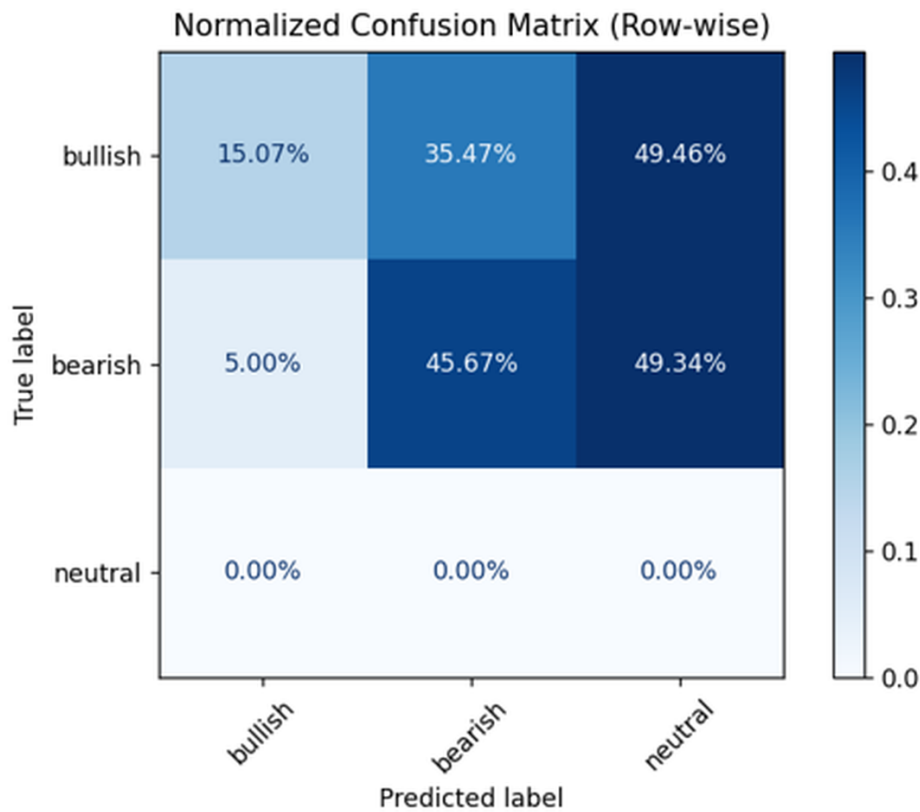
4.6.2. ROBERTA vs exBert ROBERTA-based

Since Finbert performances are very low, we try to change the base model to start from a better results and further improve the off-the-shelf model prediction capability. Since Finbert, which is trained over financial text corpus, doesn't perform well on short social media emoji-rich text, we can expect that the second base model, twitter-roberta-base-sentiment which already trained to classify social media posts, could perform better over the two datasets before extending its vocabulary.



| Metric | Value |
|-----------|--------|
| Accuracy | 0.3179 |
| Precision | 0.4432 |
| Recall | 0.2136 |
| F1-score | 0.2857 |

Figure 4.13: Confusion matrix and macro-averaged metrics for ROBERTA on the SPY-StockTweets dataset.

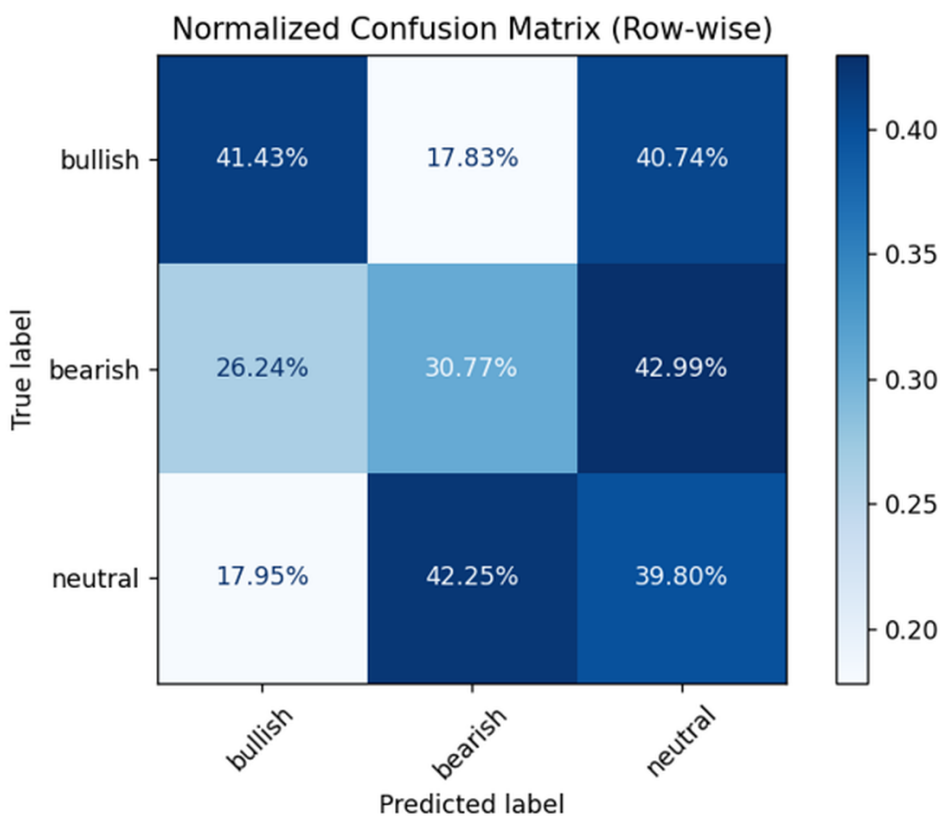


| Metric | Value |
|-----------|--------|
| Accuracy | 0.2937 |
| Precision | 0.4350 |
| Recall | 0.2024 |
| F1-score | 0.2477 |

Figure 4.14: Confusion matrix and macro-averaged metrics for exBERT ROBERTA-Based on the SPY-StockTweets dataset.

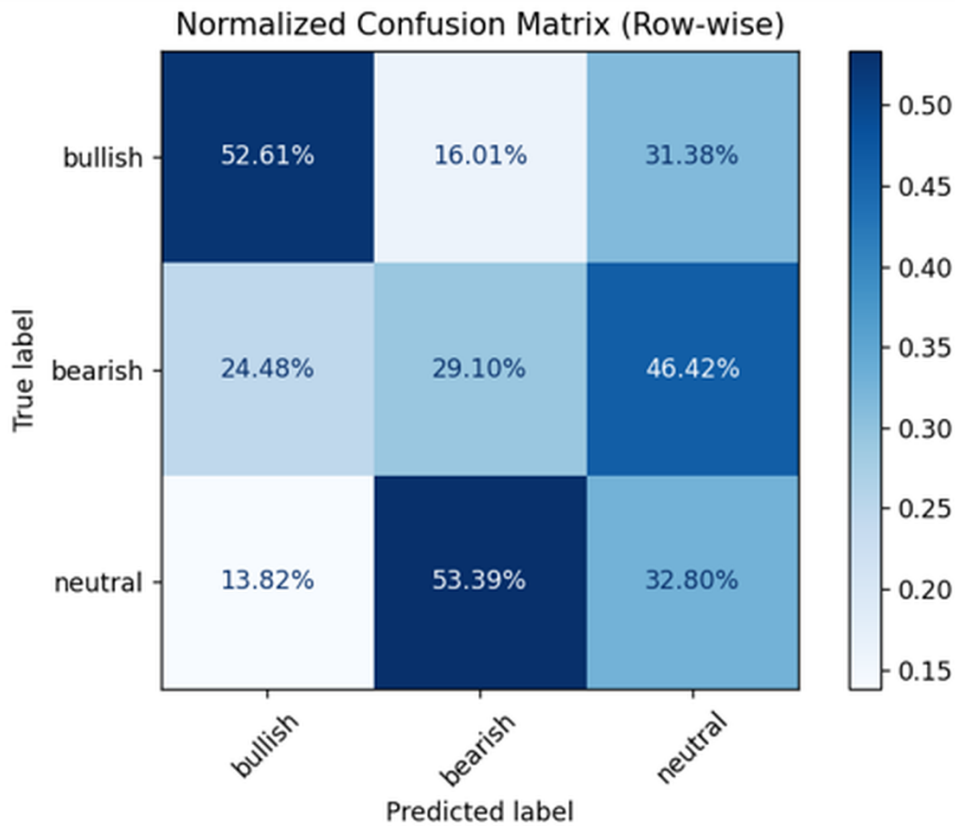
The first confusion matrix, corresponding to RoBERTa, shows that the model has better performances compared to FinBERT (Fig 4.9, with an accuracy of about 29% compared to FinBERT's 12%). This base model grants a better starting point and might allow us to end up with better overall accuracy after the extension since ROBERTa already performs well even before it. Nevertheless, the complete absence of Neutral ground-truth examples in the dataset further amplifies the errors, as every Neutral prediction is inevitably counted as an error. This again results in a strong bias toward lower accuracy.

Unfortunately, the second confusion matrix, corresponding to ROexBERT, doesn't displays a notable improvement in performance. Actually, correct classification rates decreases for Bullish posts but raises for Bearish posts. The two changes compensate one another and Compared ROexBERT doesn't gain in overall accuracy compared to ROBERTa. In this case ROexBERT doesn't outperform ROBERTa, it achieves similar performances in all statistics.



| Metric | Value |
|-----------|--------|
| Accuracy | 0.3734 |
| Precision | 0.3921 |
| Recall | 0.3733 |
| F1-score | 0.3670 |

Figure 4.15: Confusion matrix and macro-averaged metrics for ROBERTa on the Stocktweet-Emoji dataset.



| Metric | Value |
|-----------|--------|
| Accuracy | 0.4505 |
| Precision | 0.4034 |
| Recall | 0.3817 |
| F1-score | 0.3831 |

Figure 4.16: Confusion matrix and macro-averaged metrics for exBERT ROBERTA-Based on the Stocktweet-Emoji dataset.

In the second dataset where also Neutral labels are present both the base model and the extended version achieve the best performances. ROBERTA has an overall accuracy of 37% while the model with the extension module improves its performances up to 45%.

4.6.3. Finbert fine-tuned

To finish off the analysis one last model is trained by simply finetuning the FinBERT model without adding the extension module and the extended vocabulary. This finetuned

version is obtained by simply taking the original model weights and further training the model over the same finetuning dataset used to finetune the extended models. With this approach we want to evaluate the effectiveness of the extension module and if recognizing emojis is useful to correctly classify the sentiment.

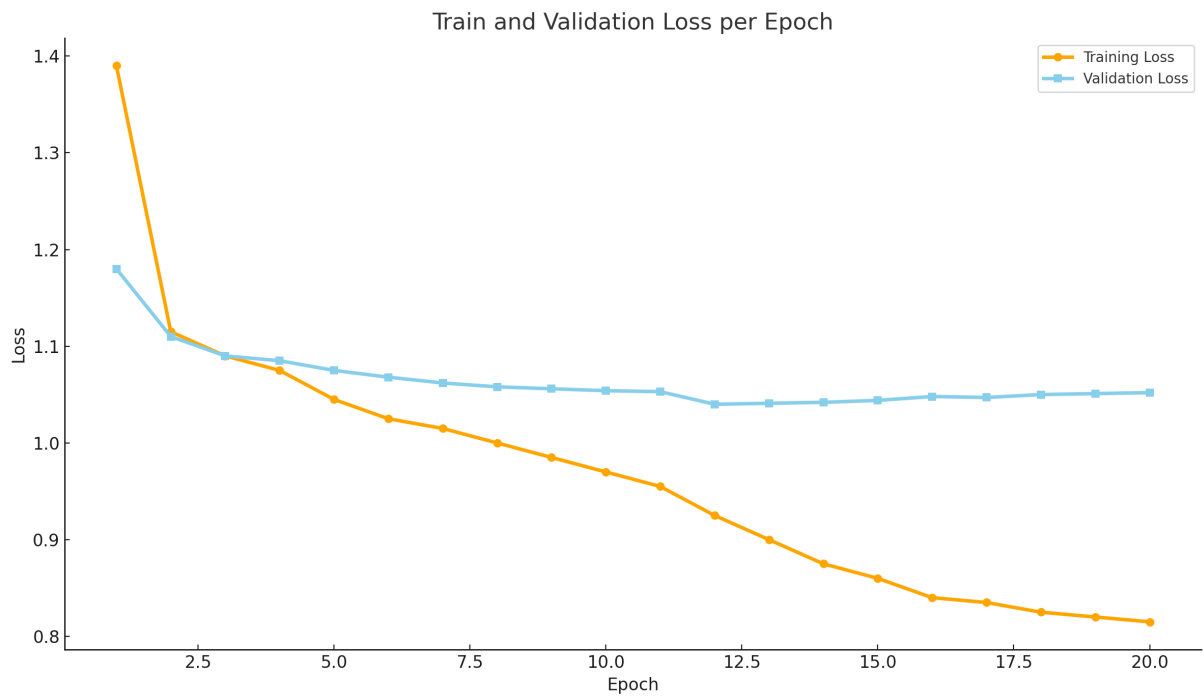
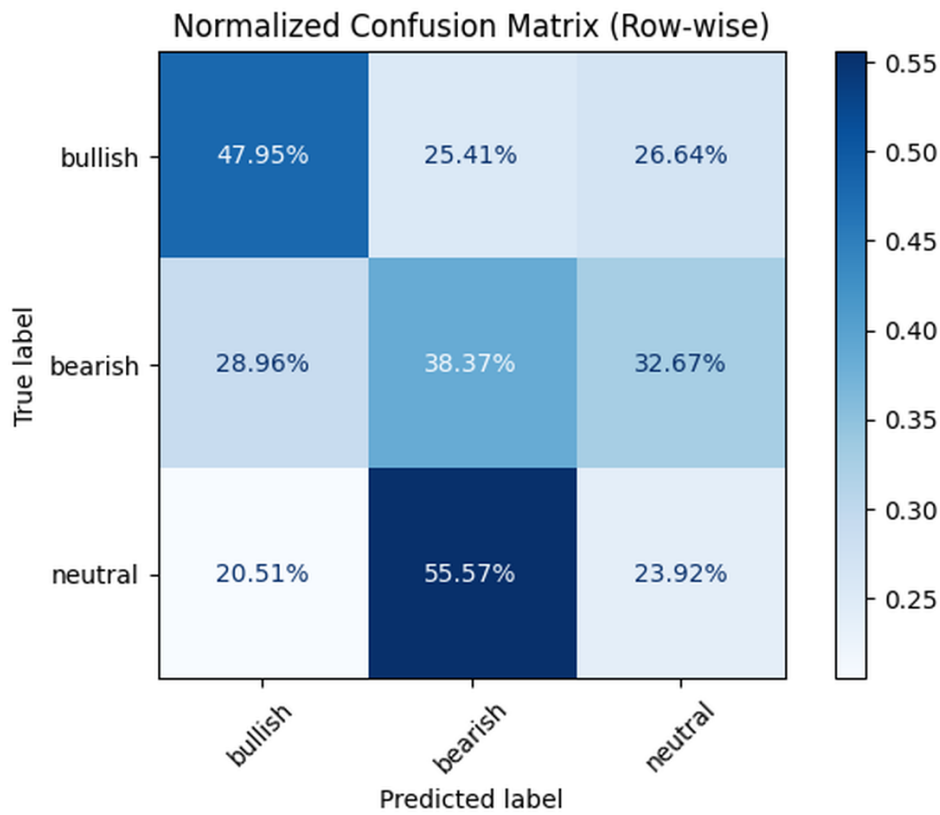


Figure 4.17: Finbert finetuning Training Loss vs Validation Loss



| Metric | Value |
|-----------|--------|
| Accuracy | 0.3935 |
| Precision | 0.3798 |
| Recall | 0.3675 |
| F1-score | 0.3706 |

Figure 4.18: Confusion matrix and macro-averaged metrics for finetuned Finbert (no extension module) on the Stocktweets-Emoji test set.

This simple approach grants pretty good results compared to exBert Finbert-based. By further finetuning we mitigate the tendency of the model to always predict the neutral class and we achieve an accuracy of 39,34%.

4.7. Models Comparison

In this section, we provide a deeper analysis of the results obtained by the different models and compare the best-performing system with human performance on the same classification task.

The purpose is to better understand both the effectiveness of automating sentiment classification and the usefulness of the proposed extension module technique. Among the two test datasets used in the experiments, only the Stocktweets-Emoji dataset contains posts distributed across all three sentiment labels (Bullish, Bearish, and Neutral), while the SPY-Stocktweets dataset is restricted to only Bullish and Bearish posts. For this reason, results derived from the Stocktweets-Emoji dataset are considered more reliable for drawing general conclusions, as they reflect the full complexity of the task. Consequently, the comparisons between models will primarily rely on the outcomes obtained from this dataset. Summarizing the results, the two baseline models FinBERT and RoBERTa show suboptimal performance when directly applied to the test data. While RoBERTa outperforms FinBERT in terms of accuracy, both models achieve limited predictive power, especially in distinguishing directional sentiment from the Neutral class. Importantly, however, the introduction of the extension module consistently improves classification accuracy for both models by approximately 10 percentage points. This confirms the general effectiveness of the approach in enhancing the model’s ability to capture sentiment signals. The most remarkable improvements are observed in the RoBERTa-based extended model, which achieves the highest accuracy among all models tested. This model is capable of correctly classifying the majority of Bullish posts while still capturing a meaningful portion of Bearish and Neutral cases. Its performance clearly demonstrates the benefits of the extension strategy, not only in raising accuracy but also in producing a more balanced distribution of predictions across categories. Even the fine-tuned FinBERT variant, though less powerful than the extended RoBERTa, achieves noticeable gains compared to the plain FinBERT baseline, reaching an accuracy of approximately 40%. This result suggests that while the extension module provides an additional boost, it is not strictly indispensable for achieving a reasonable level of classification performance. Careful fine-tuning of the base architecture alone can already yield significant improvements.

A second way to assess the necessity of the extension module lies in analyzing the models’ performance on posts that contain a large number of emojis. By inspecting the dataset, we observed that in several cases, emojis play a determinant role in conveying the true sentiment of the author. For instance, posts composed solely of emojis often rely entirely on symbolic expression, leaving no textual clues for a language-only model. In these

cases, the extended models demonstrate clear advantages, as they are explicitly designed to interpret emoji semantics within context. Conversely, there are also examples in which emojis are misleading or add little value, and sentiment can be determined purely from the textual component. This highlights the dual nature of the extension module: it is more useful when dealing only with emojis and may de grade model’s performances when it is able to correctly label the post by using just the words contained in it, and in this case emojis can be misleading. Taken together, these findings support two key conclusions. First, the extension module represents an effective enhancement strategy, improving accuracy, reducing Neutral bias, and allowing the models to better interpret sentiment-bearing emojis. Second, the results suggest that while the extension is highly beneficial in emoji-rich scenarios, well-executed fine-tuning can also provide competitive improvements without additional modifications. Therefore, the ultimate choice between standard fine-tuning and the extension module depends on the target application: in domains where emojis are central to communication, the extension proves crucial, while in more traditional textual datasets, fine-tuned base models may be sufficient.

Moreover, to justify the automation of the sentiment classification task through a Large Language Model we need to compare the ability of the model human performances on the same task. In order to do so human evaluation experiment was conducted on the Stocktweets-Emoji dataset. A sample of 100 posts was randomly selected, balanced across the three sentiment classes (33 bullish, 33 bearish, and 34 neutral). The posts were divided into five sets of 20 and distributed to participants using Google Forms, ensuring that each participant classified posts consecutively within the same form. A total of 138 forms were submitted, corresponding to 2760 individual responses. The participants in the study were students from the Politecnico di Milano, enrolled in the Department of Mathematics. This choice of participants was particularly relevant as their academic background suggests solid analytical skills and experience in the financial domain which should guarantee sufficient ability to interpret financial texts enriched with emojis. Two evaluation strategies were applied to measure human performance. First, accuracy was computed through majority voting, where the most frequently selected label for each post was compared with the true label. Second, accuracy was calculated at the level of individual votes, where each participant’s response was directly compared with the ground truth.

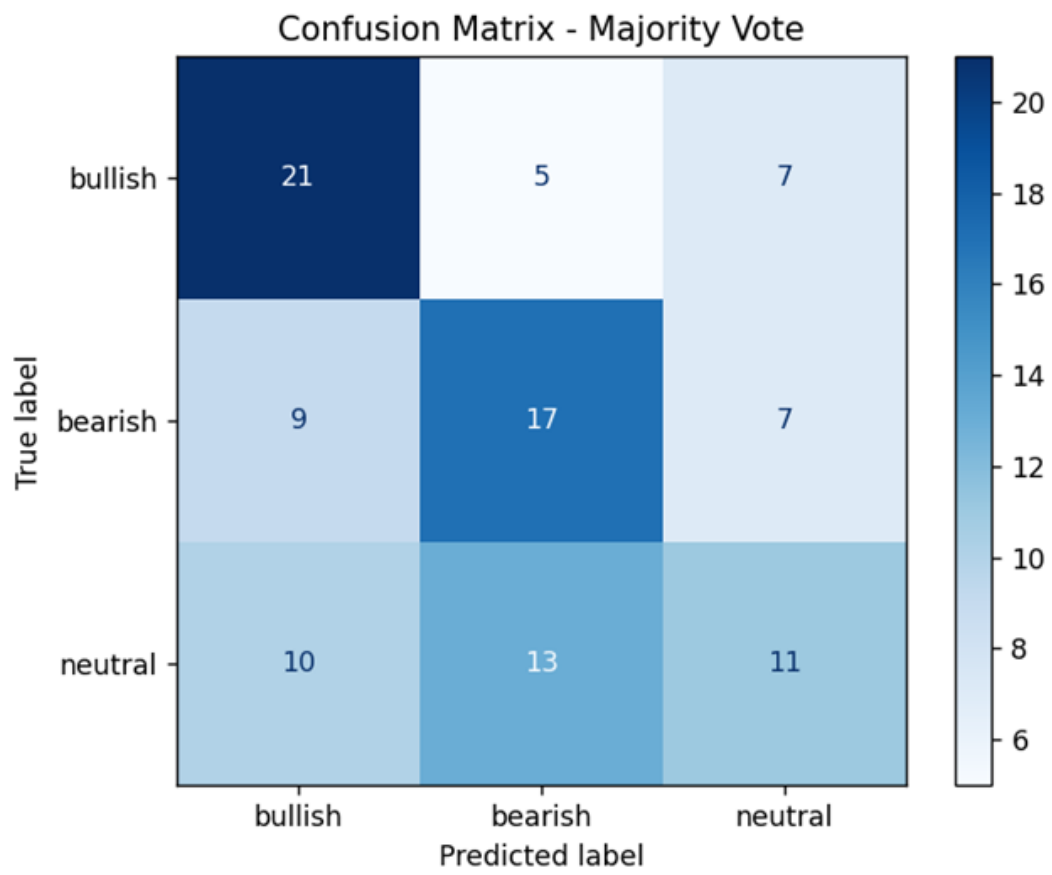


Figure 4.19: Human Accuracy on sentiment classification task (Majority Voting)

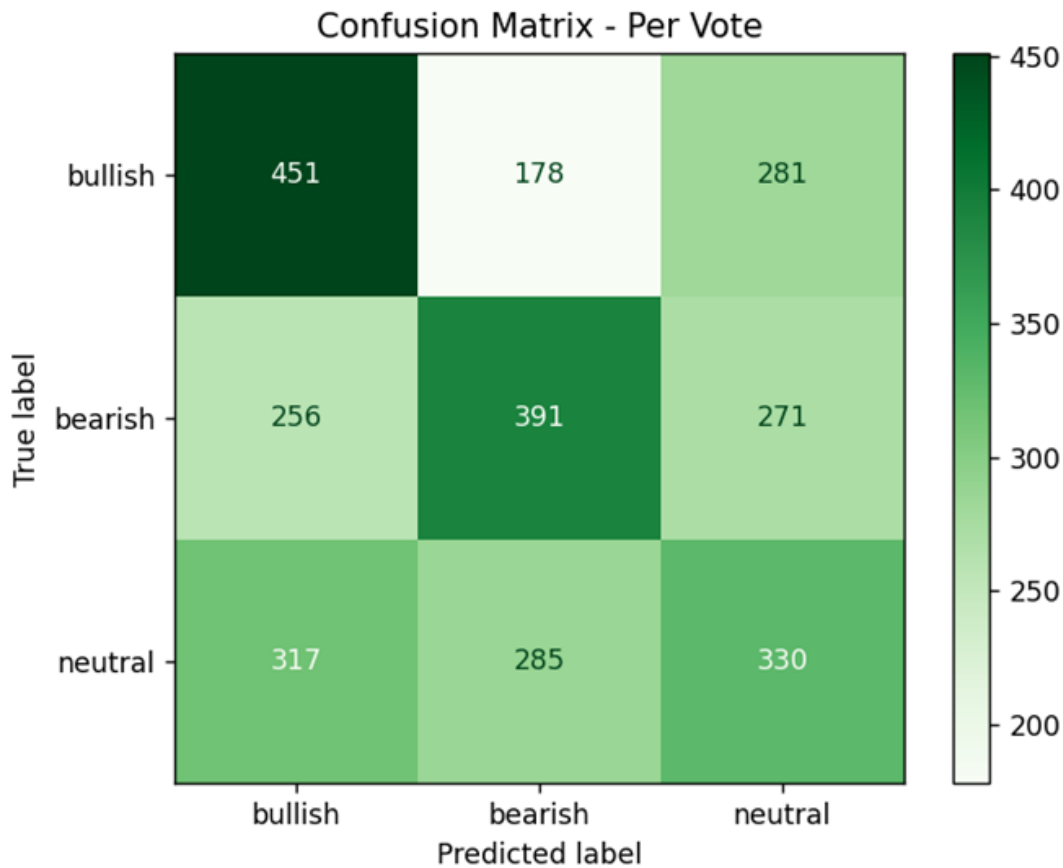


Figure 4.20: Human Accuracy on sentiment classification task (Single Voting)

The first approach yielded an overall accuracy of 49%, indicating that when aggregating human judgments, performance was slightly below 50%. The second approach where accuracy was calculated at the level of individual votes, dropped to 42.46%.

These results suggest that even for humans, the sentiment classification of stock-related posts enriched with emojis is a challenging task. Ambiguities in language, the financial domain related meaning of terms, the nature of emojis, and the context-dependent meaning of posts contribute to relatively low performance. The fact that majority voting achieves higher accuracy than individual votes highlights the benefit of aggregating multiple perspectives to mitigate individual misinterpretations. When compared to these human baselines, the performance of **the best LLM models obtained above (the ROexBERT model) has similar accuracy results**. On the same dataset, ROexBERT achieves an accuracy of around 45%, a performance which is in line with human annotators. While the difference may appear modest in absolute terms, it is highly significant given the difficulty of the task. Importantly, ROexBERT achieves this level of performance consistently

and without the variability observed among individual annotators.

Finally, the human baseline obtained in this experiment provides a meaningful benchmark against which the performance of automated models can be compared, highlighting both the complexity of the task and the potential value of robust NLP approaches in speeding up the task of classifying a huge amount of posts coming from web sources and eliminating the necessity of humans annotators by achieving similar performances in term of accuracy

5 | Price Prediction Model

The final goal of this research is to accurately predict the next timestep in a time series analysis of financial data. In this chapter it will be shown how the above market sentiment extraction process is used to build a novel feature to enhance financial assets price prediction accuracy.

5.1. Price Forecasting

The models for sentiment classification developed in the previous chapter allow us to assign a sentiment score with any desired time frequency to the market of any target financial asset. This sentiment score reflects the opinion and emotions of the authors of the analyzed posts and therefore it will be used together with more classical financial data as a feature for price prediction algorithms. By taking posts from web sources, published during the target time interval, we model the sentiment score as an integer number using the simple formula:

$$\text{Sentiment Score} = \#\text{Bullish} - \#\text{Bearish}.$$

Clearly, the assumption is that the bigger the sentiment score the more confident investors feel about positive future performance of the target asset. Conversely, the lower the sentiment score, the more negative are the expected performances of the market. This metric therefore captures market mood in a straightforward and interpretable way.

Alongside the novel feature we will use two classical features that are well known and widely used in the literature to perform price prediction over time series: **past returns and past volumes**. Returns represent the relative change in the price of an asset over consecutive time intervals. They are widely used in finance as a normalized measure of price variation and are typically computed as either simple returns

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}},$$

where P_t denotes the asset price at time t . In this work we use returns instead of raw prices because returns are generally considered a *stationary* process, meaning that their statistical properties such as mean and variance remain approximately constant over time. This property makes returns more suitable for modeling and prediction, as many statistical and machine learning methods, including LSTM networks, assume or benefit from stationary input data. Volumes measure the total quantity of an asset traded during a given time interval, usually expressed as the number of shares, contracts, or units exchanged. Mathematically, volume at time t can be defined as

$$V_t = \sum_{i=1}^{N_t} q_i,$$

where q_i is the number of units exchanged in trade i , and N_t is the total number of trades within the interval t .

In this case we want to assess the quality and the importance of the sentiment score feature and understand whether it could be a significant price movement indicator. Moreover, beside the impact of the feature itself, we want to analyze the effect of the *quality* of the sentiment score. To this end, the same dataset will be labeled by different sentiment models described above, and we will evaluate the predictive results taking into account the accuracy of the model that generated the sentiment feature. This allows us not only to measure the contribution of sentiment as a feature, but also to quantify the impact of improving sentiment classification quality on downstream financial forecasting.

5.2. LSTM Architecture

Among the numerous algorithms for times series forecasting discussed in the literature review section, the most recent approach and therefore coherent with the deep learning methods adopted during this research is the LSTM architecture. In particular, the predictive model is based on a deep recurrent architecture employing three stacked bidirectional LSTM layers. As mentioned above the network receives three input features: Returns, Volume and Sentiment Score. Since the LSTM architecture works on sequential data, it is necessary to decided the window length of the input sequence: this parameter is eventually set to 15 time steps. The input layer receives sequential data of the described shape, which is then processed by the recurrent stack. The first two bidirectional LSTM layers, each with 256 hidden units, are configured to return sequences, thereby preserving temporal information across the layers. A third bidirectional LSTM layer with 384 hidden units follows, summarizing the sequential dynamics into a richer representation.

To reduce overfitting, a dropout layer with a rate of 0.1 is applied to the final recurrent output. The network concludes with a fully connected dense layer with ReLU activation, producing a single continuous output suitable for regression tasks such as financial time-series prediction. The choice of ReLU activation helps to ensure stability in the output and prevents vanishing gradient issues. The model is compiled using the Adam optimizer with the mean squared error (MSE) as the primary loss function, while both mean absolute error (MAE) and MSE are tracked as evaluation metrics during training. This configuration is designed to capture complex temporal dependencies in sequential data while maintaining robustness and generalization.

Table 5.1: LSTM Model Architecture

| Layer (type) | Output Shape | Param # |
|---------------------------|-----------------|-----------|
| Input (InputLayer) | (None, 15, 3) | 0 |
| Bidirectional (LSTM, 256) | (None, 15, 512) | 532,480 |
| Bidirectional (LSTM, 256) | (None, 15, 512) | 1,574,912 |
| Bidirectional (LSTM, 384) | (None, 768) | 2,755,584 |
| Dropout (rate=0.1) | (None, 768) | 0 |
| Dense (ReLU, 1) | (None, 1) | 769 |

| | |
|-----------------------------|-----------------------|
| Total Parameters | 14,591,237 (55.66 MB) |
| Trainable Parameters | 4,863,745 (18.55 MB) |
| Non-trainable Params | 0 (0.00 B) |
| Optimizer Parameters | 9,727,492 (37.11 MB) |

Details on layers, hidden units, and activation functions are summarized in the table above. All the hyper-parameters in the network were tuned with the Bayesian Optimization algorithm implemented in the tuner of the Python library TensorFlow Keras. The relatively large number of parameters (over 14 million) reflects the depth of the network and its ability to capture nonlinear dependencies in the data, but it also requires a careful managing of regularization and validation to avoid overfitting.

5.3. Trading Strategy

Finally, the effectiveness of the prediction model must also be evaluated in terms of its practical application to trading. To this end, we consider a simple comparison between two strategies: a **naive baseline strategy** and **LSTM prediction-based informed strategy**. The model-based trading strategy assumes that a position is opened at $t - 1$

and closed at t , with investment amounts determined by the magnitude of the predicted return. The algorithm works as follow:

- If $1.005 \leq r_t \leq 1.01$, invest by buying for \$100.
- If $r_t > 1.01$, invest by buying for \$500.
- If $0.99 \leq r_t \leq 0.995$, invest by selling for \$100.
- If $r_t < 0.99$, invest by selling for \$500.
- Otherwise, no trade is executed.

The naive trading strategy based only on observed past returns, without any information on future returns generated by the model. At each step, the return from the previous time interval determines the trading decision for the next interval.

- If $1.001 \leq r_{t-1} \leq 1.005$, open a small long position (\$100).
- If $r_{t-1} > 1.005$, open a large long position (\$500).
- If $0.995 \leq r_{t-1} \leq 0.999$, open a small short position (\$100).
- If $r_{t-1} < 0.995$, open a large short position (\$500).
- Otherwise, no trade is executed.

This simple rule-based baseline provides a lower-bound benchmark: it exploits only past observations and no predictive modeling, thereby allowing us to directly measure the added value introduced by the sentiment-informed LSTM predictions.

The cumulative profit is obtained by summing the gains or losses of each operation across the entire time horizon in both strategies. This framework allows us to quantify the contribution of sentiment-informed predictions not only in terms of accuracy metrics, but also in terms of potential financial gain. The comparison between naive and model-based strategies therefore highlights the practical utility of incorporating sentiment analysis into financial forecasting pipelines. In particular, it allows us to understand whether the complexity of the LSTM model is justified by a measurable improvement in real-world trading performance.

5.3.1. Experiments

In order to evaluate the effectiveness of the proposed sentiment score as an additional feature for financial time-series forecasting we will conduct experiments over the price prediction of the well know S&P 500 index (\$SPY ticker). The experimental setup involves

the use of the SPY-StockTweets dataset also used in the previous chapter to calculate the sentiment score for the asset. This is the main reason why we focused also on this dataset in the evaluation of the models, even though it does not contain Neutral posts. The dataset consists of a total of 3,543,945 tweets covering 498 business days, from 01-04-2020 to 13-03-2022, each with its corresponding date and time of publication. Among these, 1,419,711 tweets are human-labeled, while the remaining 2,124,234 are unlabeled and can be exploited for semi-supervised training or as additional unlabeled data. To align sentiment information with traditional financial indicators, historical market data for the \$SPY ticker (returns and traded volumes) were downloaded for the same time span using the YahooFinance API. This pairing enables the construction of a unified dataset that integrates market sentiment, historical returns, and volumes, providing the basis for evaluating the predictive contribution of the sentiment score feature. Now the data pipeline works as follows: tweets are first fed into the sentiment classification models proposed in the previous chapter. Each model outputs a sentiment label for every post, after which the sentiment score formula is applied by aggregating the posts published within the target time frame of the experiment. To ensure a wider perspective, two experiments were conducted over different aggregation windows: **Daily** and **Hourly**. Consequently, returns and volumes were also sampled at the same frequency as the sentiment score, pairing daily returns and volumes of the \$SPY ticker with the daily calculation of the sentiment score. The same procedure was applied for the hourly time frame. The resulting dataset contains time series with three features: *Returns*, *Volume*, and *Sentiment Score*. This dataset is split into training and test sets using a 90%-10% partition. The training set is further divided with another 90%-10% split into training and validation subsets, which are used to tune the parameters of the LSTM network. Once trained, the model is applied to predict future returns. To evaluate predictive performance, we rely on two standard error metrics: the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). MSE penalizes larger deviations more heavily, making it useful to assess whether the model avoids large mistakes, while MAE provides a more interpretable measure of the average prediction error in the same scale as the returns. Using both metrics ensures a comprehensive evaluation of predictive accuracy. After prediction, the two trading strategies described above are applied to further assess the practical implications of the results. **Lastly, as a baseline model for comparison, an LSTM network was trained using only returns and prices as input features, excluding the sentiment score computed by the language model. This baseline setup allows us to isolate the contribution of the sentiment score and assess its added value in enhancing predictive performance**

5.3.2. Results Evaluations

First we start off the analysis by comparing the model trained without the sentiment score feature and the one trained using the sentiment score. For this experiment the selected time frame is daily.

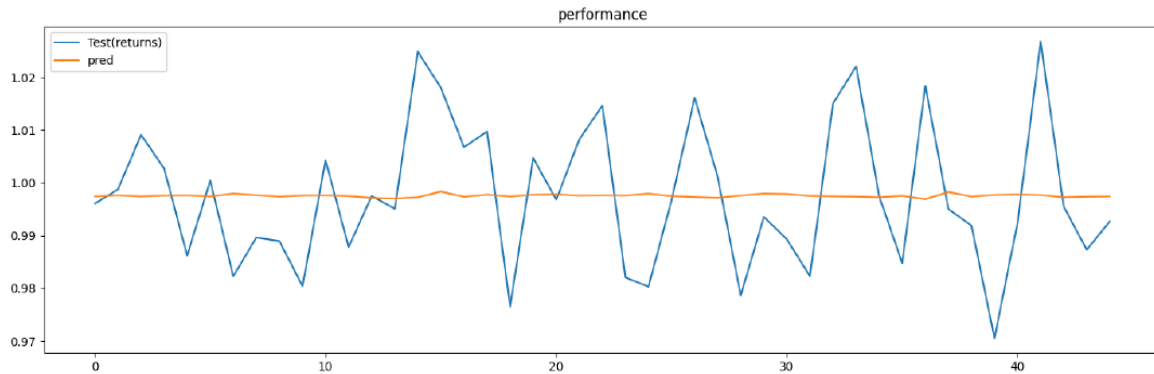


Figure 5.1: Daily returns predictions without the sentiment score feature

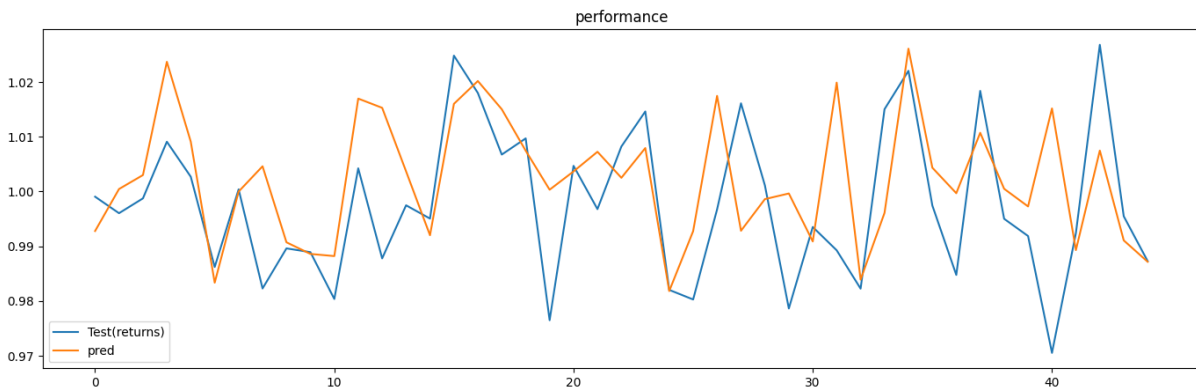


Figure 5.2: Daily returns predictions with the sentiment score feature

The first graph shows the predicted returns (in orange) as flat line around 1.00, the mean of returns, while the actual returns (in blue) fluctuate significantly. Clearly LSTM trained only on past returns and volumes without sentiment is unable to capture the variability of the market and the model as expected predicts the mean since it minimizes MSE but fails to provide meaningful short-term dynamics. On the other side, the second graph shows a stronger alignment between the predicted returns and the actual returns. While the model still smooths the volatility, it now captures the direction of several movements, correctly following local spikes. This improvement highlights the contribution of the sentiment score as an additional feature: it injects information that allows the model to better

anticipate fluctuations that cannot be explained by past prices and volumes alone. Therefore, the comparison shows that **including the sentiment feature makes the LSTM model more responsive to short-term variations and enhances its predictive capability**. While the predictions remain imperfect, they provide a closer approximation of actual market behavior compared to the baseline model without sentiment.

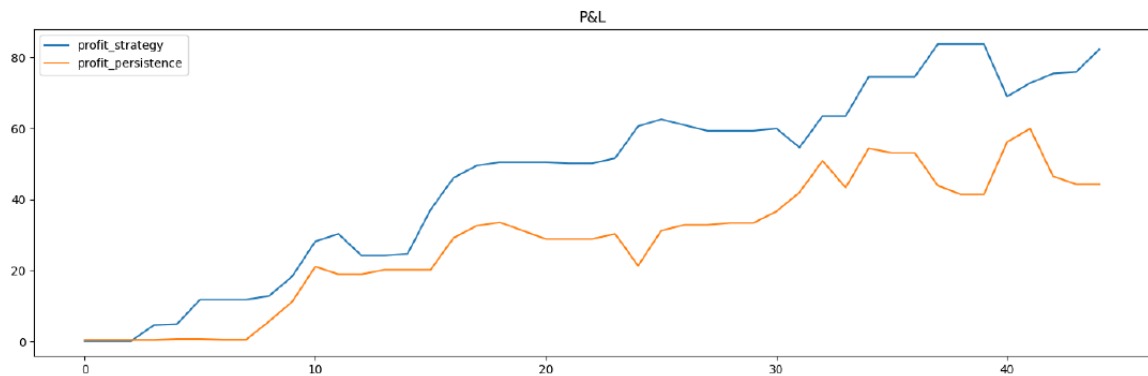


Figure 5.3: Net Profit and Loss following the prediction-based trading strategy and the naive trading strategy

The above graph shows the simulation of the value of a portfolio managed with the two strategies proposed in the Trading Strategy section. In blue the prediction-based strategy and in orange the naive strategy. Even though both strategies guarantee net profit over the simulated time span the prediction-based strategy outperforms the naive strategy in terms of financial gain, enforcing the claim of the usefulness of the sentiment score as important feature.

The same experiment is repeated over the hourly time frame.

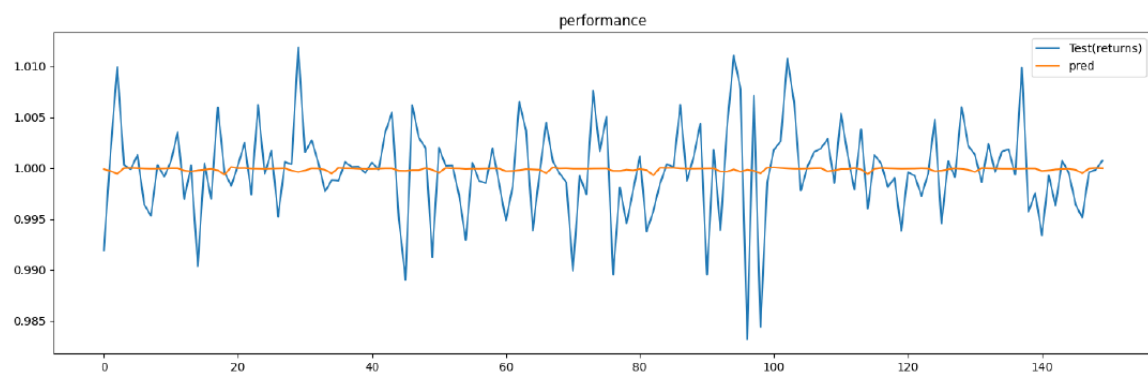


Figure 5.4: Hourly returns predictions without the sentiment score feature

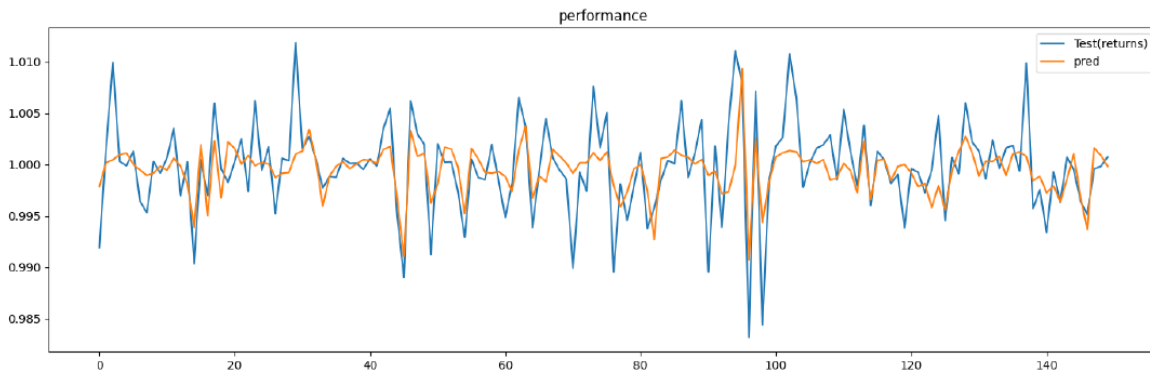


Figure 5.5: Hourly returns predictions with the sentiment score feature

Once again the second graph displays a much stronger accuracy of the model's predicted returns. The LSTM is now able to capture short-term fluctuations with more precision. Although some mismatches remain, we can conclude that the inclusion of sentiment information clearly enhances model performances and predictive capabilities, allowing it to approximate local variations instead of converging to a flat average.

Furthermore, the comparison highlights that **sentiment contributes more significantly in the hourly setting than in the daily one**. This is because intraday movements are often driven by fast-changing investor sentiment expressed in posts and tweets, making sentiment features especially valuable in capturing short-term market dynamics. Daily aggregation, by contrast, tends to smooth out these fluctuations, potentially obscuring the immediate market impact of sudden sentiment shifts. For this reason, the next experiments will be conducted over the hourly time frame. Operating at higher frequency not only allows the predictive model to fully exploit the information carried by the sentiment score feature but also unlocks the full potential of the automation approach in trading. Automated systems are particularly suited to process large streams of real-time information and to react quickly without the delays and biases typical of human decision-making. By working with hourly data, the model can continuously update trading signals and execute strategies with greater reactivity, capturing opportunities that would otherwise be lost on a daily horizon. In this way, the combination of sentiment analysis and high-frequency forecasting maximizes the usefulness of automation in financial markets.

After assessing the usefulness of the sentiment score feature we will evaluate the impact that the different quality of predictions by the different models developed in the previous chapter has on the price forecasting.

Table 5.2: Comparison of Models using MSE and MAE

| Features | MSE | MAE |
|-----------------|--------------|------------|
| FinBERT | 1.75533e-05 | 0.002964 |
| RoBERTa-twitter | 1.675195e-05 | 0.002830 |
| FinBERT FT | 1.481982e-05 | 0.002781 |
| exBert | 1.506633e-05 | 0.002764 |
| ROexBERT | 1.424594-05 | 0.002564 |

The table shows the performance of the different models discussed in chapter 4 when used to build the sentiment score features, evaluated in terms of Mean Squared Error (MSE) and Mean Absolute Error (MAE). FinBERT, the baseline model, shows the highest error values among all the tested configurations, confirming its limited suitability for handling the emoji-rich financial text of the dataset. RoBERTa-twitter improves over FinBERT, achieving both lower MSE (1.67×10^{-5}) and MAE (0.00283). Further improvements are observed when the models are fine-tuned (FT). FinBERT FT actually outperforms exBert since its prediction accuracy over the dataset is better. The best performing model is ROexBERT, combining RoBERTa’s stronger architecture with the extended vocabulary. It achieves the lowest errors across the board with an MSE 1.42×10^{-5} and MAE 0.0026, confirming the effectiveness of the model. As expected these results highlight the importance of the quality of the predictions given by the different models since these predictions are the one used to calculate the sentiment score. The table shows that by improving the upstream task of correctly classifying posts by using a more accurate model improves the downstream task of predicting next step return for the asset. For this reason, being able to quickly and correctly classify posts and being able to correctly interpret emojis is crucial if we want to obtain more accurate price prediction performances.

6 | Conclusions and Future Work

In this chapter we will sum up the methodology and the results reported in the previous chapters and draw the final conclusions of the work. Moreover we will point out possible ideas to further develop and improve the research and the results obtained.

This thesis sets out whit the objective to investigate the potential of extending pre-trained Large Language Models such as BERT with domain-specific vocabulary, with the purpose of improve sentiment analysis in the context of emoji-rich online social media post and then to evaluate the usefulness of sentiment information as a feature in financial time-series forecasting. The two contributions are closely connected: on one hand, we aimed to improve the accuracy of sentiment classification on financial social media posts, a task complicated by the use of informal language and heavy use of emojis; on the other hand, we assessed whether these improved sentiment representations translate into measurable gains in predictive modeling for asset prices.

The first part of the work focused on the extension of existing pre-trained models through the exBERT approach. By adding to the model's original vocabulary emojis and extending the inner layers of the network, and by fine-tuning only the extended model under constrained training resources, we demonstrated that it is possible to significantly enhance the classification performance of models such as FinBERT and RoBERTa. The results showed that exBERT can outperform its non-extended counterparts, reducing misclassifications particularly in posts where sentiment is expressed primarily through emojis. The best results were achieved with ROexBERT, which combined RoBERTa's stronger architecture with an extended vocabulary, achieving the highest accuracy and balanced predictions across bullish, bearish, and neutral classes. These findings confirm the validity of the extension module approach as an efficient strategy to adapt large pre-trained models to specialized domains without requiring full retraining.

The second part of the research integrated the sentiment predictions into a price forecasting model based on an LSTM architecture. Here, the sentiment score, defined as the difference between the number of bullish and bearish posts in a given time frame, was used alongside classical financial features such as returns and volumes. The experiments

highlighted two key results. First, the inclusion of the sentiment score systematically improved forecasting performance over models trained exclusively on traditional financial indicators. Both daily and hourly experiments showed better alignment between predicted and actual returns when sentiment was included, with the effect being particularly strong in the hourly setting. This confirms that sentiment captures short-term market dynamics that are otherwise invisible when relying solely on past prices and volumes. Second, the quality of the underlying sentiment classification model directly impacted forecasting accuracy. Models based on extended vocabularies, such as exBERT and ROexBERT, provided sentiment scores that led to lower error rates in return prediction compared to those generated by baseline FinBERT or RoBERTa models. This demonstrates a direct link between advances in sentiment classification and downstream improvements in financial forecasting. Finally, the trading strategy simulations further reinforced these findings. A sentiment-driven strategy, guided by the predictions of the LSTM model enriched with sentiment information, outperformed a naive return-based strategy in terms of cumulative financial gain. This result provides practical evidence that sentiment analysis is not merely an auxiliary tool but can deliver tangible benefits when integrated into automated trading pipelines.

In conclusion, this thesis provides a twofold contribution: it validates the exBERT extension approach as a resource-efficient method to improve domain-specific language models, and it demonstrates the value of sentiment as a predictive feature in financial forecasting. The evidence suggests that sentiment-informed models can complement traditional financial indicators, enhancing both predictive accuracy and trading performance. These findings encourage further exploration of domain-adapted language models in finance and highlight the importance of leveraging investor sentiment to better anticipate market movements.

6.1. Future Work

Building on these results, several paths for future research can be pursued.

The first improvement would be to integrate the entire framework into a fully automated trading pipeline. This would involve implementing a system capable of continuously downloading financial posts from online platforms in real time, classifying them with the sentiment models developed in this thesis, and using the aggregated sentiment score as input to the forecasting model. Such a system would produce trading signals on the fly and execute them automatically, effectively realizing an end-to-end sentiment-driven trading machine. This would not only test the feasibility of the proposed approach in real-

world conditions but also allow for the evaluation of additional factors such as latency, transaction costs, and market impact.

As highlighted in the theses, the sentiment classification accuracy is crucial in order to obtain better price forecasting performance. Therefore, a promising direction is the development of a hybrid architecture for sentiment classification that could improve the effectiveness of the framework. Since financial posts differ widely in their composition, some being dominated by textual content while others rely heavily on emojis, it may be beneficial to dynamically select the most appropriate model depending on the nature of the input. For instance, a threshold could be defined for emoji density, with posts exceeding this threshold routed to an extended model like exBERT or ROexBERT, while more text-oriented posts could be processed by lighter baseline models which performs better over text-only content. Such an adaptive strategy could improve classification accuracy, ultimately enhancing the quality of the sentiment signal used for forecasting.

By pursuing these extensions, future research can further consolidate the role of sentiment-informed models in financial forecasting and strengthen the bridge between natural language processing advances and practical trading applications.

Bibliography

- [1] T. Adams, A. Ajello, D. Silva, and F. Vazquez-Grande. More than words: Twitter chatter and financial market sentiment. *Finance and Economics Discussion Series*, (2022-009), 2022. doi: 10.17016/FEDS.2022.009. URL <https://doi.org/10.17016/FEDS.2022.009>.
- [2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL <https://arxiv.org/abs/1607.06450>.
- [3] F. Barbieri, F. Ronzano, and H. Saggion. What does this emoji mean? a vector space skip-gram model for twitter emojis. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC)*, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).
- [4] G. E. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1970.
- [5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [6] F. Cao, G. Zhang, J. Yaog, S. Nig, and Y. Zhangg. Sentiment inferencing model for stock related comments. <https://huggingface.co/zhayunduo/roberta-base-Stocktwits-finetuned>, 2022. Accessed on 17 June 2022.
- [7] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, 1995. doi: 10.1007/BF00994018.
- [8] J. Deveikyte, H. Geman, C. Piccari, and A. Provetti. A sentiment analysis approach to the prediction of market volatility. *arXiv*, 2020. Preprint.
- [9] J. Devlin, M.-W. 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*, 2019. URL <https://arxiv.org/abs/1810.04805>.
- [10] T. Dimson. Emojineering part 1: Machine learning

- for emoji trends. <https://instagram-engineering.com/emojineering-part-1-machine-learning-for-emoji-trends-machine-learning-cf5f568b472b>
2015. Instagram Engineering Blog.
- [11] D. Dodd and B. Graham. *Security Analysis*. Whittlesey House, New York, 1934.
- [12] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. *Advances in Neural Information Processing Systems*, pages 155–161, 1997.
- [13] B. Eisner, T. Rocktäschel, I. Augenstein, M. Bošnjak, and S. Riedel. emoji2vec: Learning emoji representations from their description. *arXiv preprint arXiv:1609.08359*, 2016. URL <https://>.
- [14] M. Fernández-Gavilanes, J. Juncal-Martínez, S. García-Méndez, E. Costa-Montenegro, and F. J. González-Castaño. Creating emoji lexica from unsupervised sentiment analysis of their descriptions. *Expert Systems with Applications*, 103:74–91, 2018. doi: 10.1016/j.eswa.2018.02.043. URL <https://doi.org/10.1016/j.eswa.2018.02.043>.
- [15] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1243–1252. PMLR, 2017. URL <https://arxiv.org/abs/1705.03122>.
- [16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. URL <https://arxiv.org/abs/1512.03385>.
- [17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [18] D. Jurafsky and J. H. Martin. *Speech and Language Processing (3rd edition draft)*. 2023. <https://web.stanford.edu/~jurafsky/slp3/>.
- [19] P. Kralj Novak, J. Smailović, B. Sluban, and I. Mozetič. Sentiment of emojis. *PLOS ONE*, 10(12):e0144296, 2015. doi: 10.1371/journal.pone.0144296. URL <https://doi.org/10.1371/journal.pone.0144296>.
- [20] T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the*

- 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, 2018.
- [21] X. Li, X. Fu, G. Xu, Y. Yang, J. Wang, L. Jin, Q. Liu, and T. Xiang. Enhancing bert representation with context-aware embedding for aspect-based sentiment analysis. *IEEE Access*, 8:46868–46876, 2020. doi: 10.1109/ACCESS.2020.2978511.
- [22] Y. Lou, Y. Zhang, F. Li, T. Qian, and D. Ji. Emoji-based sentiment analysis using attention networks. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 19(5):1–13, 2020. doi: 10.1145/3389035. URL <https://doi.org/10.1145/3389035>.
- [23] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR*, 2013. URL <https://arxiv.org/abs/1301.3781>.
- [24] S. Y. K. Mo, A. Liu, and S. Y. Yang. News sentiment to market impact and its feedback effect. *Environment Systems and Decisions*, 36(2):158–166, 2016. doi: 10.1007/s10669-016-9580-1.
- [25] J. J. Murphy. *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. Penguin, 1999.
- [26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.
- [27] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.
- [28] W. Tai, H. T. Kung, X. Dong, M. Comiter, and C.-F. Kuo. exbert: Extending pre-trained models with domain-specific vocabulary under constrained training resources. *arXiv preprint arXiv:2004.00018*, 2020. URL <https://arxiv.org/abs/2004.00018>.
- [29] H. Tang, W. Tang, D. Zhu, S. Wang, Y. Wang, and L. Wang. Emfsa: Emoji-based multifeature fusion sentiment analysis. *PLOS ONE*, 19(9):e0310715, 2024. doi: 10.1371/journal.pone.0310715. URL <https://doi.org/10.1371/journal.pone.0310715>.
- [30] P. C. Tetlock. Giving content to investor sentiment: The role of media in the stock market. *The Journal of Finance*, 62(3):1139–1168, 2007. doi: 10.1111/j.1540-6261.2007.01232.x.

- [31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL <https://arxiv.org/abs/1706.03762>.
- [32] Y. Wu, M. Schuster, Z. Chen, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [33] Y. Zhang, R. Jin, and Z.-H. Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1):43–52, 2010. doi: 10.1007/s13042-010-0001-0.

List of Figures

| | | |
|------|--|----|
| 2.1 | Overview of the sentiment-enhanced trading agent architecture. | 5 |
| 3.1 | Encoder-Decoder Transformer Architecture. Vaswani et al. [31] | 10 |
| 3.2 | Overview of the sentiment-enhanced trading agent architecture. | 14 |
| 3.3 | Architecture of an LSTM cell showing the input gate, forget gate, and output gate that control the flow of information. | 20 |
| 4.1 | Top 10 Most Frequent Emojis Distribution | 23 |
| 4.2 | Post Length Distribution in number of words(whitespace separation) . . . | 24 |
| 4.3 | Top 10 Most Frequent Emojis Distribution | 25 |
| 4.4 | Post Length Distribution in number of words(whitespace separation) . . . | 26 |
| 4.5 | exBert model architecture Tai et al. [28] | 30 |
| 4.6 | Fibert-Based exBert Training Loss vs Validation Loss | 34 |
| 4.7 | FinBERT-Based exBert Validation Accuracy | 34 |
| 4.8 | ROBERTA-Based exBert Training Loss vs Validation Loss | 35 |
| 4.9 | Confusion matrix and macro-averaged metrics for FinBERT on the SPY–Stock- Tweets dataset. | 38 |
| 4.10 | Confusion matrix and macro-averaged metrics for exBert finbert-based on the SPY–StockTweets dataset. | 39 |
| 4.11 | Confusion matrix and macro-averaged metrics for Finbert on the Stock- Tweets–Emoji test set. | 40 |
| 4.12 | Confusion matrix and macro-averaged metrics for exBert FinBERT-based on the StockTweets–Emoji test set. | 41 |
| 4.13 | Confusion matrix and macro-averaged metrics for ROBERTA on the SPY–Stock- Tweets dataset. | 43 |
| 4.14 | Confusion matrix and macro-averaged metrics for exBERT ROBERTA-Based on the SPY–StockTweets dataset. | 44 |
| 4.15 | Confusion matrix and macro-averaged metrics for ROBERTA on the Stocktweet- Emoji dataset. | 45 |

| | | |
|------|--|----|
| 4.16 | Confusion matrix and macro-averaged metrics for exBERT ROBERTA-Based on the Stocktweet-Emoji dataset. | 46 |
| 4.17 | Finbert finetuning Training Loss vs Validation Loss | 47 |
| 4.18 | Confusion matrix and macro-averaged metrics for finetuned Finbert (no extention module) on the Stocktweets-Emoji test set. | 48 |
| 4.19 | Human Accuracy on sentiment classification task (Majority Voting) | 51 |
| 4.20 | Human Accuracy on sentiment classification task (Single Voting) | 52 |
| 5.1 | Daily returns predictions without the sentiment score feature | 60 |
| 5.2 | Daily returns predictions with the sentiment score feature | 60 |
| 5.3 | Net Profit and Loss following the prediction-based trading strategy and the naive trading strategy | 61 |
| 5.4 | Hourly returns predictions without the sentiment score feature | 61 |
| 5.5 | Hourly returns predictions with the sentiment score feature | 62 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | Examples of posts labeled with sentiment. | 4 |
| 4.1 | Example posts from the StockTwits Dataset with timestamp and sentiment label. | 22 |
| 4.2 | Example one-word posts from the StockTwits dataset. | 23 |
| 4.3 | Example posts from the dataset with corresponding sentiment labels . . . | 24 |
| 4.4 | Configuration of the BERT-based sentiment analysis model. | 27 |
| 4.5 | Configuration of the RoBERTa-based sentiment analysis model. | 28 |
| 4.6 | Configuration of the extention module. | 32 |
| 5.1 | LSTM Model Architecture | 57 |
| 5.2 | Comparison of Models using MSE and MAE | 63 |

Ringraziamenti

Devo confessare che questa è stato il paragrafo più difficile da scrivere tra tutti quelli presenti nella tesi. Non è mai facile riconoscere quello che gli altri fanno per noi e siamo sempre avidi di ringraziamenti poiché diamo molte delle cose che riceviamo per scontato. Per questa ragione voglio ringraziare tutte le persone che mi sono state vicini in questi anni. Il primo pensiero va alla mia famiglia: mio Padre Enrico, mia Sorella Giulia e mia Nonna Carla e in particolare mia madre Sabrina che mi hanno aiutato mettendomi a disposizione tutto il necessario per potermi concentrare sullo studio, compreso tanto affetto. Poi a tutti gli altri parenti: zie, zii e cugini che mi hanno sempre insegnato molto e sono stati per me modelli virtuosi. Un pensiero va ovviamente anche agli amici, compagni di studio, di divertimento ma soprattutto di vita. Con rammarico non posso nominarli tutti in questo breve paragrafo ma chi sa di aver condiviso i migliori anni e le più significative esperienze con me si sentirà chiamato in causa leggendo queste righe. L'unica menzione speciale la faccio a Ester, con cui ho condiviso dall'inizio alla fine questo percorso: ti ringrazio per avermi sempre motivato e aiutato a superare l'inerzia che ogni tanto mi affligge, ma soprattutto per essere sempre stata gentile e essermi stata vicina nel viaggio.