BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION Computer Science, English

DIPLOMA THESIS

Sentiment and Credibility-Adjusted Stock Price Prediction

Supervisor
Lect. Dr. Bogdan-Eduard-Mădălin Mursa

Author
Sergiu-Rareș Goian

2025

ABSTRACT

In the last years, there was a significant change in people's opinions about stock markets. Some decades ago, the majority of people were skeptical about investing their money and stock markets were usually seen as being accessible only for the richest. However, the number of investors has grown a lot and today the percentage of people who invest globally is in range 15% - 20%, according to statistics. Despite this growth, there is room for improvement for whoever wants to learn the basics of economics.

Nowadays, lots of tools are providing assistance for investment-related activities such as stock prediction and financial news sentiment analysis. Through this thesis application, we want to provide a risk-free environment where users can learn and become familiar with markets without losing their money. They will also be able to get an overview of future prices based on reliable news sentiment analysis using text classification for sentiment and credibility analysis and for time series forecasting a deep learning model.

The main feature of the app is stock price prediction with a short-term forecast, which uses data of last 2 years to predict prices over the next 2 weeks. Time series forecasting for the prices will be performed using a type of Recurrent Neural Network, called Long Short-Term Memory, designed for learning patterns in sequential data. The predicted prices will then be adjusted based on the reliability and sentiment of related news, using 2 fine-tuned TinyBERT models, one fine-tuned for sentiment analysis and the other one for credibility analysis.

All of these models will be integrated into a mobile app that has educational purpose in the finance field and provides an overview of the future prices for various stocks. In addition to this section, learning and quiz sections will be included (questions and answers for the quizzes are generated by Gemini API). They allow users to earn rewards in the form of virtual money, which can be used in a demo investing platform to build their own portfolios. These features create together a well organized step-by-step plan that allows users to learn financial basics from scratch, getting hands-on experience in a safe environment with real listed stock prices and use AI-powered tools to help them make the best choices.

Contents

Chapter 1

Introduction

1.1 Problem Space and Motivation

Stock markets are an important factor in the global economy, they drive economic growth and are often seen as an indicator of the economic situation. They allow individuals to invest almost any amount of money, with a wide range of alternatives such as mutual funds or exchange-traded funds.

Financial illiteracy remains a widespread issue. In their 2023 study, Annamaria Lusardi and Jialu L. Streeter examined this topic through a large scale US survey [1]. One of the key insights from the study is illustrated in the next table, which links individuals' financial knowledge to their financial behavior. The data demonstrates a clear relationship between financial literacy and responsible financial planning.

	Planners	Non-planners	Not Financially Fragile	Financially Fragile	Not too much debt	Too much debt
Interest rate question						
Correct	77.7	64.8	75.7	59.4	72.5	64.9
DK	8.7	19.5	10.5	22.8	14.7	15.6
Inflation question						
Correct	64.7	45.7	61.0	39.4	57.3	46.3
DK	12.2	30.4	16.6	33.9	22.1	24.0
Risk diversification question						
Correct	55.2	31.9	49.5	27.1	45.0	35.9
DK	30.5	55.8	37.5	59.4	43.6	47.7
Summary						
Correct: Interest and inflation	58	38.0	54.5	30.3	50.4	38
Correct: All three	41.9	19.3	36.6	13.4	32.5	21.7
Number of correct answers (among Big Three)	2.0	1.4	1.9	1.3	1.7	1.5

Figure 1.1: Table 4 of [1], showing how lack of knowledge in basic financial areas often correlates with limited engagement in essential long-term financial activities

Technology contributed to the development of the financial sector over the years. Digital platforms made the information gathering and learning process easier. However, there is still a large number of people who seem to fear getting started due to limited financial education. According to [16], technology such as artificial intelligence may change the way investors look at business decisions and risk. AI might

represent a learning tool for people who want to improve their knowledge about economy and stock markets.

Many beginners still face challenges in financial field. The major difficulty often stands in applying knowledge to actual investment. The idea of putting real money into an account and have the fear of losing a part or all of it can be really discouraging.

Most of the times, investors use news to get informed. Financial articles provide important insights, since investors often rely on reliable news sources to understand the current state of the market. However, the big volume of information available can be overwhelming, especially for beginners.

Beyond reading news, understanding the sentiment behind it is also important. Sentiment analysis helps by determining whether the overall trend of financial news is optimistic or pessimistic, since overall sentiment of the masses (market sentiment) has a significant impact on stock prices. By integrating sentiment analysis among the functionalities, these trends could be captured to predict better how public opinion might influence stock prices.

Another factor is the credibility of news, since many financial articles are inaccurate and exaggerated. Misinformation can cause chaos and drop prices. Once the panic fades, the prices tend to recover to the previous level. By integrating credibility analysis, the significance of the unreliable sources can be minimized to ensure the price predictions are based on credible articles.

1.2 Objectives

Given the importance of the factors described above, integrating AI features for news sentiment analysis and credibility assessment into stock price prediction provides a realistic approach to market forecasting and by combining these features, beginners can make better informed decisions.

This thesis presents an app designed to make financial learning more accessible. Its core feature is stock price prediction for short-term (next 2 weeks) using time series forecasting adjusted by two fine-tuned NLP (Natural Language Processing) models for news sentiment and credibility. Sentiment scores are weighted by credibility to adjust the forecasted prices, the impact fading over time.

The app also includes educational content, quizzes with rewards and a simulated trading platform. Users start with virtual money, which they can increase by solving quizzes or making smart demo investments. Access to the prediction tool costs virtual currency, encouraging learning before using AI-assisted decisions.

1.3 Original Contributions

Text classification models have emerged, mainly due to the recent developments in capabilities of large language models (LLMs). LLMs were propelled by the introduction of self-attention layers and transformers [25]. These innovations increased the precision of AI tasks for performing NLP tasks. NLP models are widely used in finance for tasks like sentiment and credibility analysis of news. Sentiment analysis links economic text to financial indicators [11], while credibility analysis helps assess information reliability [22]. Most existing models don't integrate these models together for price forecasting.

This work proposes a hybrid approach that extends LSTM-based forecasting with fine-tuned TinyBERT models for sentiment and credibility. Unlike traditional models relying solely on historical data, this method dynamically adjusts predictions using external news signals.

1.4 Thesis Structure

The first chapter highlights the role of technology in financial literacy. Chapter 2 reviews related work and existing models. Chapter 3 introduces key theoretical concepts, like LSTM for time series and transformers for our NLP models. Chapter 4 details the implementation, followed by results and conclusions in Chapter 5 and future improvements in Chapter 6.

Chapter 2

Related Work

2.1 State of the Art

In this chapter, we will explore previous methodologies, existing work, comparisons and limitations with a main focus on stock price prediction and text classification tools. There are many platforms that already integrate this type of feature (e.g. Etoro), giving users a chance to experiment using virtual money. For rewarded quizzes, apps like SaverLife offer money missions or gamified lessons to make financial education more fun.

Nowadays, many platforms make use of AI tools, including price prediction. We will describe different approaches for time series forecasting.

The table below presents a comparison of three common approaches to stock price prediction, statistical (ARIMA), machine learning (XGBoost) and deep learning (LSTM), based on standard evaluation metrics using data adapted from recent comparative studies [19, 14, 13].

Model	**RMSE**	**MAE**	**R^2**	**Strengths**	**Limitations**
ARIMA	35.42	24.68	0.9969	Effective for linear, stationary time series	Assumes linearity, poor for complex dynamics
XGBoost	32.38	22.06	0.9974	Captures non-linear relations, good with structured inputs	Requires manual feature engineering, lacks time-dependence modeling
LSTM	18.94	12.53	0.9987	Learns temporal dependencies; effective on sequential data	Requires larger datasets

Table 2.1: Comparison of Stock Price Prediction Models (Data adapted in this order from [19, 14, 13])

As shown above 2.1, LSTM outperforms both the other approaches in ability to

model time-dependent patterns. Thus, LSTM proves to be the most suitable model for stock price forecasting.

An even more recent development is represented by transformer-based models (we'll use them for our sentiment and credibility analysis tasks). These models are very good at understanding natural language, since they can look at all parts of a sequence at once, using a mechanism called self-attention (to capture links among sequences of words). They have achieved great performance in financial sentiment analysis too [2].

2.2 Comparisons of Existing Works

To analyze the sentiment of some text, researchers often use models based on transformer architecture. Researchers have increasingly adopted models like BERT (Bidirectional Encoder Representations Transformers) in financial NLP tasks due to its strong generalization and contextual understanding capabilities [4]. Its theoretical background will be approached in the theoretical chapter too.

The following table compares several models on popular sentiment benchmarks:

Model	**SST-2 Accuracy (%)**	**IMDb Accuracy (%)**	**Architecture Type**
BERT-base	93.5	94.1	Encoder (Bidirectional)
RoBERTa-base	94.6	95.2	Encoder (Robust BERT variant)
DistilBERT	91.3	92.8	Compressed Encoder
GPT-2 (small)	82.0	84.0	Decoder (Unidirectional)
T5-base	92.5	93.1	Encoder-Decoder
XLNet-base	94.1	94.8	Permutation-based Encoder

Table 2.2: Accuracy comparison of transformer models on sentiment and text classification tasks [26, 17].

Several variants of BERT developed to address specific needs such as speed, model size or domain specificity. TinyBERT is optimized for mobile and edge devices while retaining much of BERT's accuracy [8].

Model	#Params (M)	Pretraining Task	Specialization	Speedup vs BERT
BERT-base	110	MLM + NSP	General-purpose	1.0×
DistilBERT	66	MLM (distilled)	General-purpose (smaller)	1.6× faster
TinyBERT	14.5	MLM + distillation	Lightweight NLP	3–7× faster
FinBERT	110	MLM (financial corpus)	Financial sentiment	1.0× (domain-tuned)

Table 2.3: Overview of BERT variants by size, specialization, and inference speed [8, 21, 2].

In finance-specific applications, FinBERT generally outperforms generic models due to its domain-specific vocabulary and training corpus. However, TinyBERT provides an attractive trade-off in real-time where fast inference and small model footprint are needed.

Model	Financial News Accuracy (%)	Inference Time (ms)	Model Size (MB)
FinBERT	91.0	85	418
BERT-base	89.5	90	418
DistilBERT	88.0	60	268
TinyBERT	86.5	25	55

Table 2.4: Performance of BERT variants on financial sentiment classification [2].

We can easily conclude that TinyBERT is much better for application where small size and fast inference are essential.

While sentiment is useful, the articles can still be fake. Using bad information can hurt the quality of predictions, that's why our system also includes a credibility analyzer. Existing approaches for credibility or fake news detection usually combine source reliability databases (e.g. Fact Check), linguistic features (excessive punctuation or spelling errors), semantic features (like comparing claims to known facts), transformer models. Below, there is a quick comparison of different models capable of performing tasks like sentiment and credibility analysis:

Model Type	Accuracy (avg)	Pros	Cons
Rule-based (VADER, SentiWordNet)	65–75%	Fast, easy to explain	Poor with complex text
SVM	70–78%	Simple, efficient	Needs feature engineering
BERT-based	88–93%	High accuracy, contextual	Needs more resources
FinBERT	90–95%	Best accuracy in finance/news	Large, harder to deploy

Table 2.5: Comparison of model types for sentiment and credibility analysis, information collected from [7, 3, 14, 13, 4, 2]

Overall, using transformer-based models for both sentiment and fake news detection gives more reliable results and by combining their outputs, we can make smarter decisions.

2.3 Limitations

LSTM models rely on historical prices and may miss sudden changes caused by news. Sentiment-based models help, but they often overlook news credibility. Filtering low-credibility sources improves results, but still it is not perfect.

Then comes the issue of combining multiple models to solve these problems. A setup that integrates the features below together can definitely improve prediction performance:

- LSTM for price trends,
- Transformer (like fine-tuned TinyBERT) for reading sentiment,
- Another Fine-tuned TinyBERT for verifying news credibility,

There are still some disadvantages, such as training and inference time, that can increase. A problem that often occurs with large models or datasets is that models can overfit, as described in more details in [5]. None of these models can perfectly capture human emotions, especially in cases of market manipulation, where large groups may buy or sell based on misleading information or fake news.

Considering all the trade-offs discussed above, our setup will include a 2-layered LSTM for price prediction and two fine-tuned TinyBERT models for sentiment and credibility analysis, to achieve more realistic predictions considering more external factors integrated together.

Chapter 3

Theoretical Concepts

3.1 Theoretical Overview

This section will explore in depth the main theoretical concepts behind the development of the AI models in our educational finance app. Since the app integrates deep learning models for stock price prediction, sentiment analysis and fake news detection, these models will be discussed in detail, including their architectures, methodologies and how they work together to enhance predictions.

3.2 Time Series Forecasting for Stock Prices

3.2.1 Recurrent Neural Networks (RNNs)

These models were the first deep learning models used in finance, designed to handle sequences, so they work well with sequential data, where RNNs handle each point in the sequence individually and keep in ”memory” what they’ve seen before using a hidden state:

$$h_t = \tanh(W \cdot x_t + U \cdot h_{t-1} + b) \qquad \textbf{Source:} \text{ GeeksforGeeks}$$

RNNs forget things over time, especially when the sequence is long. That’s because the gradients (used for learning) shrink with each time step, being harder to learn long-term effects [5]. To fix that, Long Short-Term Memory networks (LSTMs) introduce cells that decide what information is retained or discarded over time, which have a state that acts like a connection that allows information to pass through with minimal modification, helping preserve gradients over long sequences. This subsection aims to describe how LSTM works when used for time series forecasting. The mathematical operations involved are as follows:

$$
\begin{aligned}
f_t &= \text{sigmoid}(W_f \cdot [h_{t-1}, x_t] + b_f) && \text{(Forget gate)} \\
i_t &= \text{sigmoid}(W_i \cdot [h_{t-1}, x_t] + b_i) && \text{(Input gate)} \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) && \text{(Candidate memory)} \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t && \text{(New memory)} \\
o_t &= \text{sigmoid}(W_o \cdot [h_{t-1}, x_t] + b_o) && \text{(Output gate)} \\
h_t &= o_t * \tanh(C_t) && \text{(Hidden state)}
\end{aligned} \tag{3.1}
$$

Adapted from standard LSTM formulas in [14].

For a stock price prediction task where the input at time t is $x_t = 1$, LSTM uses two internal memory components from the prior time step: short-term memory (also called hidden state: $h_{t-1} = 0$, for instance price momentum) and the long-term memory (or cell state: $C_{t-1} = 0$, such as ongoing trends)

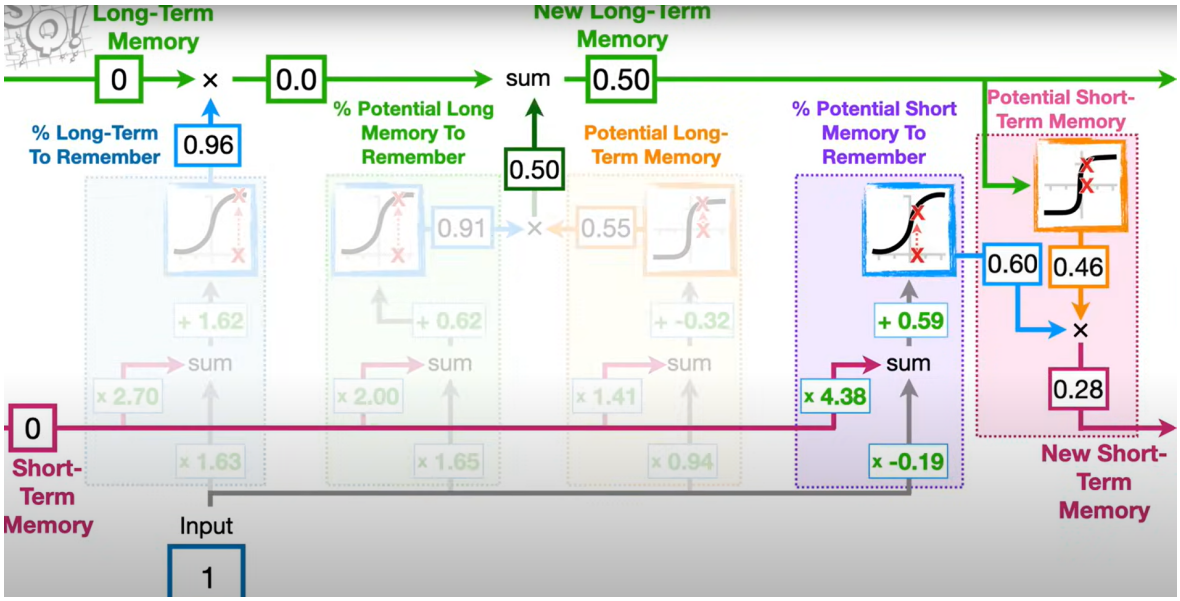

Figure 3.1: Inspired by explanation from StatQuest with Josh Starmer [23]. The input $x_t = 1$ (bottom blue cell representing normalized stock price) is processed along with the initial memory values $h_{t-1} = 0$ (previous hidden state, short-term memory from the last time step) and $C_{t-1} = 0$ (upper left green cell, being the previous cell state, long-term memory capturing historical patterns).

These memory values are used by the LSTM unit to selectively forget, update and output new information. Two key activation functions guide these decisions.

Sigmoid function maps inputs to values in $[0, 1]$. It acts like a soft filter, controlling how much information is passed through. The next is the hyperbolic tangent (tanh) function, mapping inputs to values in $[-1, 1]$ and is typically used to scale the cell state and candidate values. For mathematical definitions of these functions, see

[14].

With these functions defined, we will describe the flow of execution within the LSTM layer. First, the forget gate determines how much of the prior cell state should be removed, so this gate decides how much influence the long-term memory (e.g. knowledge of price movements from previous days or weeks) should continue to have on the prediction at the current time step. Since the information that was kept from the older states depends on the value of C_{t-1} (long-term memory), the forget gate scales it accordingly:

$$\text{Retained memory: } f_t \cdot C_{t-1}$$

Source: [14]

Next, the input gate and the candidate memory collectively regulate what new information is considered for inclusion in the cell's state (see input gate and candidate memory formulas in Eq. 3.1), where the newly added memory is given by:

$$i_t \cdot \tilde{C}_t$$

At each time step, the LSTM integrates today's stock data (like price or volume) into its memory. If the input suggests a strong market move, the input gate i_t opens, allowing new trends to update the memory cell C_t.

The updated cell blends past memory with today's signal. Then, the output gate decides how much of this memory is used to compute the new hidden state h_t, which represents the model's understanding of the current market.

This hidden state is used to predict the next value—like tomorrow's stock price—while the cell state C_t continues storing useful long-term patterns.

The figure below shows how these units connect: each unit receives the current input x_t, the previous hidden state h_{t-1}, and cell state C_{t-1}, and passes updated states h_t and C_t to the next step.

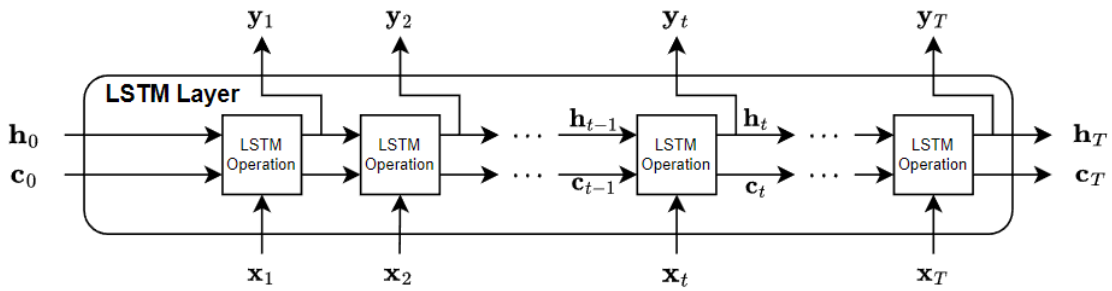

Figure 3.2: Each LSTM unit receives x_t as input together with the previous hidden state h_{t-1} and cell state C_{t-1}, then outputs the updated hidden state h_t and cell state C_t. These states are passed along the sequence to capture temporal patterns in the data. (Adapted from standard LSTM architectures as discussed in [12])

This sequential structure allows the LSTM to combine short-term signals (e.g. daily price shifts) with long-term patterns (e.g. trends or macroeconomic effects). The hidden state h_t can be forwarded to the next time step or used directly for prediction. Together, the outputs $y_1, y_2, \ldots, y_T$ form a sequence of predictions influenced by the full input history.

3.3 Natural Language Processing (NLP)

NLP has a large range of applications, especially in tasks that include text classification (sentiment analysis), translation and summarization. These tasks were boosted by the transformer-based models (like BERT and GPT). This together with very complex embedding layer models are very powerful for understanding prompts when integrated together.

In the following, we will walk through the main concepts behind the architecture of modern NLP models. The table below presents a comparison between several widely used transformer models:

Model	Architecture Type	#Params	Accuracy (%)
BERT-base	Encoder-only	110M	82.1
RoBERTa-base	Encoder-only	125M	83.4
DistilBERT	Encoder-only	66M	79.0
GPT-2	Decoder-only	117M	70–75
T5-base	Encoder-Decoder	220M	82.6
DeBERTa-base	Encoder-only	139M	85.0
TinyBERT	Encoder-only	14.5M	79.1

Table 3.1: Comparison by architecture, parameters and classification performance [4, 9, 18, 17, 6].

3.3.1 Text Classification and Sentiment Analysis

Text classification is a core NLP task that involves assigning predefined categories to textual data. One of the most common forms of text classification is sentiment analysis. The purpose is to be able to extract the sentiment from texts, commonly labeled as positive / negative / neutral. In finance, sentiment analysis is useful to classify various texts, for example:

- "Apple Reports Record Profits This Quarter" $\rightarrow$ Positive
- "Tesla Faces Legal Trouble Over Safety Concerns" $\rightarrow$ Negative

Accurate classification and sentiment analysis require understanding context, something transformer models handle very well.

3.3.2 Transformers Architecture

This architecture was introduced in "Attention Is All You Need" [25] and they create the foundation for modern NLP models. They can model long-range dependencies in text without requiring sequential processing. This is enabled by a mechanism called self-attention and a highly parallelizable architecture (multi-head attention).

The figure below provides an overview of the full transformer architecture, illustrating the flow of information from input tokens to final output generation:

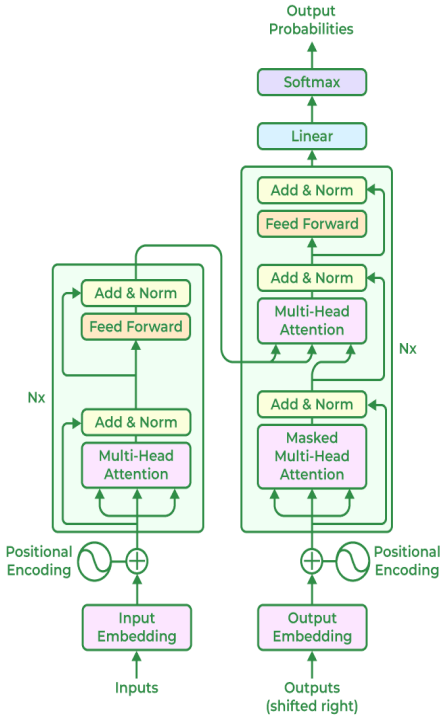

Figure 3.3: Overall Transformer Architecture, from input embeddings to output probabilities, source: erdem.pl

As shown in the diagram, a transformer processes input in the following key steps:

- Convert input into embeddings and add positional encodings.
- Pass through encoder layers with self-attention and feed-forward blocks.
- Decoder applies masked attention and cross-attention using encoder output.
- Final linear layer + softmax outputs predicted token probabilities.

In the next part of this section, we will walk through the transformer's architecture step by step.

Embedding Layer and Positional Encoding

In models having this architecture, raw text cannot be used directly and it must be converted into numerical representations. This conversion happens in two main stages: tokenization and embedding.

Tokenization

Input sentences are first broken down into tokens (typically subword units). For instance, the sentence:

`financialization isn't evolving rapidly`

is tokenized using WordPiece in a similar way as:

`[[CLS], financial, ##ization, is, n't, evolving, rapidly, [SEP]]`

TinyBERT uses the WordPiece tokenizer (from BERT), which splits text into known words or subwords from a pretrained vocabulary [4]. For example, "financialization" becomes "financial" and "##ization". This ensures all inputs are tokenized into recognized units for embedding and model processing.

Embedding Lookup

Once sentences are tokenized, each token is converted into a high-dimensional vector through an embedding matrix:

$$E \in R^{V \times d}, \quad e_i = E[x_i]$$

where V represents size of the vocabulary and d is the embedding dimension ($d = 312$ in TinyBERT) [25].

These dense vectors capture semantic meaning, similar words (e.g. "stock", "market") are placed close to each other in the vector space, helping the model generalize better, even when inputs are noisy or misspelled [4, 8].

$\texttt{king} - \texttt{man} + \texttt{woman} \approx \texttt{queen}$ (Gender relationship)

$\texttt{Paris} - \texttt{France} + \texttt{Italy} \approx \texttt{Rome}$ (Country–capital relationship)

$\texttt{doctor} - \texttt{hospital} \approx \texttt{teacher} - \texttt{school}$ (Profession–location relationship)

BERT's input is a sequence of tokens that has been embedded and combined with positional information. The representation of each input token is obtained by summing three distinct embeddings:

- Token embedding: standard word embedding from a vocabulary
- Segment embedding: distinguishes between sentence A and sentence B for tasks like predicting the next tokens
- Positional embedding: represents token's relative position in the input text

The input representation is a fixed-length vector for each token in the sequence. These vectors are then passed through more encoder layers. TinyBERT's embedding layer is pretrained on large corpora using masked language modeling. Token vectors are learned (not manually set) =to reflect contextual usage, so similar words get similar embeddings [8].

Once trained, this layer serves as the first transformation from raw text into a form suitable for processing by attention mechanisms and deeper layers of the transformer.

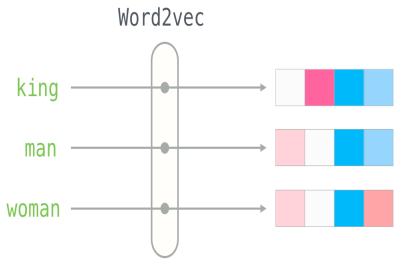

Figure 3.4: Word embedding visualization via Word2Vec (conceptually similar to BERT's embedding space. Source: blog.aiensured)

Semantic Clustering in Vector Space

When visualized (using PCA or t-SNE), embeddings show interesting patterns: semantically related words cluster together. For example, "market", "stock" are typically closer in embedding space, but "banana" if far from them.

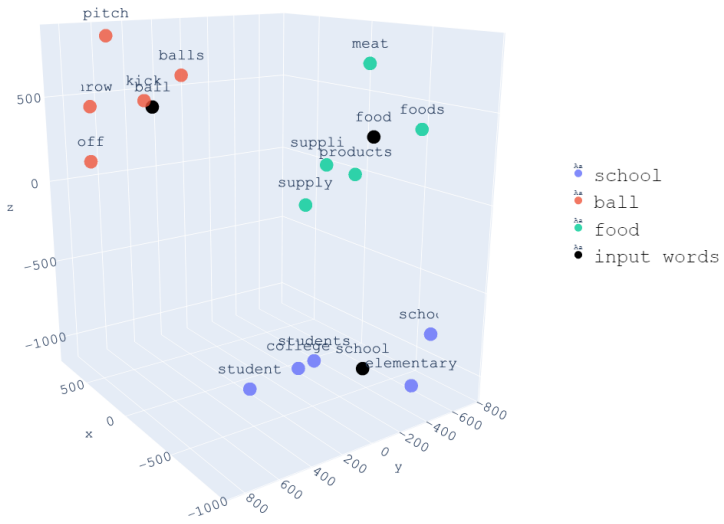

Figure 3.5: Visualization of word embeddings in 3D

Positional Encoding

Transformers have no sense of order, so positional encoding is added to retain the sequence information. The original Transformer uses sinusoidal positional encoding, allows the transformer to attend not only to token meaning (via embeddings) but also to token position in the sentence (see formulas in image 3.6).

Each input vector is passed to the model as:

$$z_i = e_i + \mathrm{PE}_i$$

Each input token is not only represented by its word embedding (semantic meaning) but also by its position in the sequence. To encode this, the Transformer adds a positional encoding to each embedding. These encodings are generated using sine and cosine functions at different frequencies, giving each position a unique pattern.

This helps the model understand word order. For example, even though ”The cat sat” and ”Sat the cat” use the same words, their meaning differs due to word position [25]. The sinusoidal encoding allows the model to detect such differences by giving each word a distinct location signature.

The result is a combined vector that captures both what the word means and where it appears in the sentence. A visual example is shown below:

This method allows the Transformer to reason not only about content but also about the relative and absolute positions of tokens, even without recurrence or convolution.

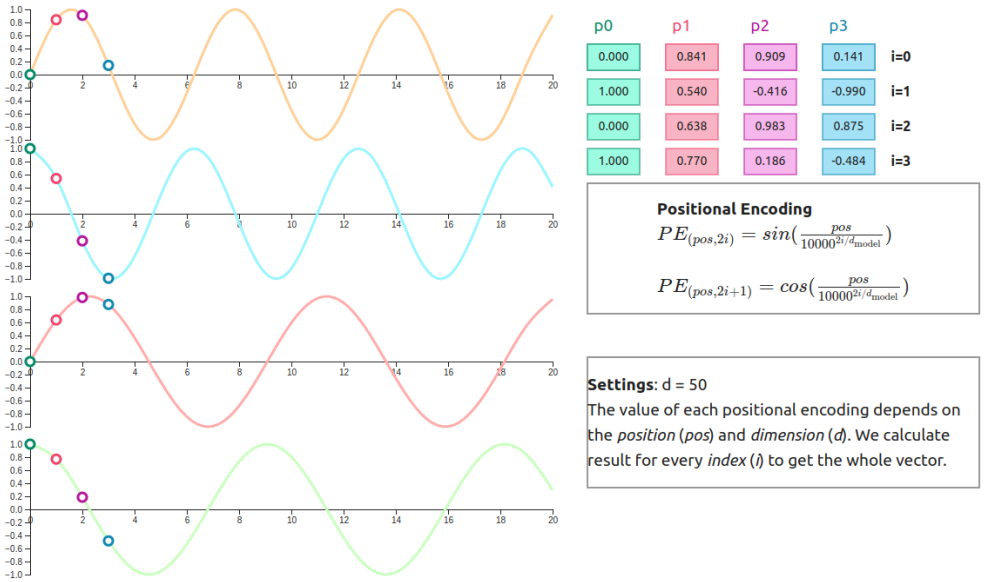

Figure 3.6: Sinusoidal positional encoding using multiple sine and cosine at different frequencies. So in the input sequence, each position receives a distinct encoding vector. Source: erdem.pl

Encoders and Decoders

This transformer is composed of two main components: encoder and decoder. The first one processes the input sequence and maps it to a sequence of continuous vector representations. Then the decoder takes the given representation and generates an output sequence sequentially (one token at a time), considering in its predictions both previous outputs and the encoded input.

Encoders consist of:

- `Multi-head self-attention`
- `Position-wise feed-forward network`
- `Residual connections and layer normalization`

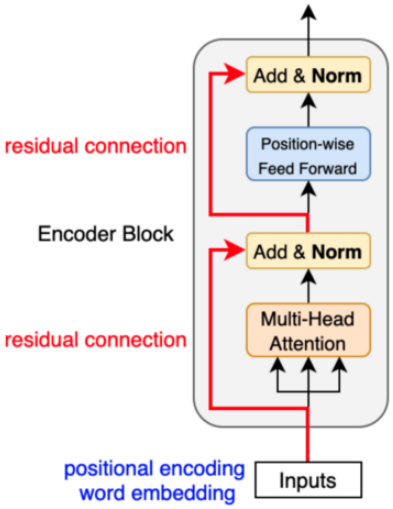

Figure 3.7: Schema of an encoder with its layers **??**naokishibuya.medium.com

The input tokens are first embedded and enriched with positional encodings (shown in blue at the bottom). The combined embeddings are passed into the first sublayer: multi-head self-attention.

Residual connection and Layer Normalization

The output of each sublayer is added to its input (residual connection) to improve training stability. Layer normalization is then applied to stabilize and speed up learning.

Position-wise feed-forward network

Contains 2 linear layers with ReLU between them, at each token position, to add non-linearity and refine representations.

These components form an encoder block, repeated multiple times (e.g. 6–12 layers) in the full transformer encoder. The single difference from the encoder is that the decode includes an additional key component: masked multi-head self-attention to make sure that predictions for a given position are bound to the previous positions. This enforces causal (left-to-right) generation. Additionally, the decoder integrates cross-attention. This allows the decoder to attend to the encoder's output representations.

Each decoder includes:

- `Masked multi-head self-attention` (prevents seeing future tokens)
- `Cross-attention` (attends to encoder outputs)
- `Feed-forward layer with residual connections and normalization`

Unlike the encoder, the decoder uses masking for causal decoding and cross-attention to incorporate encoder context, enabling effective sequence-to-sequence generation.

Self-Attention Mechanism

This mechanism helps the model understand the meaning of each word by comparing it to all other words in the sentence. This way, the model can catch important connections between words, even if they are far apart.

For a given input sequence that is represented as a matrix $X \in R^{n \times d}$, self-attention first maps it into three matrices: queries (Q), keys (K), and values (V), through learned linear transformations:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \qquad \text{Source: [25]}$$

where $W^Q, W^K, W^V \in R^{d \times d_k}$ are parameter matrices learned during training.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad \text{Source: [25]}$$

This formula produces a weighted sum of the value parameter matrices / vectors, where the weights emphasize the similarity of each query with all keys, scaled and normalized via the softmax function.

To make this more concrete, consider the sentence:

"The doctor asked the nurse a question. She said yes."

When computing the self-attention for the word "She", the model must determine what earlier context it refers to. This is done by calculating attention scores between "She" (as the query) and all other words (as keys). The most relevant words, such as "doctor" and "nurse", will receive higher weights, influencing the representation of "She".

Multi-head attention

Instead of using a single attention function, transformers employ multiple attention heads in parallel. Each head operates in a different subspace of the input and captures distinct types of relationships:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \qquad \text{Source: [25]}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \qquad \text{Source: [25]}$$

In the example above, one head may focus on syntactic dependencies (e.g. ”doctor” as subject), while another captures coreference (”nurse” as potential referent for ”she”).

Visual Representations

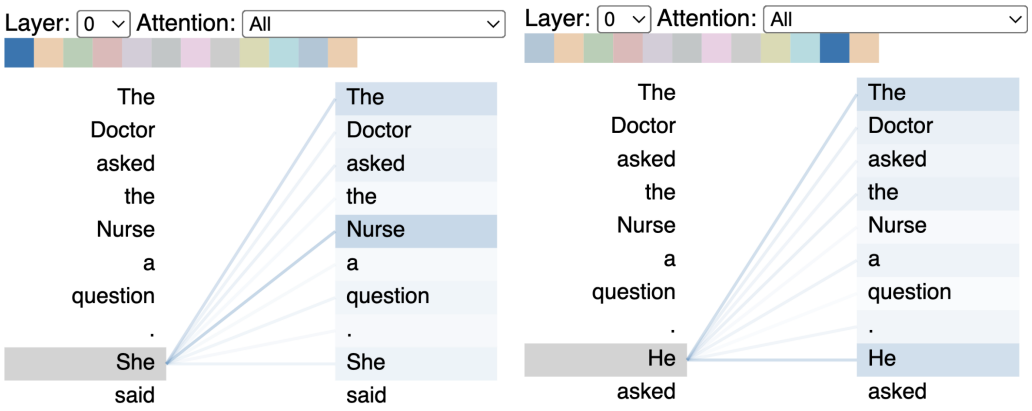

Figure 3.8: Self-attention visualization for the sentence: ”The doctor asked the nurse a question. She/He said...” At layer 0, the token ”She” (left side of the image) strongly attends to ”Nurse”, while ”He” (right) attends more broadly. This shows how attention patterns can vary with subtle lexical changes. Source: comet.com

The figure above illustrates how self-attention operates at the first layer of a transformer model. On the left, the word ”She” exhibits strong attention to the word ”Nurse”, since ”She” is referring to the ”Nurse”. On the right, when ”She” is replaced with ”He”, its attention is shifted more broadly across the sentence, including the word ”Doctor”.

Encoder-Only Architectures: Bidirectional Encoder Representations Transformers

These models are designed for understanding text, not generating it, in contrast with models like GPT that use only decoders designed for text generation (completing sentences, writing stories or code). Instead, BERT is based entirely on the encoder’s architecture, being composed by more encoder layers.

Each encoder layer uses self-attention to compute how strong are the links between each token and the every other token in the sequence. This is done using the self-attention formula discussed in the previous sections (see 3.3.2).

3.3.3 Model Distillation: TinyBERT

TinyBERT (a distilled version of BERT), maintains the same encoder structure with the same mechanisms, including multi-head self-attention and feed-forward layers (see 3.3.2). Distillation compresses the model by transferring knowledge from the larger BERT (teacher) to the smaller TinyBERT (student), aligning their embeddings, attention maps, and hidden states. The process is done in two stages: pretraining and fine-tuning, to preserve both general language understanding and performance on specific tasks [8]. Among its variants, the most widely used configuration is `huawei-noah/TinyBERT_General_4L_312D`, available on Hugging Face. Its main structure can be seen below:

- 4 encoder layers (transformer blocks)
- 312 hidden units per layer
- 12 self-attention heads
- Approximately 14.5 million parameters

The goal is to make the ”student” learn the internal representations of the ”teacher”. To make this work, the ”student” architecture is designed to represent a smaller version of the ”teacher”, preserving the general structure but with fewer layers and smaller hidden sizes. TinyBERT uses a task-aware layer-wise distillation strategy, meaning it learns from a larger model by copying not just the final output, but also the internal layers. First, the BERT-base model is fine-tuned on a specific task / domain, then the ”student” is trained in two stages:

1. *General Distillation:* It learns from the ”teacher” using large amounts of unlabeled text. The goal is to capture general language understanding by matching token-level embeddings, attention weights and hidden states.
2. *Task-Specific Distillation:* the ”student” is further trained using data from a specific downstream task.

During distillation, the deeper layers of the ”teacher” (12 layers in total) are mapped to the fewer layers of the TinyBERT student (typically 4 or 6). This is done through layer mapping strategies, where every few layers of the teacher correspond to one layer of the student. so for a 4-layer TinyBERT, each student layer may be aligned with every third layer of the teacher. This alignment ensures the student still receives high-level semantic signals from the deeper parts of the teacher model, despite its shallower architecture. This allows TinyBERT to retain much of the representational power of BERT-base while being faster and lighter.

3.3.4 Fine-Tuning TinyBERT

The goal of this process is to adapt a pretrained language model, in this case TinyBERT, to a specific task / domain using supervised learning, allowing models to specialize in individual domains by updating their internal parameters using a task-specific dataset. During this process, all layers of the model are made trainable, allowing the model to adjust its internal representations to stick better to the target task.

3.4 News Credibility Analysis

A separate TinyBERT could also be adapted for credibility classification if it is trained on a proper different dataset.

3.4.1 Datasets for Exaggeration and Sensationalism Detection

The LIAR dataset is particularly suitable for fake news and credibility detection due to its fine-grained labels and structured metadata. Below is a brief comparison with other commonly used datasets:

LIAR

- 12,836 short political statements, manually fact-checked.
- 6-level label scale: *pants-fire*, *false*, *barely-true*, *half-true*, *mostly-true*, *true*.
- Focused on statement-level analysis and designed for fine-grained credibility modeling.

FakeNewsNet

- News articles paired with social media engagement labeled as *fake* or *real*.
- Good for studying propagation patterns, less suited for linguistic nuance.

BuzzFeedNews / PolitiFact

- News article-level data with binary or coarse-grained labels.
- Often limited in size or scope.
- Focused more on publisher credibility than statement-level truthfulness.

Chapter 4

Implementation

In this chapter, the development and technical details of the project will be explained, starting with a general overview of the app's main features and details of implementation on both server and client side (functionalities). Next, the chapter walks through the models used to predict stock prices, analyze news sentiment and assess the credibility of news articles.

4.1 Overview of the Educational Finance App Functionalities

4.1.1 Client

The app starts with an authentication panel to ensure secure access to user specific features. New users can register by providing a username and password. The password is securely hashed before being stored and each account is initialized with a virtual balance (10000). Once registered, users can log in by submitting their credentials. If verified, the system generates a Json Web Token (JWT), which allows the user to remain authenticated for a limited period. This token must be included in future requests to access protected resources. Besides the home page that contains the user details, there are also the 4 main sections. First contains an overview of the app and links to public resources that contain relevant financial materials such as courses. ”Quiz” and ”Demo Investing” are the 2 next main sections.

Quizzes and Portfolio Management

The ”Quiz” section provides users with the opportunity to earn virtual money as a reward, depending on the difficulty level of the solved quiz. The questions are generated using the Gemini API through prompt engineering: 10 multiple-choice

questions are created, each with four answer options and a flag indicating which answer is correct. They cover a variety of financial literacy related topics.

Initially, generating questions sequentially resulted in noticeable delays (approx. 15 seconds for 10 questions), so we reduced significantly the response time (3-4 seconds), by using a concurrent approach:

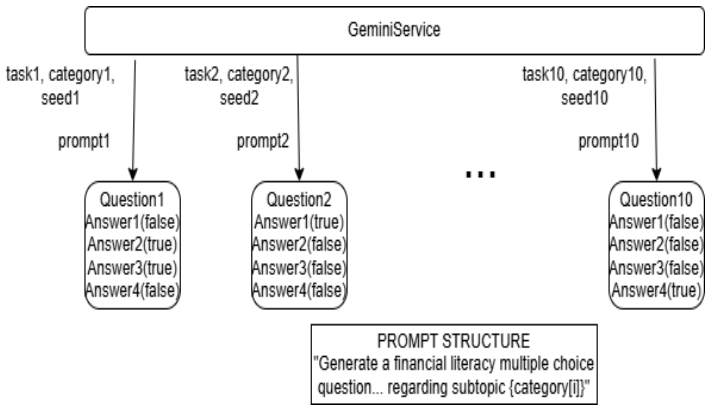

Figure 4.1: Launching 10 parallel tasks for faster quiz generation

The demo investing feature lets users simulate trading with virtual money. Users can search stocks by ticker substring, view live prices (via `yfinance` and `yahooquery`) and buy or sell shares by quantity. A dedicated portfolio page displays owned stocks, quantities, current values and average buy prices. All operations depend on the user's available virtual funds.

Price Prediction Page

The last main page contains the stock price prediction model (its implementation will be discussed in the later sections). Its predicted price will finally be adjusted depending on the sentiment extracted from the news analyzer and credibility score from the credibility analyzer.

4.1.2 Server

As backend we chose FastAPI, a modern high-performance Python web framework designed for building APIs fast and efficiently. This framework leverages Python type hints for automatic data validation and serialization and supports asynchronous programming, enabling it to handle high-concurrency workloads effectively.

To highlight FastAPI's performance advantages, the following table compares metrics across popular Python web frameworks:

Framework	Requests per Second	Average Latency
FastAPI	45,000–50,000	10–15 ms
Flask	30,000–40,000	20–25 ms
Django	35,000–45,000	15–20 ms

Table 4.1: Performance comparison of Python web frameworks. Source: [24].

As shown, FastAPI outperforms Flask and Django in both throughput and latency, primarily due to its asynchronous capabilities and efficient request handling.

For providing a general overview of the app, we start by providing below the architecture diagram of the backend structure:

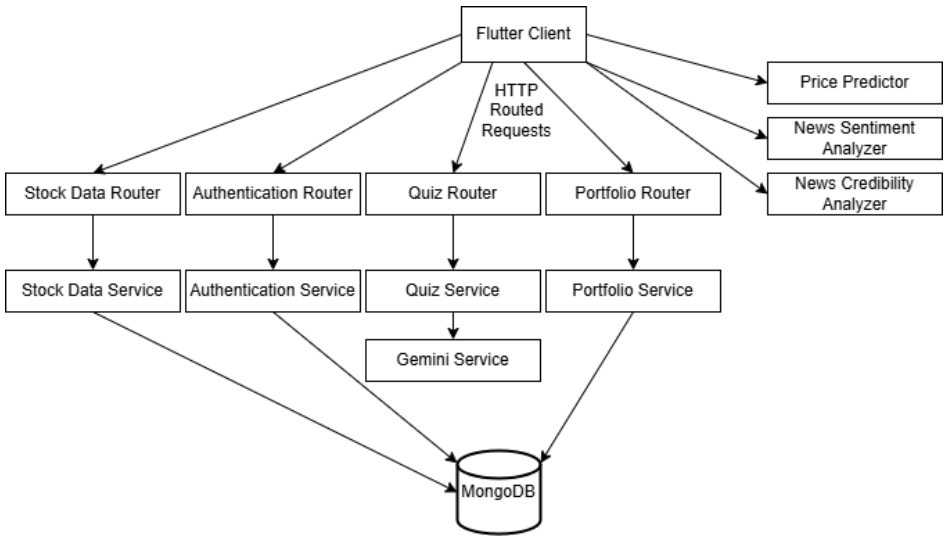

Figure 4.2: Architecture Diagram

In general, class diagrams help to visualize the interactions between the classes in an application. This design facilitates seamless data flow and interaction between the client and server components, ensuring that each part of the application is modular and scalable. The following class diagram shows the relationships and design choices made to structure the backend functionality. It highlights the key classes and their methods, giving a clearer picture of the application's internal workings:

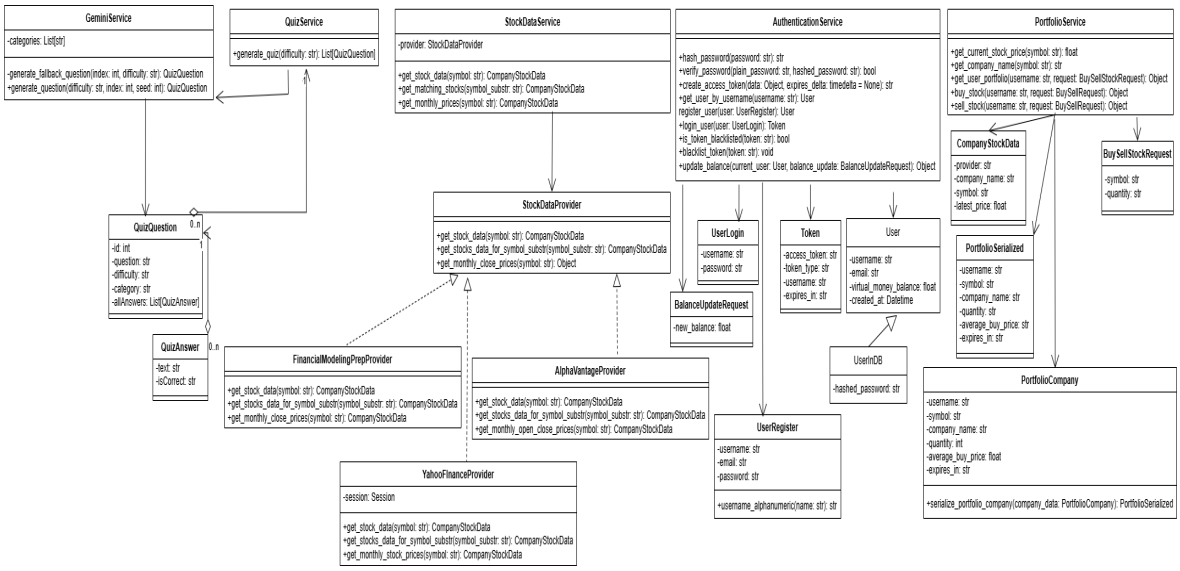

Figure 4.3: Class Diagram

4.1.3 Fetching Stock Data via APIs

The backend integrates three providers: Yahoo Finance, Alpha Vantage and FinancialModelingPrep. Data is fetched either for a single stock symbol or for multiple stocks matching a substring of the ticker. For Yahoo Finance, the yfinance and yahooquery Python packages are used to allow retrieval of daily prices and monthly close data. Alpha Vantage and FinancialModelingPrep are queried directly via HTTP requests, using API keys. The provider is selected by the user in the client side.

4.2 Price Prediction (Time Series Forecasting)

The steps include data collection, preprocessing, model design and training. The goal is to predict the closing prices of a stock for the next 10 days using daily data from the last 2 years.

4.2.1 Data Collection and Preprocessing

Data Sources and Train-Test Splitting Strategy

The data used for prediction is collected using the yfinance and yahooquery. YFinance is a tool that lets you download historical market data (e.g prices, dividends) directly from Yahoo Finance. Yahooquery also accesses Yahoo Finance data, but via Yahoo Finance API endpoints. Their responses differ, yfinance retrieves data in DataFrames, while yahooquery in JSON and this is what we will use as dataset.

The dataset gets split into training and validation subsets using an 80-20 split. No shuffling is performed to preserve the chronological order of the data.

Feature Selection

Out of all features, we use closing price and trading volume.

Closing price	Earnings per share (EPS)
Trading volume	Dividend yield
Opening price	Income statements
Daily high and low	Balance sheet data
Previous close	Analyst recommendations
Market capitalization	Company profile information
Price-to-earnings (P/E) ratio	

Sequence Generation (Sliding Window)

To prepare the data for supervised learning, a sliding window approach is used. Each input sequence contains 20 consecutive days (time steps) of price and volume data, and the target is the closing price on the 21st day.

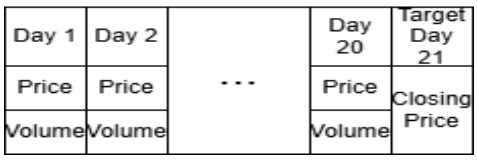

Figure 4.4: Representation of sequence generation, the choice of 20 was made to provide enough historical context for the model without making the sequences too long, which can increase computational complexity and overfitting risk. For generating 22nd day, days in range 2 - 21 are considered

Normalization and Scaling

Before training, all values are scaled using `MinMaxScaler` to fall within the range [0, 1]. This helps the model converge faster and ensures that each feature contributes proportionally to the loss. This is done by using:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

(Adapted from standard Min-Max normalization, e.g. `sklearn.preprocessing.MinMaxScal`

Supposing 3 closing prices \$150, \$200 and \$250, if the minimum price in the dataset is \$100 and the maximum is \$300, then their normalized values would be like:

- \$150: $\frac{150-100}{300-100} = \frac{50}{200} = 0.25$
- \$200: $\frac{200-100}{200} = 0.5$

•\$250: $\frac{250-100}{200} = 0.75$

4.2.2 Model Architecture

LSTM with Two Layers

The model includes two stacked LSTM layers. The first LSTM layer receives the input sequences and outputs hidden states. A dropout layer with a rate of 0.3 follows to reduce overfitting (dropout will be discussed in the next subsection). The output is then passed to a another LSTM layer, followed by another dropout layer with a rate of 0.2 / 20%. The last layer is a fully connected (linear) layer that maps the final output to a single predicted closing price.

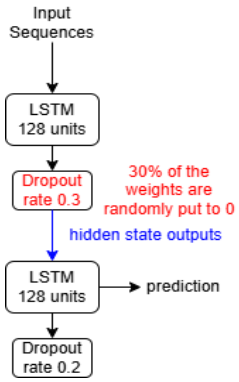

Figure 4.5: The architecture of our LSTM price prediction model

Regularization Techniques

Dropout is a widely used technique for regularization, where a fraction of neurons are randomly disabled during training. Regularization prevents the network from relying too much on any specific set of neurons. In our model, dropouts have rates 0.3 and 0.2, chosen to balance between regularization and retaining enough learning capacity.

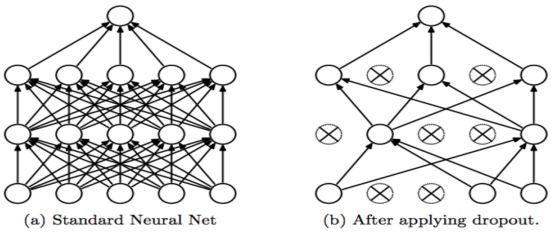

Figure 4.6: Representation of how dropout disables random neurons in a network
medium.com/dropout-in-deep-learning

4.2.3 Training the Model

Training Process

Figure 4.7: Representation of the training flow (aims to minimize the Mean Squared Error (MSE) between predicted and actual prices)

How Prediction Works

For each prediction, the model takes the last 20 days of closing price and volume as input (see 4.4) and then predicts the next day's closing price. For multi-step prediction, this predicted value is added to the sequence (along with the last known volume), and the window is shifted forward. This new sequence is then used to predict the next day's price, and so on. This autoregressive approach continues iteratively for as many days as needed.

4.3 Sentiment Regressor For Financial News

4.3.1 Data Collection and Preprocessing

Financial Phrase Bank Dataset

The Financial Phrase Bank [11] is a benchmark dataset for sentiment analysis in finance, containing 4849 English sentences from financial news labeled as positive, negative or neutral by business students.

It includes multiple agreement levels (50%, 75%, 100%), with the 100% subset being the most reliable (2270 samples) and the 50% one providing the full dataset but with more label noise. We used the 50% agreement level to leverage more training data for our TinyBERT model.

- **50% agreement**: 4849 samples — more data, but moderately noisy labels
- **75% agreement**: 3716 samples — balanced trade-off
- **100% agreement**: 2270 samples — very high-quality labels, but limited data

Overall, the 50% agreement dataset provides a better foundation for training robust models. we let below some insights about the 50% agreement subset:

- **Total samples:** 4,849

- **Positive:** 2,216 samples (45.7%)
- **Negative:** 1,242 samples (25.6%)
- **Neutral:** 1,391 samples (28.7%)

We chose this dataset to train and evaluate our fine-tuned TinyBERT sentiment regressor because it contains high-quality news labeled from a financial sentiment perspective, including the kind of financial terms and expressions commonly found in market news.

Custom Dataset With Financial Words, Phrases and Scores

In addition to the existing dataset, we developed a custom dataset focused on financial terminology by writing a Python script to generate labeled sentiment examples. The goal was to create a rich set of financial words / phrases, helping the TinyBERT sentiment regressor learn more precise associations between financial language and sentiment polarity.

These words and phrases are categorized as positive, negative or neutral. They are labeled by 3 functions regarding their sentiment (positives labeled randomly in [0.6, 0.8], negatives randomly in [-0.8, -0.6], neutrals in [-0.2, 0.2]), allowing the regressor to produce more precise sentiment predictions on individual words.

To improve the richness and realism of the data, we defined some other categories:

- Intensifiers (e.g. “significant”, “huge”) were combined with all words to simulate stronger sentiment (their scores were accordingly amplified in [0.8, 0.97], [-0.97, -0.8]).
- Negations (e.g. “not”, “never”) were applied on each word to invert sentiment (e.g., “not profitable” flips from 0.7 to -0.7 + random(-0.1, 0.1) to apply a tiny randomness).
- Tricky phrases (e.g. in tricky positives “decrease in expenses”).
- All phrases were lemmatized using spaCy to ensure consistent representation of different grammatical forms (e.g., “rising” $\rightarrow$ “rise”).

This approach resulted in a custom dataset of over 10 thousands of entries to introduces a flexible scoring and cover nuanced sentiment across individual words.

4.3.2 Model Architecture

TinyFinBERTRegressor Design

The core model architecture is based on TinyBERT (smaller variant of BERT, see 3.1). We adapted this architecture for regression by adding a custom attention-based pooling mechanism and a linear output layer to predict a continuous sentiment score.

After processing the input tokens, TinyBERT outputs contextual embeddings for each token in the sequence. The embeddings are then sent through an attention pooling layer that learns to focus on sentiment-relevant tokens. The resulting pooled vector is then passed to a fully connected linear layer that outputs a scalar sentiment score and then tanh activation function is applied on it to map it in [-1, 1].

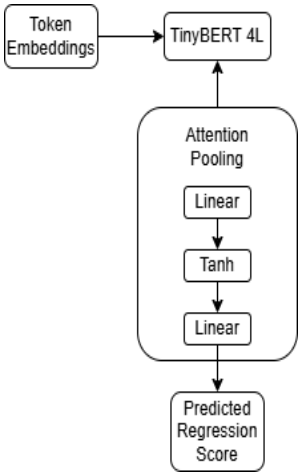

Figure 4.8: Architecture of the TinyFinBERT Regressor

Attention Mechanism for Weighted Pooling

Instead of relying on the [CLS] token or mean pooling, we introduce a trainable self-attention pooling layer that computes attention scores over all token embeddings. This is implemented as a two-layer feedforward network with a $\tanh$ nonlinearity followed by a scalar projection:

$$\text{score}_i = \text{Linear}_2(\tanh(\text{Linear}_1(h_i)))$$

The resulting scores are masked and normalized with a softmax function to produce weights for each token. These are then used to compute a weighted sum over all token embeddings:

$$\text{pooled\_output} = \sum_{i=1}^{n} \text{softmax}(w_i) \cdot h_i$$

This attention pooling allows the model to emphasize contextually important tokens—such as negations, intensifiers, or financial keywords—that most strongly influence the overall sentiment signal.

4.3.3 Fine-Tuning Strategy

Training Process and Implementation

The model is fine-tuned using the Hugging Face Transformers and Datasets libraries. The training process involves preprocessing, tokenization, dataset preparation, model setup and iterative optimization using the `Trainer` API.

Preprocessing and Tokenization:

- Text is preprocessed (lowercased, lemmatized) using a custom function.
- A TinyBERT tokenizer (`huawei-noah/TinyBERT_General_4L_312D`) converts sentences into subword token IDs.
- Inputs are padded/truncated to a maximum length of 128 tokens.
- The tokenizer outputs both token IDs and attention masks.

Dataset Preparation:

- Preprocessed `train_df` and `test_df` (and optional `extra_df`) are combined and mapped into Hugging Face `Dataset` objects.
- A tokenization function maps text to features and attaches each corresponding sentiment score as the label.

Model Configuration and Objective:

- We use a custom regression model based on TinyBERT (`TinyFinBERTRegressor`).
- The output is a single scalar sentiment score.
- Since this is a regression task, Mean Squared Error (MSE) is used as the loss function.

Training Arguments:

- `TrainingArguments` defines all key hyperparameters:
  - Batch size: 16 (train) / 64 (eval)
  - Epochs: 5
  - Learning rate: 2×10^{-5}, weight decay: 0.01
  - Evaluation and saving strategy: every epoch
  - Best model loaded based on lowest validation loss

Training Loop:

- The `Trainer` API handles training, evaluation and saving.
- At each step:
  - The model receives tokenized batches.
  - Predicts sentiment scores.
  - MSE loss is computed and minimized.
  - Metrics (MSE and R^2) are tracked.
- After training, the best model is saved along with the tokenizer.

Saving and Loading Model Weights

After training, the model's weights and tokenizer are saved using `torch.save()` and `save_pretrained()`. For later inference or evaluation, the model is reloaded with `torch.load()` and the tokenizer is restored from the saved files. This avoids retraining and allows quick reuse for fast prediction.

Inference and Predictions

The tanh output ensures score range is in [-1, 1], indicating the model's prediction in the sentiment of the input.

After training, the model's weights and tokenizer are saved using `torch.save()` and `save_pretrained()`. For later inference or evaluation, the model is reloaded with `torch.load()` and the tokenizer is restored from the saved files. This avoids retraining and allows quick reuse for fast prediction.

4.4 Credibility Regressor for News Statements

4.4.1 Data Collection and Preprocessing

LIAR Dataset

The LIAR dataset is a large-scale benchmark corpus for credibility assessment. It contains over 12,000 short statements from various contexts (e.g. political debates, interviews), each labeled as one of these categories: pants-fire, false, barely-true, half-true, mostly-true or true. Since our model is a regressor, we mapped them to numerical scores in the [0.0, 1.0] range: pants-fire $\rightarrow$ 0.0, false $\rightarrow$ 0.2, barely-true $\rightarrow$ 0.4, half-true $\rightarrow$ 0.6, mostly-true $\rightarrow$ 0.8, true $\rightarrow$ 1.0 (greater scores induce greater credibility).

Custom Financial News Dataset

Besides the LIAR dataset, we created a custom dataset containing words and phrases labeled with credibility scores using a Python script. These words and phrases are labeled by 5 functions with a random score from their corresponding interval: [0.01, 0.2], [0.2, 0.45], [0.45, 0.55], [0.55, 0.8], [0.8, 0.99] (e.g. "anonymous" $\rightarrow$ random score in [0.01, 0.2], while "verified" $\rightarrow$ random score in [0.8, 0.99]). To balance the final dataset, the custom dataset is duplicated and combined with LIAR.

4.4.2 Credibility Regressor Architecture

This model also extends TinyBERT to generate contextual embeddings, followed by multi-head attention and attention pooling to compute a weighted sentence representation.

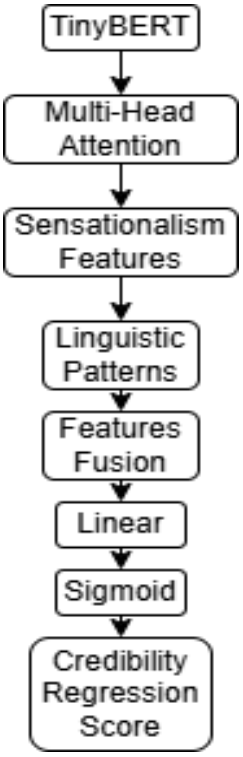

Figure 4.9: Architecture of the Credibility Regressor Model

In contrast to the sentiment analyzer, this model combines three feature vectors:

- **Semantic content:** contextual output from TinyBERT
- **Sensationalism:** exaggeration patterns via FFN
- **Emotional intensity:** focus on emotionally charged language

Compared to the Sentiment Analyzer:

- Uses **multi-head inputs** beyond sentiment (structure, emotion, exaggeration)
- Final score **blends model + grammar + punctuation** signals
- Uses **sigmoid output** for score in $[0, 1]$ instead of tanh (since our sentiment analyzer outputs in [-1, 1])

Grammar and Punctuation Scores

To support credibility assessment, additional language quality scores are computed:

1. **Grammar Score:** Computed using the `language_tool_python` package, which detects grammar issues in the input text (returned in a list called "matches"). The score is calculated by penalizing the ratio of detected grammar errors per word:
$$S_{\text{grammar}} = \max(0.05,\ 1 - \lambda \cdot \frac{E}{N})$$
where E is the number of grammar errors, N is the number of words, and λ is a penalty factor (e.g., 3.0). A higher error rate leads to a lower score.

2. **Punctuation Score:** Based on the presence of exaggerated punctuation patterns (e.g., "!!!", "??", "..."). The score is computed by penalizing the ratio of such patterns relative to the text length:
$$S_{\text{punct}} = \max(0.05,\ 1 - \mu \cdot \frac{P}{N})$$
where P is the number of exaggerated punctuation matches, N is the number of words, and μ is the penalty factor (e.g., 1). This discourages overly emotional or sensational punctuation.

To get the final credibility score, they are weighted together:

$$\text{Credibility Score} = \alpha \cdot S_{\text{punctuation}} + \beta \cdot S_{\text{grammar}} + \gamma \cdot S_{\text{classifier}} \tag{4.1}$$

where:

$S_{\text{grammar}} \in [0, 1]$: Score based on spelling and grammar

$S_{\text{punctuation}} \in [0, 1]$: Score based on possible excessive punctuation

$S_{\text{regressor}} \in [0, 1]$: Score from our TinyBERT regressor,

α, β, γ: Weights that add up to 1 (we took them as 0.5 for regressor, 0.25 for both punctuation and grammar).

The result is a credibility score between 0 and 1, to be multiplied with the sentiment score:

$$S_{\text{final}} = S_{\text{sent}} \cdot S_{\text{cred}} \tag{4.2}$$

This way, unreliable news has less influence, while still keeping useful signals from trusted sources. For example, a well-written article from an unknown site flagged as fake gets a lower score, while an article from a known, trustworthy source gets more weight.

4.4.3 Fine-Tuning Strategy

Training Process and Implementation

The model is trained using PyTorch with a combination of the LIAR dataset and custom financial news data. The training pipeline includes dataset loading, tokenization, model definition, training loop, and evaluation.

Dataset Loading and Label Mapping:

- The LIAR dataset downloaded via Kaggle and filtered to retain only valid credibility labels.
- Labels are mapped to continuous credibility scores in $[0.0, 1.0]$ using a custom scale.
- A custom financial dataset is loaded and duplicated to increase its weight in training.
- The two datasets are concatenated to form a balanced training set.

Preprocessing and Tokenization:

- Statements are tokenized using TinyBERT's tokenizer (`huawei-noah/TinyBERT_Gener`
- Sentences are padded or truncated to a maximum of 128 tokens.
- Tokenized outputs include token IDs, attention masks, and attached credibility labels.

Dataset Preparation:

- Tokenized samples are converted into Hugging Face `Dataset` format.
- The dataset is split into training and validation sets with an 80/20 ratio.
- `collate_fn` is used to construct input batches as PyTorch tensors.

Model Configuration and Objective:

- `CredibilityRegressor` has a custom architecture based on TinyBERT with multi-head attention and specialized feature branches.
- It outputs a scalar credibility score in $[0, 1]$ using a final sigmoid activation.
- The loss function is a weighted sum of Mean Squared Error and Huber loss to balance outlier robustness and sensitivity.

Training Arguments:

- The model is trained using the AdamW optimizer with:
  - Batch size: 32
  - 10 Epochs (more need comparing to the 5 epochs of sentiment regressor because of the complexity of this credibility model: multiple specialized heads: sensationalism, emotion, linguistic structure)
  - Learning rate: 2×10^{-5}, Weight decay: 0.01 (avoid overfitting minima)
- Cosine Annealing Warm Restarts are used as a scheduler to adjust learning rate dynamically.
- Gradient clipping with max norm 1.0 is applied for training stability.

Training Loop:

- For each epoch:
  - Batches are passed through the model to compute predictions and loss.
  - Gradients are backpropagated, the scheduler updates the learning rate.
- After each epoch, the model is evaluated on the validation set using the same loss function.

Saving and Loading Model Weights

After training, the model's state dictionary and tokenizer is saved using `torch.save()` and `save_pretrained()`. If saved weights are detected on future runs, they are loaded to skip retraining using torch.load().

4.4.4 Inference and Predictions

The sigmoid output ensures score range is in [0, 1], indicating the model's confidence in the credibility of the input.

4.5 Adjusting Price Regarding Sentiment and Credibility Weighted Score

In the stock prediction pipeline, sentiment alone is not sufficient to adjust price forecasts. Instead, a combined score is calculated by multiplying the sentiment score (ranging from -1 to 1) with the credibility score (ranging from 0 to 1). This ensures that highly emotional but non-credible news has a reduced influence on predicted prices, while credible and strong sentiment can meaningfully adjust forecasts. For example:

If sentiment = 0.9 and credibility = 1.0, the combined score is $0.9 \rightarrow$ strong upward adjustment.

If sentiment = -0.8 and credibility = 0.5, the combined score is $-0.4 \rightarrow$ weaker downward influence due to moderate credibility.

If sentiment = -1.0 but credibility = 0.0, the combined score is $0.0 \rightarrow$ no adjustment is applied.

The combined score is passed to an adjustment function which modifies the predicted prices across multiple days using an exponentially decaying effect. The decay factor (default 0.5) reduces the influence of sentiment with each passing day. That is, the first day's adjustment is the strongest, the second is half as strong, the third is a quarter as strong, and so on.

The adjustment factor applied each day is calculated as:

$$\text{adjustment\_factor}_i = 1 + \left(\frac{\text{combined\_score}}{10}\right) \times \text{decay}^i \tag{4.3}$$

This factor is then used to modify the predicted price. For example, with a combined score of 0.6 and a decay rate of 0.5:

- Day 1: $1 + (0.6/10) \times 1.0 = 1.06$
- Day 2: $1 + (0.6/10) \times 0.5 = 1.03$
- Day 3: $1 + (0.6/10) \times 0.25 = 1.015$

These multiplicative adjustments are applied progressively. If the sentiment is negative, the adjustment factor becomes less than 1, leading to a decreasing predicted price.

An additional mechanism ensures that if the adjusted trend contradicts the original predicted trend (like switching from increase to decrease), the adjustment factor

is inverted. This correction ensures trend consistency and avoids generating unrealistic reversals in market behavior due to noise.

If no combined score is provided (i.e., score is zero or not computed), the model skips sentiment-based adjustment entirely. The predictions then reflect only the outputs of the LSTM model and volatility simulation, maintaining robustness in absence of valid sentiment or credibility signals.

After sentiment adjustment, a final stage applies simulated daily volatility using a small random percentage in the range [0%, 1%], while preserving the original directional trend of the price, to mimic real market behavior by adding small fluctuations without changing the overall predicted movement.

Chapter 5

Results and Conclusion

5.1 Evaluation Metrics for Each Model

Below, we outline the evaluation results for each of our models.

5.1.1 Time Series Forecasting

To evaluate the performance of our time series forecasting model, we used four key metrics:

- **MAE (Mean Absolute Error)** – measures the average magnitude of prediction errors, useful for interpretability in real price units.
- **MAPE (Mean Absolute Percentage Error)** – indicates prediction accuracy as a percentage, easier to interpret across stocks.
- **Directional Accuracy** – computes the percentage of times the model correctly predicts the direction (up/down), valuable for investment decisions.
- **R^2 Score (Coefficient of Determination)** – shows how well the model explains the variability in the target data.

These metrics offer a good overview of trend consistency, for further reading on time series evaluation metrics, see [10]. The metrics we have obtained for our model are as follows:

AAPL:

- MAE: $18.80
- MAPE: 8.73%
- Directional Accuracy: 52.6%

- R^2 Score: 0.7322

S&P 500 (^GSPC):

- MAE: $273.18
- MAPE: 4.78%
- Directional Accuracy: 54.6%
- R^2 Score: 0.70

While AAPL shows good variance explanation, ^GSPC demonstrates lower error and better directional accuracy, due to its stability as an index.

5.1.2 Sentiment Regressor

To assess the performance of different versions of the TinyBERT model on financial sentiment regression, we evaluated three configurations:

- **Base TinyBERT (no fine-tuning)**, used as it is + a regression head, without any specific training.
- **TinyBERT fine-tuned on Financial Phrase Bank only (50% agree subset)**.
- **TinyBERT fine-tuned on both the Financial Phrase Bank and our custom dataset of financial words, phrases and contextual sentiment scores**.

The base TinyBERT model was included to serve as a baseline, to show the contribution of fine-tuning to sentiment prediction. Since the model was trained to output continuous sentiment scores, evaluation was conducted by making these outputs discrete (splitting into categorical sentiment classes: negative / neutral / positive). The predicted sentiment score $\hat{y}$ was classified using the following rule:

$$\text{label} = \begin{cases} -1 & \text{if } \hat{y} < -0.3 \\ 0 & \text{if } -0.3 \leq \hat{y} \leq 0.3 \\ 1 & \text{if } \hat{y} > 0.3 \end{cases}$$

The true labels were rounded to the nearest class (-1, 0, or 1), allowing classification-based metrics to be applied to the regression task. For a better performance comparison, the models were evaluated using both regression and classification metrics:

- **Mean Squared Error (MSE)**: Captures the average squared difference between predicted and actual sentiment scores. Lower values indicate better regression performance.

•**R-Squared (R^2)**: How well the model explains variance in sentiment scores.

•**Accuracy**: Proportion of correct sentiment predictions.

•**Precision, Recall, F1 Score**: Evaluate classification performance by measuring correctness (precision), completeness (recall), and their harmonic mean (F1), especially important for imbalanced class distributions.

•**ROC-AUC**: Assesses model's ability to distinguish between sentiment classes.

•**Cohen's Kappa**: Evaluates agreement between predicted and true labels, accounting for agreement by chance.

For detailed descriptions of these evaluation metrics in the context of sentiment analysis and regression, see Rosenthal, Farra, and Nakov [20].

Model Comparison

Table 5.1: Performance Comparison of TinyBERT Variants

Metric	Base TinyBERT	Phrase Bank FT	Both Datasets FT
MSE	0.4194	0.1672	**0.1645**
R^2	-0.1116	0.5568	**0.5640**
Accuracy	0.5835	0.7887	**0.8021**
Precision	0.3453	0.8054	**0.8120**
Recall	0.5835	0.7887	**0.8021**
F1 Score	0.4338	0.7913	**0.8042**
ROC-AUC	0.4976	0.8424	**0.8507**
Cohen's Kappa	-0.0065	0.6349	**0.6550**

Confusion Matrices

They provide a detailed breakdown of a classifier's performance by showing how often predicted labels match true labels for each class. [15] provides a more detailed analysis in this topic.

array

Table 5.2: Confusion Matrices for Each TinyBERT Variant

Base TinyBERT

$$\begin{bmatrix} 0 & 110 & 0 \\ 5 & 566 & 0 \\ 0 & 289 & 0 \end{bmatrix}$$

Phrase Bank Fine-Tuned

$$\begin{bmatrix} 88 & 15 & 7 \\ 31 & 433 & 107 \\ 4 & 41 & 244 \end{bmatrix}$$

Fine-Tuned (Both Datasets)

$$\begin{bmatrix} 92 & 12 & 6 \\ 41 & 452 & 78 \\ 4 & 51 & 234 \end{bmatrix}$$

The results show a clear performance gap between the base and fine-tuned models. Base TinyBERT failed to classify any positive or negative examples, predicting only neutral sentiment. In contrast, both fine-tuned variants performed substantially better, with the model trained on both datasets achieving the highest overall

accuracy and class balance. This highlights the importance of domain-specific fine-tuning and also of the custom dataset with scored financial words and phrases for inducing regression in prediction, since Phrase Bank has only negative / neutral / positive labels.

5.1.3 Credibility Regressor

Evaluation was conducted on a combination of the LIAR dataset and our custom credibility words dataset. Over 10 training epochs, our regressor showed stable convergence with training loss decreasing from 0.0570 to 0.0130.

Since the model outputs continuous credibility scores between 0 and 1 and every small change in this range can impact interpretation, we split the validation data into 5 credibility bins:

MAE per Credibility Bin:

- 0.0–0.2: 0.0459
- 0.2–0.4: 0.3146
- 0.4–0.6: 0.2598
- 0.6–0.8: 0.1603
- 0.8–1.0: 0.1840

The model was evaluated on a validation set comprising unseen examples from our 2 datasets. This ensures that evaluation reflects generalization to new samples, not memorized training data.

This evaluation was performed using only the model's output, without considering grammar or punctuation. These linguistic features are more relevant for assessing writing quality or realism, while the LIAR dataset focuses primarily on misinformation, exaggeration and sensationalism, which are not necessarily linked to grammar or punctuation mistakes.

Evaluation Metrics (on model output only):

- MAE: 0.1140
- MSE: 0.0360
- RMSE: 0.1898
- R^2: 0.5689

The model performs best at identifying clearly non-credible statements (low bin errors), but is less reliable on borderline cases. This means that a much larger custom dataset could induce more regression to help the model understand these specific cases even better.

The following visualizations illustrate model behavior:

- **Actual vs Predicted Scores**: Shows score alignment and cluster behavior.
- **Prediction Error Distribution**: Highlights residual bias.
- **Predicted Score by Credibility Bin**: Displays prediction spread across ground-truth ranges.

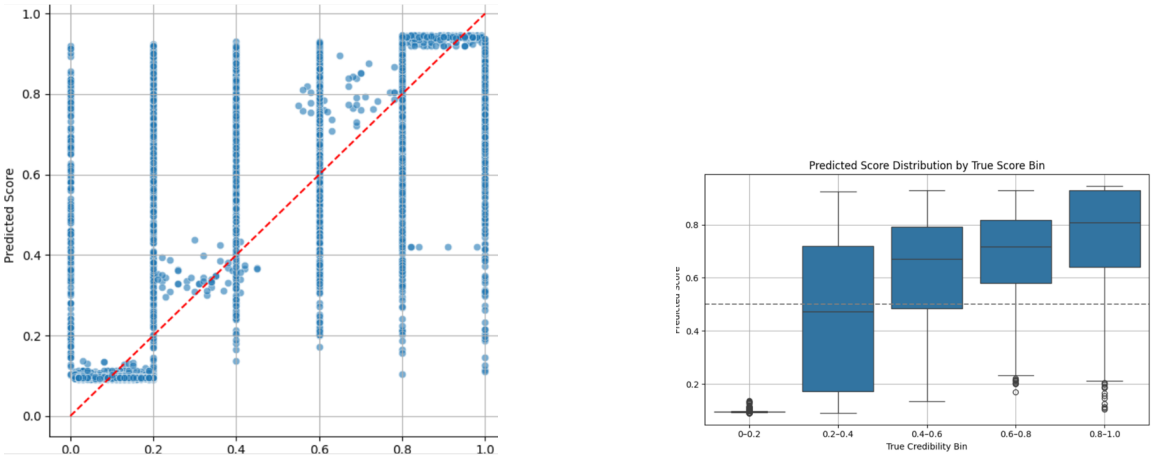

Figure 5.1: Model prediction evaluation: predicted vs actual (left), prediction spread by bin (right).

5.1.4 Overall Adjusted Price Prediction

To evaluate our prediction model under a realistic scenario, we analyzed Target Corporation after a major news report published on May 21 2025. The article reported a 5.2% drop in stock price following disappointing Q1 earnings, DEI policy backlash, and the impact of new tariffs. The full article read:

> "In the first quarter of 2025, Target Corporation reported a significant drop in sales, with comparable sales falling by 3.8%. This decline was attributed to multiple factors, including economic uncertainty stemming from new tariffs and backlash over changes to the company's diversity, equity, and inclusion (DEI) policies. Target scaled back several DEI initiatives in January, leading to customer boycotts and reduced spending. Additionally, the company's reliance on imports from China made it more susceptible to tariff impacts. As a result, Target revised its annual sales forecast downward. The company's stock responded by falling 5.2%, closing at $93.01 on May 21, 2025."

The sentiment model assigned the article a score of **-0.738**. The credibility components were grammar: 1.0, punctuation: 1.0, and credibility model output: 0.71. The credibility-adjusted sentiment was computed as follows:

$$\text{Credibility Score} = 0.25 \cdot 1.0 + 0.25 \cdot 1.0 + 0.5 \cdot 0.71 = \mathbf{0.855}$$

$$\text{Final Adjusted Sentiment Score} = -0.738 \cdot 0.855 = \mathbf{-0.631}$$

The LSTM model predicted the following raw prices for the next 10 days:

$$[98.35,\ 98.54,\ 98.73,\ 98.90,\ 99.08,\ 99.27,\ 99.44,\ 99.61,\ 99.77,\ 99.92]$$

We then applied exponential sentiment adjustment starting on May 21 using:

$$\text{adjustment factor}_i = 1 + \frac{\text{adjusted sentiment score}}{10} \cdot (0.5)^i$$

This resulted in the following sentiment-adjusted prices:

$$[92.14,\ 95.05,\ 96.55,\ 97.31,\ 97.69,\ 97.88,\ 97.98,\ 98.03,\ 98.05,\ 98.06]$$

Although sentiment adjustment modifies the magnitude of the predicted prices, it does not arbitrarily reverse the price direction. This is ensured by our built-in trend correction mechanism, as a result, adjusted prices follow the same directional trend (upward or downward) as the LSTM's raw predictions, preserving the model's learned behavior while still incorporating the sentiment signal.

Comparison with Actual Market Prices:

Date	Actual Price	Raw Predicted Price	Adjusted Price
May 21, 2025	\$93.01	\$98.35	\$92.14
May 22, 2025	\$93.01	\$98.54	\$95.05
May 23, 2025	\$94.29	\$98.73	\$96.55

This test illustrates how the model's output reacts to credible negative sentiment. Even though raw predictions pointed to slight increases, the adjusted outputs realistically reflect the amplified momentum, aligning more closely with actual observed prices. This example demonstrates the importance of combining time-series learning with NLP-driven sentiment and credibility scoring. While the LSTM reflects past patterns, the sentiment module introduces event-driven correction, improving accuracy after impactful news events.

5.2 Conclusion

Our LSTM model is able to capture historical patterns and the sentiment and credibility modules allowed it to react dynamically to real-world news events and adjust the predicted prices accordingly, while also preserving the captured trends. For instance, as discussed previously, on AAPL and S&P500, we obtained relatively low MAPE, overall showing a good generalization. However, directional accuracy that ranged for our examples between 53-55% highlights that even if trends are learned, raw predictions sometimes missed short-term sentiment shifts.

To address this, we introduced a sentiment and a credibility analyzer, so we aimed to fine-tune 2 lightweight distilled models. The sentiment analysis model was trained on Financial Phrase Bank and a custom dataset created by us using a Python script and the credibility analysis model was trained on LIAR dataset and another custom dataset created by us using another script. The results show that these significantly smaller models are capable of performing text interpretation tasks (in this case, sentiment and credibility analysis) and output scores closely related to reality, when trained on qualitative data. By integrating credible sentiment into price forecasts, our model better reflects market reactions to impactful news.

While raw predictions offer stable trends, sentiment adjustments add realism by capturing investor behavior.

Nonetheless, our approach has also some clear limitations and areas for improvement in future work, which will be discussed next.

5.3 Current Limitations

Our approach comes unavoidable with multiple limitations and one of them is the prediction time (can go up to even 4 seconds), since we retrain LSTM before any prediction (we couldn't train a single LSTM model for all the stocks because the predicted prices for a ticker would be affected by the data trained on other tickers). We could train a separate LSTM for each stock symbol from US market for instance, but that would need a lot of space for storing each of them and time to train them. Also, by integrating our 3 models together, we get a summed up prediction time (time sentiment analyzer + time credibility + time price prediction + adjustment), that leads to slow predictions.

Another current limitation comes from the unpredictable daily volatility in stock market, which can't be represented by any patterns. Data availability represents also a problem for training our models because our NLP models would need very large amounts of words / phrase labeled to handle the classification even better.

The last very important limitation is that even if some articles are fake, some

people could still rely on that for making a certain decision, which could lead drop / raise prices in an artificial manner, and our models can't predict this kind of mass behavior.

Chapter 6

Future Work

In the future, we plan to make our NLP models a lot better by training them with more text data, this means combining more datasets for sentiment and credibility analysis and also building our own dataset with more labeled examples. We'd also like the models to look at the opinions of bigger groups of people, maybe by pulling data from Reddit threads or Twitter comments.

The learning experience could be improved by adding more AI features, such as models capable of turning the text-based learning materials into audio or training a model to create financial quizzes, to avoid the oversaturation or possible constrained future limitations of the currently used GeminiAPI.

Bibliography

[1] Jialu L. Annamaria Lusardi Streeter. "Financial Literacy and Financial Well-Being: Evidence from the US". In: *Journal of Public Policy* (2023).

[2] Dogu Araci. "FinBERT: Financial Sentiment Analysis with Pre-trained Language Models". In: *arXiv preprint arXiv:1908.10063* (2019).

[3] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. "SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining". In: *LREC* (2010).

[4] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL* (2019).

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning". In: *MIT Press* (2016). `https://www.deeplearningbook.org`.

[6] Pengcheng He et al. "DeBERTa: Decoding-enhanced BERT with Disentangled Attention". In: *ICLR* (2021).

[7] C.J. Hutto and Eric Gilbert. "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text". In: *Proceedings of ICWSM*. 2014.

[8] Xiaoqi Jiao, Yichun Yin, et al. "TinyBERT: Distilling BERT for Natural Language Understanding". In: *Findings of EMNLP* (2020).

[9] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[10] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "The M4 competition: Results, findings, conclusion and way forward". In: *International Journal of Forecasting* 34.4 (2018), pp. 802–808.

[11] P. Malo et al. "Good Debt or Bad Debt: Detecting Semantic Orientations in Economic Texts". In: *Journal of the Association for Information Science and Technology* 65.4 (2014), pp. 782–796.

[12] Christopher Olah. *Understanding LSTM Networks*. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2025-06-06. 2015.

[13] Karan Pardeshi, Sukhpal Singh Gill, and Ahmed M. Abdelmoniem. "Stock Market Price Prediction: A Hybrid LSTM and Sequential Self-Attention based Approach". In: *Unknown or Add Journal Name Here* (2024). URL: https://arxiv.org/abs/2308.04419.

[14] Harshal Patel et al. *Comparative Study of Predicting Stock Index Using Deep Learning Models*. 2023. URL: https://arxiv.org/abs/2306.13931.

[15] David M W Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: *Journal of Machine Learning Technologies* 2.1 (2011), pp. 37–63.

[16] A. A. P. Pratama. "Investment Opportunity of Financial Technology". In: *The International Journal of Business Management and Technology* 3.5 (2019), pp. 1–10. ISSN: 2581-3889. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3476618.

[17] Alec Radford, Jeffrey Wu, et al. "Language Models are Unsupervised Multitask Learners". In: *OpenAI Blog* (2019).

[18] Colin Raffel, Noam Shazeer, Adam Roberts, et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *JMLR* (2020).

[19] Md. Sakib Rahman and Md. Golam Abedin. "Stock Prediction of Google based on ARIMA, XGBoost and LSTM". In: *Available at ResearchGate* (2023). URL: https://www.researchgate.net/publication/370574317_Stock_Prediction_of_Google_based_on_ARIMA_XGBoost_and_LSTM.

[20] Sara Rosenthal, Noura Farra, and Preslav Nakov. "SemEval-2017 Task 4: Sentiment Analysis in Twitter". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. 2017, pp. 502–518.

[21] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[22] Robert P Schumaker and Hsinchun Chen. "A quantitative stock prediction system based on financial news". In: *Information Processing & Management* 45.5 (2009), pp. 571–583.

[23] Josh Starmer. *StatQuest: LSTM Cells Clearly Explained*. https://www.youtube.com/watch?v=8HyCNIVRbSU. Accessed: 2025-06-04. 2017.

[24] TechEmpower. *Web Framework Benchmarks*. https://www.techempower.com/benchmarks/. Accessed: 2025-05-30. 2024.

[25] Ashish Vaswani et al. "Attention is All You Need". In: *Advances in Neural Information Processing Systems* 30 (2017). URL: https://arxiv.org/abs/1706.03762.

[26] Alex Wang, Amanpreet Singh, et al. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. In: *ICLR* (2018).