University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Damian Dąbrowski
Student no. 439954

Heorhii Lopatin
Student no. 456366

Ivan Gechu
Student no. 439665

Krzysztof Szostek
Student no. 440011

Large language models for forecasting market behaviour

Bachelor's thesis
in COMPUTER SCIENCE

Supervisor:
dr Piotr Hofman
Instytut Informatyki

Warsaw, May 2024

Abstract

This thesis concerns research into the use of machine learning and large language models in market analysis, focusing on market predictions.

Keywords

machine learning, large language models, time series forecasting, market prices

Thesis domain (Socrates-Erasmus subject area codes)

11.4 Sztuczna inteligencja

Subject classification

D. Software

Tytuł pracy w języku polskim

Duże modele językowe w przewidywaniu giełdy

Contents

Chapter 1

Introduction

1.1. Overview

In the world of stock markets a major problem is the apparent incalculability of the complex network of factors e.g. how stock prices of one company affect those of another. As the environment of stock markets becomes more and more complex, the ability to analise and confidently predict its future becomes of crucial importance for traders, investors and researchers.

With the recent advent of generative AI and the demonstrable power of Large Language Models a question arises of how these can be used to accurately analise and predict time series market prices in different environments. Therefore, this thesis presents our work on the subject.

1.2. Contributions

1.3. Outline

First, we look at what work has already been done in the field of LLM time series prediction, in particular what techniques of fine-tuning and input data transformation were used. Then we look at how different, smaller machine learning models deal with time series prediction.

We describe the datasets we used for testing small models and LLMs.

Subsequently, we discuss our own methodology; different aplied methods and techniques of input reprogramming, use of prompts and context, and LLM fine-tuning. Next, we present the results we have achieved on the chosen datasets (and compare them to some other known solutions).

Finally, we speculate on the significance of our work, its potential applications in forecasting price time-series.

Chapter 2

Preliminary definitions & guidelines

2.1. Macros

2.2. Notation

- By goal or problem or task we mean the overall task of the thesis, which is develop a machine learning model suitable for predicting prices on the financial market, based on a history of data.

2.3. Datasets

Here we describe the various datasets we used for the training and testing of our models. Each description includes the following:

- **Source:** the name and source from where we took the dataset. Including a link or a way to access and download the dataset.
- **Collection method:** a description of how the data in the dataset was gathered and over what time span.
- **Motivation:** a description of what the data in the dataset represent, what their purpose is, how they were gathered and why it is valuable to our research.
- **Size:** a description of how many datapoints the dataset contains, how many features each datapoints has and the overall size of the file.
- **Features:** a description of the features each datapoint of the dataset has and what the features represent.
- **Characteristic:** a description of the features and drawbacks of the overall dataset e.g. how dependent are the features between themselves.

2.3.1. Apple

- **Source:** Apple Stock Prices
  https://www.kaggle.com/datasets/suyashlakhani/apple-stock-prices-20152020
- **Collection method:** Collected from the stock market (probably).

- **Motivation:** Even more features then the Google Stock dataset, more interesting.
- **Size:** The dataset consists of 1258 datapoints.
- **Features:** Every datapoint has the following features
  - **close** - Closing price
  - **high** - Highest price of the day
  - **low** - Lowest Price of the day
  - **open** - Opening price of the day
  - **volume** - Volume of stock traded
  - **adjClose** - Closing price of the day, modified to account for dividends, stock splits, etc., to better reflect stock value.
  - **adjHigh** - Highest price of the day, modified to account for dividends, stock splits, etc., to better reflect stock value.
  - **adjOpen** - Opening price of the day, modified to account for dividends, stock splits, etc., to better reflect stock value.
  - **adjVolume** - Trading volume of the day, modified to account for dividends, stock splits, etc., to better reflect stock value.
  - **divCash** - Cash dividend - the amount of money payed per share to the stockholder.
  - **splitFactor** - Stock split factor - the ratio used to adjust number of shares and their prices during a stock split.

2.3.2. BTCUSD

- **Source:**
- **Motivation:**
- **Size:**
- **Features:**
- **Collection method:**
- **Quality:**
- **Plot:**

2.3.3. EURUSD

- **Source:**
- **Motivation:**
- **Size:**
- **Features:**
- **Collection method:**
- **Quality:**
- **Plot:**

2.3.4. GBPCAD

- **Source:**
- **Motivation:**
- **Size:**
- **Features:**
- **Collection method:**
- **Quality:**
- **Plot:**

2.3.5. GBPTRY

- **Source:**
- **Motivation:**
- **Size:**
- **Features:**
- **Collection method:**
- **Quality:**
- **Plot:**

2.3.6. Electricity

- **Source:**
- **Motivation:**
- **Size:**
- **Features:**
- **Collection method:**
- **Quality:**
- **Plot:**

2.3.7. US500

- **Source:**
- **Motivation:**
- **Size:**
- **Features:**

- **Collection method:**
- **Quality:**
- **Plot:**

2.3.8. Gold

- **Source:** Gold rates
https://www.kaggle.com/datasets/hemil26/gold-rates-1985-jan-2022
- **Collection method:** This data was collected from https://www.gold.org/goldhub and then cleaned. Has daily data and annual summaries.
- **Motivation:** The dataset contains lots of datapoints and lots of features.
- **Size:** Annual dataset has 43 datapoints. Daily dataset has a bit over 10 thousand datapoints.
- **Features:** Each datapoint has the following features: date and rates in six currencies: USD (American), INR (Indian), AED (Arabian), EUR (European), GBP (South Georgian) and CNY (Chinese).

2.3.9. Sine

- **Source:** sine.csv - we generated this dataset ourselves
- **Motivation:** This is a non-trivial, but predictable, dataset for testing whether LLM responds to the data.
- **Size:** Dataset consists of 1000 datapoints, equally spaced from 0 to 100 - distance between two consecutive is 0.1.
- **Features:** Each datapoint consists of date, x and target, where target = sin(x).
- **Collection method:** Dataset was generated by a script.
- **Characteristic:**
- **Plot:** A simple sine wave.

2.3.10. Time Series Practice Dataset

- **Source:** https://www.kaggle.com/datasets/samuelcortinhas/time-series-practice-dataset/data?select=tra
- **Motivation:** As the Sine dataset it is a fictional dataset to more easily tune LLM training, but more complicated than Sine and significantly larger.
- **Size:** This dataset contains 230 thousand simulated time series datapoints covering 10 years (2010-2019). The are no null values.
- **Features:** The features include date, store id, product id and number sold. The train.csv covers the years 2010-2018 and the test.csv covers 2019 only. The are 7 unique stores and 10 unique products.

- **Collection method:** This time series data was created using multiple features including various long term trends, year-long seasonality patterns, weekday/weekend effects and noise. Moreover, the products and the stores are supposed to be weakly correlated.

2.4. What metrics we used

Every metric should be described by

- **Definition:** How the metric is defined, what it measures.
- **Calculation:** Description of how the metric is calculated.
- **Interpretation:** A consideration of what the result of the metric may be interpreted to mean about the model it measures.

2.4.1. Mean Squared Error

The Mean Squared Error (MSE) is a metric used for evaluating the accuracy of a machine learning model, especially in regression tasks. It quantifies how close a model's predictions are to the actual values.

Definition

MSE calculates the average of the squares of the errors. The error is the difference between the values ($\hat{y}_i$) predicted by the model and the values (y_i) it should have predicted.

Calculation Steps

The metric is computed as follows.

1. For each prediction the error is computed by subtracting the predicted value from the actual value.
2. Each error is then squared in order to ensure that positive and negative errors do not cancel each other out and in order to emphasize larger errors.
3. The average of these squared errors is then computed to obtain the MSE.

Mathematical Formulation

The mathematical formula for MSE is given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

where:

- n is the number of data points,
- y_i is the actual value for the ith datapoint,
- $\hat{y}_i$ is the predicted value for the ith datapoint,

Interpretation

- MSE is a non-negative number where a value of 0 indicates perfect predictions.
- Larger MSE values indicate worse model performance.
- MSE emphasizes larger errors due to the squaring of each term, which can be both advantageous and disadvantageous depending on the application.

2.4.2. R^2 Score

The R^2 score, or the coefficient of determination, is a statistical measure used in regression analysis to assess how well fitted a model is. It indicates the proportion of the variance in the dependent variable that is predictable from the independent variable(s).

Definition

The R^2 score is defined as the ratio of the variance explained by the model to the total variance. It is a measure of how well the observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model.

Calculation

The R^2 score is calculated using the following formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{2.2}$$

where:

- y_i is the actual value for the ith data point,
- $\hat{y}_i$ is the predicted value for the ith data point,
- $\overline{y}$ is the mean of the actual values,
- n is the number of data points.

Interpretation

- An R^2 score of 1 indicates that the regression model perfectly fits the data.
- An R^2 score of 0 means that the model does no better, than would a prediction using the mean.
- Negative R^2 values can occur when the chosen model fits worse than a horizontal line representing the mean of the response.

Limitations

The R^2 metric has some limitations. It does not necessarily imply causation, nor does a high R^2 score mean that the model is the best choice for prediction. It's also important to note that using more features of the data by the model can artificially inflate the R^2 value, even if those features are not statistically significant.

2.5. How we describe models

Below, in the chapter "Other models" we describe several models we tried to use for the task of predicting prices. The descriptions include the following:

- **Description:** a short description of how the model works and how it is trained.
- **Motivation:** what the model is usually used for and why we chose to try it out.
- **Features and limitations:** some advantages and benefits of the model, as well as its disadvantages and drawbacks.
- **Parameters:** the description of the parameters of the model and how they affect its training.
- **Metrics:** how we measured the results of the training and testing of the model.
- **Data used:** what combinations of parameters of the model we tested and on what datasets we trained and tested the model and how these datasets were divided into training and testing subdatasets.
- **Preprocessing:** how the datasets used were preprocessed for training and testing of the model.
- **Analysis:** an analysis of our results of our training and testing of the model compared with the results obtained in literature.
- **Picture:** a picture or a plot demonstrating the results obtained from testing the model.

2.6. Literature review

Literature review contains:

- A list of approaches to the problem.
- For each approach, its basic description and its significance to our goal.
- Its features and drawbacks compared to our goal.
- Its differences, when compared to our goal.
- Whether our own results validated the results of the article.

Chapter 3

Other models

Here we present our results from trying to use the following models to extrapolate a time series. Classification models output in binary categories: increase or decrease in value, and are therefore less precise.

In the below descriptions, **overfitting** refers to a phenomenon when model learns from the training data too closely or exactly, thereby making it less generalisable to new data. See figure Figure 3.1. Figure and description are both due to Wikipedia [1].

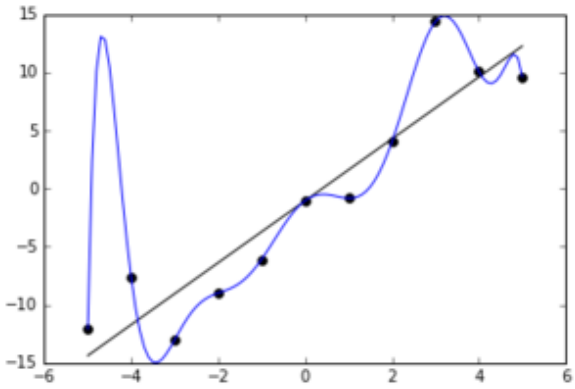

Figure 3.1: Noisy (roughly linear) data is fitted to a linear function and a polynomial function. Although the polynomial function is a perfect fit, the linear function can be expected to generalize better: if the two functions were used to extrapolate beyond the fitted data, the linear function should make better predictions.

3.1. Random forest

A random forest [2] is a machine learning model for classification and regression tasks introduced by Leo Breiman in 2001. A random forest is an ensemble of individual tree predictors, each of which depends on a random vector, chosen independently and with the same distribution for every tree. The results from individual trees are then aggregated into the overall result of the model - for classification tasks it is the mode of individual classifications and for prediction tasks it is the mean average of individual predictions.

The error of forest prediction converges as such as the quantity of trees in the forest increases. The error of forest prediction depends negatively on the accuracy of individual trees and positively on the correlation between them.

An individual tree is constructed in the following way: a **sample** - random subset of the dataset - is selected. This and only this subset is used in growing the tree. Then, for each node, a random subset of features is chosen. A best split is chosen using the `criterion` function, based on the chosen features of the sample. Then each tree is grown to maximum depth (until all leaves have one datapoint, for classifier trees).

Below is the list of parameters that influence the growth of trees:

- The `num_lags` parameter is the number of previous datapoints taken into account for individual predictions - if $k =$ `num_lags`, then datapoints $[n-k, n)$ are used to predict price at n. The larger the `num_lags` the more of past data the model takes into account.
- The `n_estimators` parameter (number of estimators) describes the number of trees grown in the forest. Increasing the number of trees increases the accuracy of the model, but it also increases the computational cost.
- The `max_features` parameter (number of features) specifies the the number of features of the data considered for a split at each node while growing an individual tree. A higher value of `max_features` may capture more information about the data at the risk of growing more correlated trees, thereby decreasing its randomness and ability to adapt to new data, causing overfitting.
- The `criterion` parameter describes the function used by the model to calculate a quality of a split at a given node

3.1.1. Results

TBA

3.2. Logistic regression

Logistic regression [3] is a statistical method used for binary classification problems, where the outcome variable has one of two possible values. It models the probability that a given input point belongs to a certain class.

In the below description, n denotes the size of the training dataset. For our case of predicting time series, an input vector x_i is a subseries of datapoints of length `num_lags`, for which the output y_i is a prediction whether the datapoint following x_i will have a higher or lower price than the previous datapoint. Therefore, in the below description, $d =$ `num_lags`$*k$, where k is the number of features of an individual datapoint.

With that in mind, the general formulation of the logistic regression model is as follows:

Given a set of n observations $\{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathbb{R}^d$ represents the feature vector and $y_i \in \{0, 1\}$ represents the binary outcome, the logistic regression model predicts the probability of the outcome being 1 (positive class) as:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d)}}$$

where β_0 is the shift term, and $\beta_1, \beta_2, \ldots, \beta_d$ are the coefficients corresponding to the d features.

The model is trained by maximizing the likelihood function, which measures the probability of observing the given data as a function of β parameters. The likelihood function for logistic regression is given by:

$$L(\beta) = \prod_{i=1}^{n} P(y_i = 1 \mid x_i)^{y_i} (1 - P(y_i = 1 \mid x_i))^{1-y_i}$$

To simplify computation, the log-likelihood function is often used:

$$\ell(\beta) = \log(L(\beta)) = \sum_{i=1}^{n} [y_i \log(P(y_i = 1 \mid x_i)) + (1 - y_i) \log(1 - P(y_i = 1 \mid x_i))]$$

The optimal parameters β are estimated by maximizing the log-likelihood function using numerical optimization techniques such as gradient descent.

3.2.1. Results

TBA

3.3. Support vector machine

Support Vector Machines (SVMs) are supervised learning models used for classification and regression tasks. They are particularly well-suited for binary classification problems. The main idea behind SVMs is to find the optimal hyperplane that maximally separates the data points of different classes in the feature space.

A hyperplane is defined by the equation

$$\mathbf{w} \cdot \mathbf{x} + b = 0,$$

where $\mathbf{w}$ is the weight vector and b is the bias term. The optimal hyperplane is the one that maximizes the margin, which is the distance between the hyperplane and the nearest datapoints from either class, known as support vectors.

To find this optimal hyperplane, SVM solves the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2$$

subject to the constraint:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall i$$

where $y_i \in \{-1, 1\}$ is the class label of the i-th data point $\mathbf{x}_i$.

For non-linearly separable data, SVM can employ kernel functions to map the input features into a higher-dimensional space where a linear separation is possible.

3.3.1. Parameters

- **Kernel:**[1] The model uses a kernel function to transform the space of data points which are not separable by a hyperplane, into one where they are separable.

  The following description is from [4] from page 4. X and Z are, respectively, the original input space and the transformed, high-dimensional space.

  Roughly speaking, a kernel $K(x, y)$ is a real-valued function $K : X \times X \to \mathbb{R}$ for which there exists a function $\Phi : X \to Z$, where Z is a real vector space, with the property $K(x, y) = \Phi(x)^T \Phi(y)$. The kernel $K(x, y)$ acts as a dot product in the space Z.

[1] Note that the word 'kernel' here is **not** meant in the linear algebra sense of part of the domain that is transformed to zero.

- **C value:** The C value specifies how accurate the model should be, that is how much it should avoid misclassifications, versus how wide the margins should be. A lower value of C corresponds to wider margins, but potentially more misclassifications.
- **Gamma:** The γ parameter is a component of non-linear kernel functions, (e.g. the Radial Basis Function (RBF) kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$. It determines how influential a single datapoint has in shaping the optimal boundary A high γ value causes the influence of each training point to be very localized, resulting in a more complex and tailored model that can capture intricate patterns in the data, but with a higher risk of overfitting. A low γ value means that the influence of each training point extends further, producing a smoother and simpler decision boundary that may generalize better to unseen data but be less accurate.

3.3.2. Results

TBA

3.4. Multilayer Perceptron

The Multilayer Perceptron (MLP) [5] is a feedforward neural network, that is made up of multiple layers of nodes in a directed graph, where each node from one layer is connected to all the nodes from the previous one. MLPs are widely used in pattern recognition, classification, and regression problems due to their ability as networks to model complex nonlinear relationships in the input data. An MLP consists of an input layer of neurons, one or more hidden layers, and an output layer. Each node, except for those in the input layer, is a neuron that uses a nonlinear activation function to combine inputs from the previous layer and an additional bias term.

3.4.1. Structure of an MLP

An MLP is made up of the following components:

- **Input Layer:** The first layer of the network, which receives the input data to be processed. Each neuron in this layer represents a feature of the input data.
- **Hidden Layers:** One or more layers that perform computations on the inputs received and pass their output to the next layer. The neurons in these layers apply activation functions to their inputs to introduce nonlinearity.
- **Output Layer:** The final layer that produces the output of the network.

3.4.2. Forward Propagation

The process of computing the output of an MLP is called forward propagation. In this process, the input data is passed through each layer of the network, transforming the data as it moves through. The output of each neuron is computed as follows:

$$a_j^{(l)} = \phi\left(\sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}\right) \tag{3.1}$$

where

- $a_j^{(l)}$ is the activation of the j-th neuron in the l-th layer,
- ϕ denotes the activation function,
- $w_{ji}^{(l)}$ represents the weight from the i-th neuron in the $(l-1)$-th layer to the j-th neuron in the l-th layer,
- $b_j^{(l)}$ is the bias term for the j-th neuron in the l-th layer,
- $a_i^{(l-1)}$ is the activation of the i-th neuron in the $(l-1)$-th layer.

3.4.3. Backpropagation and Training

To train an MLP, the backpropagation algorithm is used. This algorithm adjusts the weights and biases of the network to minimize the difference between the actual output and the expected output. The process involves computing the gradient of a loss function with respect to each weight and bias in the network, and then using these gradients to update the weights and biases in the direction that minimizes the loss. The loss function measures the error between the predicted output and the actual output. The update rule for the weights is given by:

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} \tag{3.2}$$

where

- η is the learning rate. If it is too small, the model will train very slowly and may get stuck in local minima. If it is too large, it might not converge.
- $\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}}$ is the partial derivative of the loss function $\mathcal{L}$ with respect to the weight $w_{ji}^{(l)}$.

Similar updates are made for the biases.

Through iterative training involving forward propagation, loss calculation, and backpropagation, the MLP learns to approximate the function that maps data inputs to desired predictions.

3.4.4. Results

TBA

3.5. Convolutional neural network

Convolutional Neural Networks (CNNs) [6] are a class of deep neural networks, highly effective for analyzing visual imagery. They employ a mathematical operation called convolution, which allows them to efficiently process data in a grid-like topology, such as images.

3.5.1. Architecture of Convolutional Neural Networks

A typical CNN architecture comprises several layers that transform the input image to produce an output that represents the presence of specific features or class labels. The most common layers found in a CNN are:

Convolutional Layer

The convolutional layer is the fundamental building block of a CNN. It applies a set of learnable filters to the input. Each filter activates specific features at certain spatial positions in the input. Mathematically, the convolution operation is defined as follows:

$$f(x, y) = (g * h)(x, y) = \sum_m \sum_n g(m, n) \cdot h(x - m, y - n)$$

where $f(x, y)$ is the output, g is the filter, h is the input image, and $*$ denotes the convolution operation. x and y range over output image dimensions; m and n range over the filter dimensions.

Activation Function

Following convolution, an activation function is applied to introduce non-linearity into the model. The Rectified Linear Unit (ReLU) is commonly used:

$$f(x) = \max(0, x)$$

Pooling Layer

The pooling layer reduces the spatial dimensions (width and height) of the input volume for the next convolutional layer.

Fully Connected Layer

Towards the end of the network, fully connected layers are used, where each input node is connected to each output by a learnable weight. This layer classifies the image into various classes based on the learned high-level features.

Output Layer

The final layer of a CNN outputs a probability distribution over the classes, indicating the likelihood of the input image belonging to each class.

A simple diagram of the layers can be seen on figure Figure 3.2 (due to [**?**]).

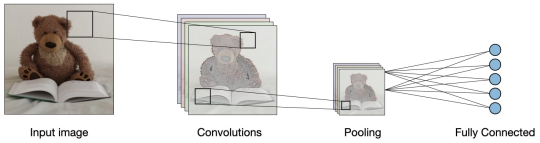

Figure 3.2: A simple diagram illustrating the different layers of the network working on an example input image.

3.5.2. Results

TBA

3.6. Residual Neural Network

A Residual Neural Network (ResNet) is a type of deep learning model specifically designed to mitigate the vanishing gradient problem, where through backpropagation only the last few layers of the network get trained. ResNets introduce skip connections, also known as residual connections, which allow the input of a layer to be directly added to the output of a subsequent layer. This is mathematically expressed as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

where $\mathbf{y}$ is the output of the layer, $\mathcal{F}$ represents the residual mapping to be learned by the layer, $\mathbf{x}$ is the input, and $\{W_i\}$ denote the weights of the layers.

The primary advantage of this architecture is its ability to facilitate the training of deeper networks by preserving the gradient flow through the network during backpropagation. This is achieved by allowing the backpropagation to bypass one or more layers, reducing the risk of gradient vanishing. As a result, ResNets can be trained to much depths than CNNs, and can have more layers.

3.6.1. Results

TBA

Chapter 4

Large Language Model

In this chapter we present the basic theory behind Large Language Models. We then introduce the LLaMA family of models, which we've been using. Subsequently, we describe the embedding technique we've used and we present our results.

4.1. Vocabulary

The following vocabulary is used:

- **Token** is a basic unit of text data a language model processes - usually words, subwords, punctuation marks etc.
- **Tokenization** is a process of breaking down input data into tokens.
- **Language model** is a *probabilistic model* of a natural language. Probabilistic - meaning that given some input text data, it's job is to predict the future token. [10]
- **Supervised learning** is a type of a method of training machine learning models where every input is supplied with the output the model is expected to produce. The model then matches its own output with the expected output to correct its own behaviour.
- **Pretraining** is a process of training the language model on a corpus of data
- **Fine-tuning** is a process of adapting a pretrained language model to a specific task (e.g. mathematics, poetry) by training it on a smaller, task-specific dataset.
- **Token embedding** is a mapping of tokens to high-dimensional vectors of real numbers. This mapping is expected to have the property that tokens similar in meaning are close in the output space. See [9].
- **Context length** is the maximal size of input tokens a large language model can process at any one time.
- **Cross-modality data** is data that combines multiple modalities, i.e. text, image, audio, video, etc. In particuliar, combination of text description and time series is cross-modality data.

4.2. Overview

A Large Language Model (LLM) [11] is a language model that is pretrained on a large collection of data (usually millions of tokens). These models utilize deep learning techniques, particularly neural networks, which consist of interconnected neuron layers that process information sequentially, one by one. The predominant architecture underpinning most contemporary LLMs is the Transformer (see below), notable for its self-attention mechanism that enables the model to assess the importance of different tokens in a sentence irrespective of the order the tokens are in.

The training of an LLM involves pretraining it with a large, diverse corpus of text, during which it adjusts its internal parameters to minimize the difference between its predictions and the actual data. This process of supervised learning equips the model with a probabilistic understanding of the language (its patterns, semantics, syntax, knowledge of the world inherent in a language), enabling it to predict a continuation of a given piece of input text.

Once trained, LLMs can perform a variety of language-based tasks such as translation, summarization, question answering, and text generation. It can then be fine-tuned to perform better on a specific task, e.g. write poetry, give cooking advice, write programming code, etc. We will see how such a model deals with analysing and predicting time series data.

4.3. The Transformer

The Transformer is a type of neural network architecture introduced in the seminal 2017 paper *"Attention is All You Need"* by Vaswani et al. [12]. It has since become foundational for many natural language processing (NLP) models due to its efficiency and effectiveness in handling data sequences, such as text.

4.3.1. Components

The Transformer (fig Figure 4.1) consists of the following components:

1. **Input Embedding**: Converts input tokens into vectors
2. **Positional Encoding**: Adds positional information to the embeddings to retain the order of the sequence. This encoding is another high-dimensional vector.
3. **Encoder** (Figure 4.1 the left):
   - Consists of $N = 6$[1] layers.
   - Each layer has two sub-layers:
     - **Multi-Head Attention**: Applies attention mechanism over the input. For each token calculates the weight or importance of over tokens in the surrounding context (*Attention*). Does so independently with h[2] 'heads', each calculating different semantic relantionships (*Multi-Head*). The results are concatenated and linearly transformed into size of one output (as if from one head).
     - **Feed Forward**: Applies a fully connected feed-forward neural network.
   - **Add & Norm**: Residual connections followed by layer normalization after each sub-layer.

[1] [12], section 3.1)
[2] In [12], $h = 6$

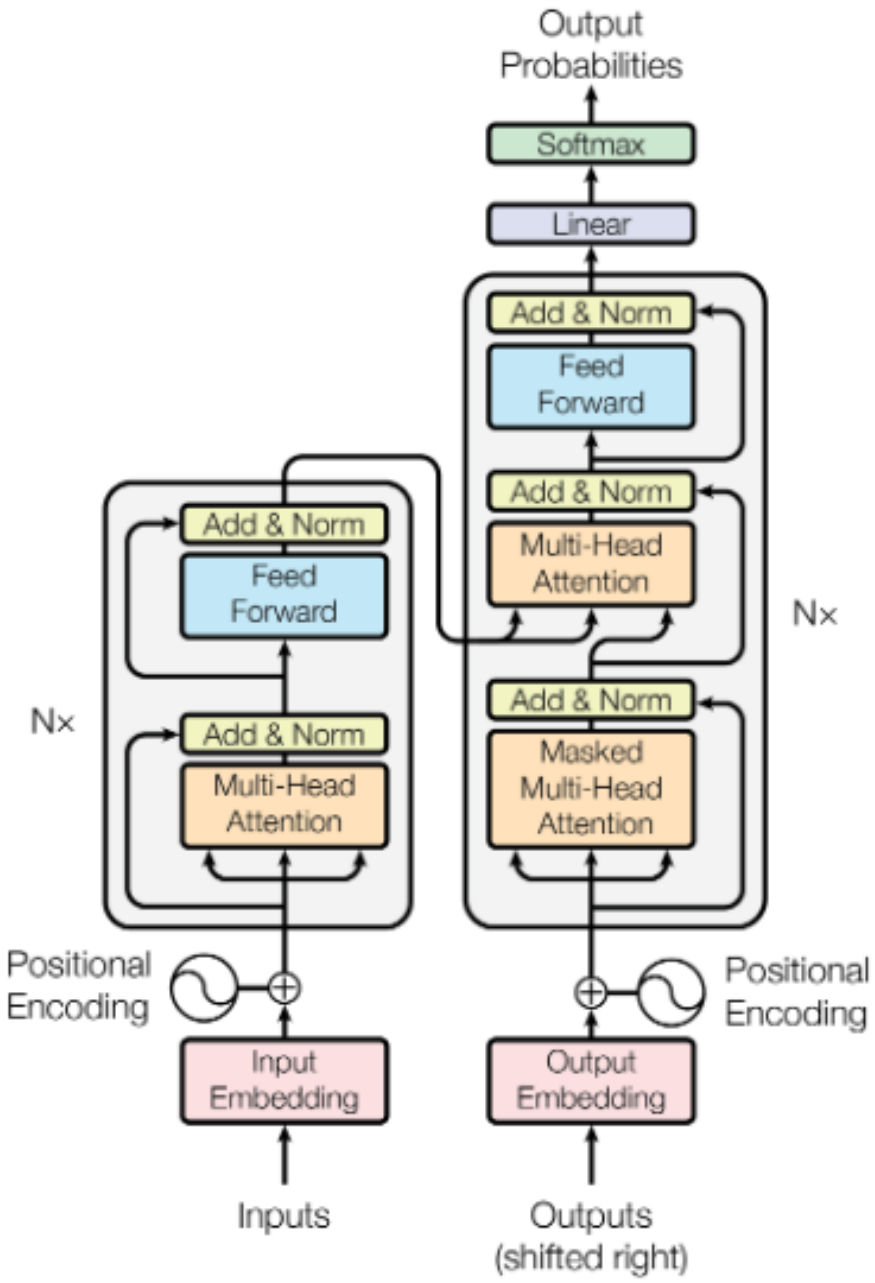

Figure 4.1: The Transformer model architecture

4. **Decoder** (Figure 4.1, on the right):

   - Also consists of $N = 6$[3] layers.
   - Each layer has three sub-layers:
     - **Masked Multi-Head Attention**: As in Encoder, but masked, i.e. attention results for future tokens are disarded - .

[3] [12], section 3.1)

- **Multi-Head Attention**: Applies the multi-head attention mechanism to encoder output.
- **Feed Forward**: Applies a fully connected feed-forward neural network.

- **Add & Norm**: Residual connections followed by layer normalization after each sub-layer.

5. **Output Embedding**: Converts decoder output tokens into semantic vector space.

6. **Linear & Softmax Layer**: Maps the decoder's output to the probability distribution over the target vocabulary. The most probable token is the output. [4]

4.4. LLaMA model

4.4.1. Introduction

LLaMA or *Large Language Model Meta AI* [13] is a collection of large language models developed by Meta AI.

4.4.2. Features

- **Model Variants:** LLaMA is available in various sizes, offering flexibility for deployment in different environments. These variants range from models of 7 billion parameters in size up to models of 65 billion parameters in size. [5]

- **Training Data:** The model has been trained on 1.4 trillion tokens of data from several sources, including CommonCrawl and Github, Wikipedia (Table 1. in [13]). It therefore has an enormous and domain diverse range of input data.

- **Accessibility:** The code that can be used to run the model has been publicly released under the open-source GPLv3 license [14].

4.4.3. LLaMA-2

LLaMA-2 [15] is an improved version of LLaMA, with similar model sizes. It has the same architecture as LLaMA-1, but was trained on a much larger set of data (2 trillion tokens). It also has doubled context length of 4096 tokens.

4.5. Time-series Embedding

We now present the technique we've used for using a vanilla (not fine-tuned) LLaMA-2 model to predict time series data. The main idea involves using a framework around a frozen LLM (i.e. one that is not changed during the training process) that transforms input time series data into a text representation the LLM can then work on. Its output is then converted into a prediction. The idea is due to an article by Jin et al. (2024) [16].

The following three paragraphs come from the article. We consider the following problem: given a sequence of historical observations $X \in \mathbb{R}^{N \times T}$ consisting of N different 1-dimensional

[4]This may seem contrary to popular experience using e.g. ChatGPT, where given the same input, it may not necessarily output the same result. However, such tools may have random seeds for each interaction session and also may take into account the context of the conversation.

[5]LLaMa-3, which came out in April 2024, has size possibilities of 8 billion and 70 billion parameters

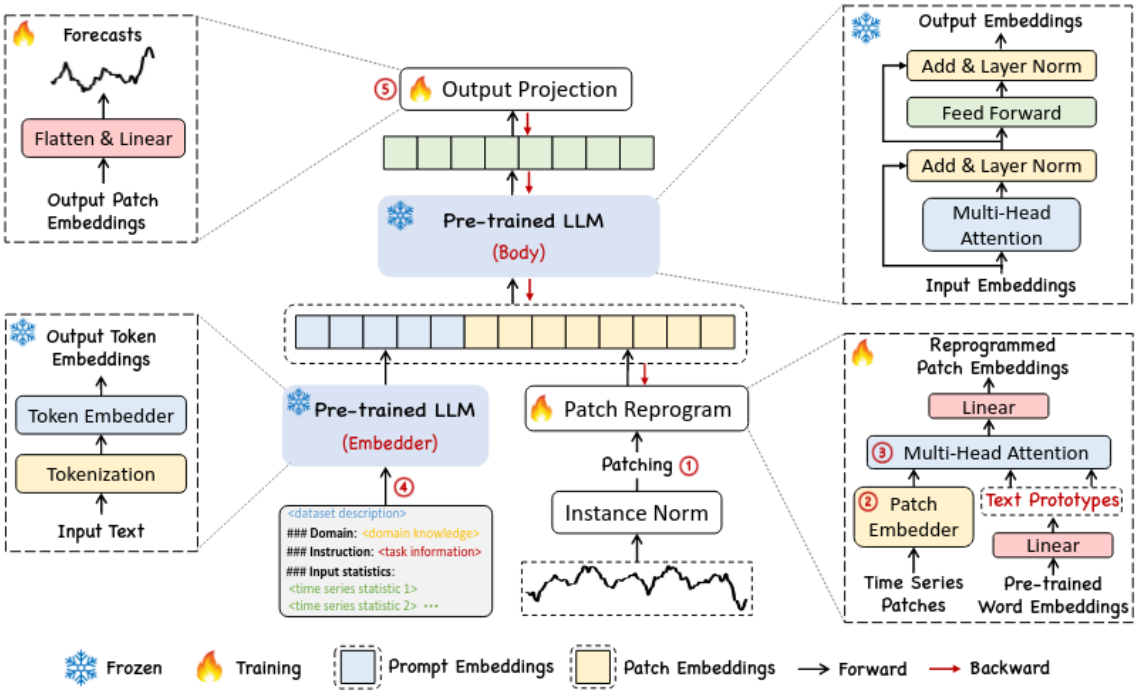

Figure 4.2: The model framework of TIME-LLM. Given an input time series, we first tokenize and embed it via (1) patching along with a (2) customized embedding layer. (3) These patch embeddings are then reprogrammed with condensed text prototypes to align two modalities. To augment the LLM's reasoning ability, (4) additional prompt prefixes are added to the input to direct the transformation of input patches. (5) The output patches from the LLM are projected to generate the forecasts.

variables across T time steps, we aim to reprogram a large language model $f(\cdot)$ to understand the input time series and accurately forecast the readings at H future time steps, denoted by $\hat{Y} \in \mathbb{R}^{N \times H}$, with the overall objective to minimize the mean square errors between the expected outputs Y and predictions, i.e., $\frac{1}{H}\sum_{h=1}^{H} \|\hat{Y}_h - Y_h\|_F^2$.

The method encompasses three main components: (1) input transformation, (2) a pre-trained and frozen LLM, and (3) output projection. Initially, a multifeature time series is partitioned into N unifeature time series, which are subsequently processed independently (Nie et al., 2023) [17]. The i-th series is denoted as $X(i) \in \mathbb{R}^{1 \times T}$, which undergoes normalization, patching, and embedding prior to being reprogrammed with learned text prototypes to align the source and target modalities. Then, we augment the LLM's time series reasoning ability by prompting it together with the transformed series to generate output representations, which are projected to the final forecasts $\hat{Y}^{(i)} \in \mathbb{R}^{1 \times H}$.

We note that only the parameters of the lightweight input transformation and output projection are updated, while the backbone language model is frozen. In contrast to vision-language and other multimodal language models, which usually fine-tune with paired cross-modality data, this use of model is directly optimized and becomes readily available with only a small set of time series and a few training epochs, maintaining high efficiency and imposing fewer resource constraints compared to building large domain-specific models from scratch or fine-tuning them.

4.6. Our methodology

In the current and the next few sections we will share the details of our implementation. The goal is to further elaborate on the idea presented in the previous section and describe the specifics and challenges of our approach.

4.6.1. Data Preprocessing

In the previously defined model architecture, the input consists of a vector of floats that represent the prices over time. Therefore we only use a single feature of the time series data. This means that in the context of the datasets discribed earlier all but the target column are discarded. Afterwards, the data is normalised. A single entry in the updated dataset consists of two features, first one being a vector of length **seq_len** floats representing successive prices with a step of **seq_step** values in the original dataset, along with the target feature which is the next **pred_len** prices using the same step size.

4.6.2. Embedding

Embedding is the part where we take the advantage of the fact that human language can be translated into a sequence of tokens which can serve as instructions to the LLM. In our case, we use the predefined methods of the LLaMa model to convert the instructions into a tensor of floats. This tensor will later be used as a part of the input to the LLM.

4.6.3. Patching

Considering that the weights of the LLM are frozen, the actual training of the model is mostly done by changing the patching. We project the feature vector into a d-dimensional vector space by repeating some of the values and apply convolution and dropout to the result. Afterwards, multi-head attention is applied using linearly modified embeddings from LLaMA. At the very end we apply a linear transformation once again. The resulting tensor is later treated as a sequence of tokens along with the output from the patching layer.

4.6.4. Body and Output Projection

Two sequences of tokens received from the pathing and embedding layers are fed to the LLaMa as a single sequence of tokens. The output is then projected to the forecasted values.

4.7. Model Parameters

In this study, we employed a set of distinctive parameters for our model training:

- **seq_len** – This parameter defines the number of records in one prompt to the model.
- **pred_len** – This parameter specifies the number of records to predict.

- **seq_step** – This parameter determines the step size in records to move the prompt. For instance, if *seq_step* is set to 4, the records would be indexed as 0, 4, 8, and so on. This approach was essential in filtering out noise caused by hourly fluctuations, significantly enhancing the results.

- **lradj** - Describes whic strategy should be used to adjust the learning rate, e.g. **type1** implies $lr_{n+1} = lr_n * 0.5$.

- **n_heads** - number of heads in the multi-head attention mechanism.

- **d_ff** - number of neurons in the feed forward layer.

4.7.1. Impact on Results

The effectiveness of these parameters is highly contingent upon the specific dataset in use. For datasets characterized by high volatility and a high frequency of records, a smaller *seq_step* is preferable. Conversely, for data that remains relatively stable over time, a larger *seq_step* is necessary to prevent the model from merely replicating the last observed value.

Our experimentation with various proportions between *seq_len* and *pred_len* revealed that optimal results were achieved when *pred_len* was approximately one-fourth of *seq_len*. This finding is intuitive, as it ensures the indicators retain their significance. A more detailed discussion on this can be found in the Prompt Engineering section.

Due to constraints in time and resources, we were unable to identify a universally optimal ratio for all datasets. Nevertheless, we believe that such a golden ratio exists and can be discovered with further research. More detailed information on this can be found in the Results and Conclusion section.

4.7.2. Overfitting Concerns

To avoid reducing the number of records available for training, we leveraged the *seq_step* parameter in our data loaders. Rather than using every *seq_step* value, we trained the model with sequences such as 0, 4, 8, 12, etc., followed by 1, 5, 9, 13, and so on (if *seq_step* was set to 4).

This approach, however, led to overfitting, particularly with a high *seq_len* of 200, a *seq_step* of 12, and a *pred_len* of 40. For illustration, consider feeding the model data from the past 20 weeks and predicting the next 4 weeks (one month). Our currency datasets contain 24 records per day over 5 days a week.

In the initial iterations, our model achieved an accuracy of 89

However, this approach failed during validation, as the model's accuracy drastically dropped when applied to completely unseen test data. This highlights the importance of diversifying training sequences to avoid overfitting and improve the model's generalization capabilities. Further exploration and refinement of these parameters are necessary to develop a robust predictive model.

4.8. Prompt Engineering

The foundational concept behind leveraging a Large Language Model (LLM) lies in its extensive knowledge about the world. Our objective was to determine optimal strategies to harness this knowledge, thereby enhancing our model's forecasting accuracy. Essentially, we aimed to

bypass fine-tuning and instead focus on crafting the most effective task descriptions to elicit accurate predictions from the outset.

Initially, we experimented with simply providing sequences of numbers, similar to our approach in other models. However, this method proved inadequate, as the responses were often off-target and lacked focus. Recognizing the need for a more sophisticated approach, we turned to autocorrelation analysis to identify recurring patterns within the data. By applying Fourier transformation, we identified the most significant patterns and selected the steps with the highest autocorrelation as our $\text{top}_k features$.

This method notably improved our accuracy, especially for datasets with clear seasonal patterns, such as electricity consumption over the year. For instance, electricity usage typically increases during the winter months and decreases during the summer, with similar trends repeating annually. However, our primary goal was to predict market movements, particularly in the forex market, which lacks such seasonal patterns. Stock prices, for example, tend to grow over the years rather than oscillate within a fixed range.

To address this challenge, we explored various analytical tools commonly used in trading. Numerous indicators are designed to forecast future prices, based on factors such as moving averages, trading volume, and price trends. We decided to incorporate three widely-used indicators: Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and Bollinger Bands (BBANDS). These indicators, already familiar to the model due to its pre-existing knowledge, significantly enhanced its predictive capabilities.

4.8.1. Indicators Used

- **Relative Strength Index (RSI)**: This momentum oscillator measures the speed and change of price movements. It oscillates between 0 and 100 and is typically used to identify overbought or oversold conditions in a market.

- **Moving Average Convergence Divergence (MACD)**: This trend-following indicator shows the relationship between two moving averages of a security's price. The MACD is calculated by subtracting the 26-period Exponential Moving Average (EMA) from the 12-period EMA.

- **Bollinger Bands (BBANDS)**: These volatility bands are placed above and below a moving average. Volatility is based on the standard deviation, which changes as volatility increases and decreases.

Incorporating these indicators yielded a substantial improvement in our model's performance. The accuracy of our predictions increased by over 2

4.8.2. Possible Improvements

4.8.3. Underlying LLM

The 7B LLaMa-2 model we used is significantly less powerful in comparison to 70B LLaMa-3 or even 70B LLama-2, as it follows from the official benchmarks. It would be interesting to see how the performance of the model would be affected by a more modern underlying LLM, however it would require significantly more computational resources as well as additional code that would allow us to run the program on multiple GPUs.

4.8.4. Larget Training Set

The model was trained on datasets which contained less than 40000 records. It takes us approximately 12 hours to train the model on a dataset of this size. Training the model on a larger dataset would allow to fit more complex patterns in the datasets which could potentially be developing over a span of multiple years.

4.9. Results

Chapter 5

Main results

In this chapter we present the results in the following way: for each dataset described in chapter 3, we present one table. The table presents accuracies each model achieves at predicting at predicting the price `Prediction Timestep` into the future.

Todo: describe parameters pred$_{l}enandseq_{s}tep$

Datasets were tested with the following parameters: AAPL, 40, 1, (ostatni) BTCUSD, 5, 2, (ostatni) EURUSD, 7, 7, (ostatni) GBPCAD, 10, 7, (czwarty) GBPTRY, 5, 2, (pierwszy) Electricity, 6, 11, (czwarty) US500, 10, 1, (dziewiąty)

Todo: describe the process of preparing datasets.

Todo: describe measuring accuracy, the loss function.

Apple Stocks Dataset							
Prediction Timestep	Random forest	Logistic regression	SVM	MLP	CNN	ResNet	LLM
8 days	xy	xy	xy	xy	xy	xy	0.48047337
16 days	xy	xy	xy	xy	xy	xy	0.48742604
24 days	xy	xy	xy	xy	xy	xy	0.49630178
32 days	xy	xy	xy	xy	xy	xy	0.52204142
40 days	xy	xy	xy	xy	xy	xy	0.54985207

BTCUSD Dataset							
Prediction Timestep	Random forest	Logistic regression	SVM	MLP	CNN	ResNet	LLM
2 hours	xy	xy	xy	xy	xy	xy	0.50065963
4 hours	xy	xy	xy	xy	xy	xy	0.5055409
6 hours	xy	xy	xy	xy	xy	xy	0.5
8 hours	xy	xy	xy	xy	xy	xy	0.50092348
10 hours	xy	xy	xy	xy	xy	xy	0.50540897

EURUSD Dataset							
Prediction Timestep	Random forest	Logistic regression	SVM	MLP	CNN	ResNet	LLM
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy

GBPCAD Dataset							
Prediction Timestep	Random forest	Logistic regression	SVM	MLP	CNN	ResNet	LLM
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy

GBPTRY Dataset							
Prediction Timestep	Random forest	Logistic regression	SVM	MLP	CNN	ResNet	LLM
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy

Electricity Dataset							
Prediction Timestep	Random forest	Logistic regression	SVM	MLP	CNN	ResNet	LLM
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy

US500 Dataset							
Prediction Timestep	Random forest	Logistic regression	SVM	MLP	CNN	ResNet	LLM
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy
t	xy	xy	xy	xy	xy	xy	xy

Chapter 6

Conclusion

Bibliography

[1] https://en.wikipedia.org/wiki/Overfitting

[2] Scornet, E., Biau, G., & Vert, J. P. (2015). Consistency of Random Forests. arXiv preprint arXiv:1511.05741.

[3] Meng, L., Cao, J., Zhang, C., Yu, S., & Yang, Q. (2020). Sufficient Dimension Reduction for Logistic Regression. arXiv preprint arXiv:2008.13567.

[4] Steinwart, I., & Christmann, A. (2006). Estimating conditional quantiles with the help of the pinball loss. arXiv preprint math/0612817.

[5] (Add complete citation when available).

[6] Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1511.08458.

[7] https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv preprint arXiv:1512.03385.

[9] Jurafsky, D., & Martin, J. H. (n.d.). Token Embeddings. Retrieved from https://web.stanford.edu/ jurafsky/slp3/6.pdf.

[10] Li, Z., Li, J., & Liu, X. (2023). Efficient Language Models with Dynamic Token Dropping. arXiv preprint arXiv:2303.18223.

[11] Zhang, Z., Li, X., & Yang, W. (2023). An Introduction to Large Language Models. arXiv preprint arXiv:2304.00612.

[12] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is All You Need. arXiv preprint arXiv:1706.03762.

[13] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Jegou, H. (2023). LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971.

[14] Meta AI. (2023). LLaMA GitHub Repository. Retrieved from https://github.com/meta-llama/llama.

[15] Touvron, H., Martin, X., Stone, A., Albert, P., Almahairi, A., Babaei, Y., ... & Jegou, H. (2023). LLaMA 2: Open Foundation and Fine-Tuned Chat Models. arXiv preprint arXiv:2307.09288.

[16] Wang, Y., Xu, J., & Lin, J. (2023). Reprogramming Large Language Models with Synthetic Data. arXiv preprint arXiv:2310.01728.

[17] Nie, Y., Nguyen, N. H., Sinthong, P., & Kalagnanam, J. (2023). A time series is worth 64 words: Long-term forecasting with transformers. In International Conference on Learning Representations.