

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

D.127. Blabalgorithms

D.127.6. Numerical blabalysis

## **Tytuł pracy w języku polskim**

Duże modele językowe w przewidywaniu giełdy



# Contents

<b>1. Introduction</b>	5
1.1. Overview	5
1.2. Contributions	5
1.3. Outline	5
<b>2. Preliminary definitions &amp; guidelines</b>	7
2.1. Macros	7
2.2. Notation	7
2.3. Datasets	7
2.3.1. House sales	7
2.3.2. Google Stock	8
2.3.3. Apple	9
2.3.4. Gold	10
2.3.5. Sine	10
2.3.6. Time Series Practice Dataset	10
2.4. What metrics we used	11
2.4.1. Mean Squared Error	11
2.4.2. $R^2$ Score	12
2.5. How we describe models	12
2.6. Literature review	13
<b>3. Other models</b>	15
3.1. Random forest	15
3.1.1. Results	16
3.2. Logistic regression	16
3.2.1. Results	16
3.3. Support vector machine	17
3.3.1. Parameters	17
3.3.2. Results	17
3.4. Multilayer Perceptron	17
3.4.1. Structure of an MLP	17
3.4.2. Forward Propagation	18
3.4.3. Backpropagation and Training	18
3.4.4. Results	18
3.5. Convolutional neural network	19
3.5.1. Architecture of Convolutional Neural Networks	19
3.5.2. Results	20
3.6. Residual neural network	20

3.6.1. ResNet Architecture . . . . .	20
3.6.2. Results . . . . .	20
<b>4. Large Language Model . . . . .</b>	<b>21</b>
4.1. Vocabulary . . . . .	21
4.2. Overview . . . . .	21
4.3. The Transformer . . . . .	22
4.3.1. Components . . . . .	22
4.4. LLaMA model . . . . .	24
4.4.1. Introduction . . . . .	24
4.4.2. Features . . . . .	24
4.4.3. LLaMA-2 . . . . .	24
4.5. Time-series Embedding . . . . .	24
4.6. Our methodology . . . . .	25
4.7. Results . . . . .	25
<b>5. Conclusion . . . . .</b>	<b>27</b>
<b>Bibliografia . . . . .</b>	<b>29</b>

# 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 analyse 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 analyse 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 applied 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.
- **Plot:** a plot of the numeric features of the dataset, its visual representation.

#### 2.3.1. House sales

- **Source:** House Property Sales Time Series (raw\_sales.csv) :  
<https://www.kaggle.com/datasets/htagholdings/property-sales>

- **Collection method:** Property sales data was gathered for the 2007-2019 period for one specific region <sup>1</sup>
- **Motivation:** A simple dataset for the basic models. A proper, manageable size, once filtered by property and number of bedrooms. Has very few features, making it easier to understand and process.
- **Size:** Consists of 29600 datapoints. Once filtered for property being house, number of bedrooms equal to 3, around 11300 datapoints remain.
- **Features:** The dataset has very few features. Every datapoint consists of:
  - **datesold:** Date at which the sale was made.
  - **postcode:** 4 digit postcode of the suburb where the property was sold (given for reference only).
  - **price:** Price for which the property was sold.
  - **propertyType:** Property type i.e. house or unit.
  - **bedrooms:** Number of bedrooms: 1, 2, 3, 4 or 5.

The dataset was filtered by propertyType and bedrooms, and the postcode was discarded.

- **Characteristic:** The prices rise with time, therefore models might easily learn to always predict "up".
- **Plot:** Below is the plot of prices with respect to time for properties with different numbers of bedrooms.

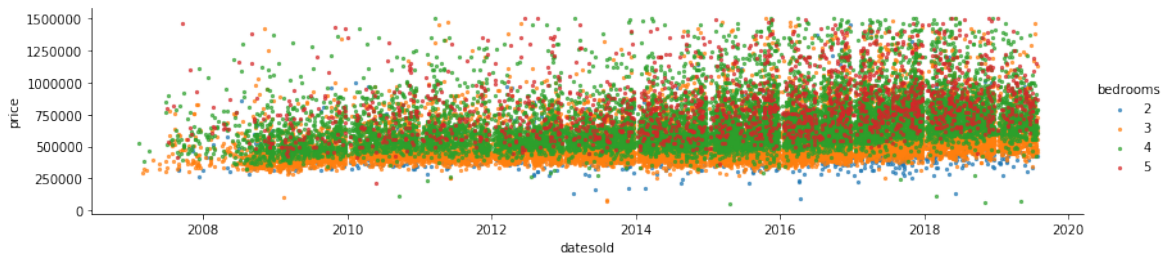


Figure 2.1: A plot of the House Sales dataset.

### 2.3.2. Google Stock

- **Source:** Google Stock Dataset  
<https://www.kaggle.com/datasets/r1shabhgupta/google-stock-price-daily-weekly-and-monthly-2023/data>
- **Collection method:** We suspect this data was collected from the stock market (the source does not specify this). Has daily data and weekly and monthly summaries.
- **Motivation:** More complex dataset, multiple time series numeric features, but fewer datapoints.

---

<sup>1</sup>The region in question was not specified in the description of the dataset.

- **Size:** Daily set has 2510 datapoints. Weekly set has 520 datapoints. Monthly set has 120 datapoints.
- **Features:** Every datapoint has the following features:
  - **Date:** the date on which the stock price was recorded.
  - **Price:** the opening price of the stock on the given date.
  - **High:** the highest price at which the stock was traded during the day.
  - **Low:** the lowest price at which the stock was traded during the day.
  - **Close:** the closing price of the stock on the given date.
  - **Volume:** the total number of shares traded on the given date.
  - **Adj Close:** the close price modified to account for dividends, stock splits, etc., to better reflect stock value.
- **Characteristic:**
- **Plot:**

### 2.3.3. 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 than 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 paid per share to the stockholder.
  - **splitFactor** - Stock split factor - the ratio used to adjust number of shares and their prices during a stock split.

#### 2.3.4. 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).

#### 2.3.5. 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  $\text{target} = \sin(x)$ .
- **Collection method:** Dataset was generated by a script.
- **Characteristic:**
- **Plot:** A simple sine wave.

#### 2.3.6. Time Series Practice Dataset

- **Source:** <https://www.kaggle.com/datasets/samuelcortinhas/time-series-practice-dataset/data?select=train>
- **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). There 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. There 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:

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

where:

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

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

where:

- $y_i$  is the actual value for the  $i$ th data point,
- $\hat{y}_i$  is the predicted value for the  $i$ th data point,
- $\bar{y}$  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.

### 3.1. Random forest

A random forest [1] 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 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. And a best split is chosen from the chosen features of the data. 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  $k = \text{num\_lags}$ , 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 overfitting and decreased randomness of the model.
- 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 [2] is a statistical model used for binary classification. It calculates the probability of whether a datapoint classifies to one class. During training of the model a sigmoid function of features of data is calculated that best fits the provided training dataset.

More precisely, the model calculates the function  $Y(X) = \frac{1}{1+e^{-z}}$ , where

- $z = B_0 + B_1 \cdot X_1 + \dots + B_n \cdot X_n$ ,
- $X_1, \dots, X_n$  are features of data  $X$ ,
- $B_0, B_1, \dots, B_n$  are parameters of the model.

Function  $Y$  assumes values only in the range  $(0, 1)$ . If  $Y(X) \geq 0.5$ , the model classifies the datapoint as 1 (in our model below - the price will be higher). If the converse is true, the datapoint is classified as 0 (the price will be lower).

During training the parameters  $B_0, B_1, \dots, B_n$  are chosen using Maximum Likelihood Estimation function, so that the results of the  $Y$  function best fit the training dataset.

### 3.2.1. Results

Two models have been trained. The first one is trained to predict if the next price in the dataset will be higher than the previous one, provided with a sequence of prices spanning over the last `num_lags` days. The second one does the same but operates on monthly averages instead of single records (for this, the dataset was averaged over subsequent months). The first model proved to be more precise with an accuracy of 82% for an optimal value of `k` of around 40 (Fig 3.1a). At the same time the second model only scored 75% (Fig 3.1b).

However, simply predicting whether the price will be higher or lower is much easier then predicting the actual next price. Additionally, logistic regression assumes an independence of datapoints from each other, wherefore it performs poorly for time series datasets. Therefore this model is insufficient for our purpose.

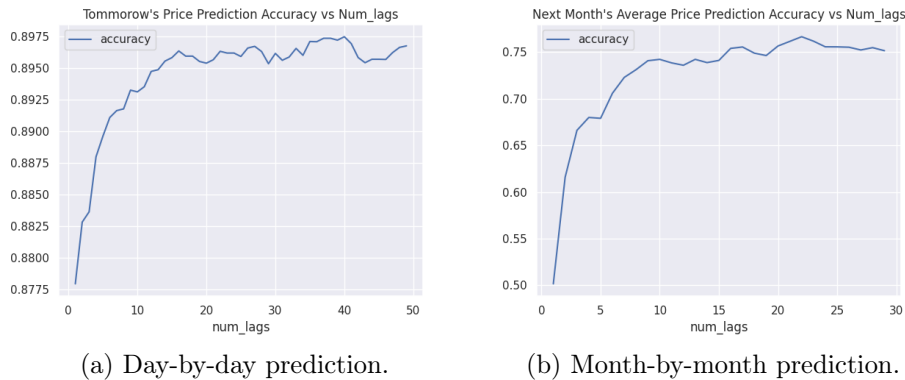


Figure 3.1: Plots of two models of logistic regression.

### 3.3. Support vector machine

Support vector machines [3] are a widely popular model for machine learning classification problems, due to their high generalization abilities and applicability to high-dimensional data.

The model construes datapoints as high-dimensional vectors and finds the best hyperplane that divides the two classes the data is classified into. The goal of training is to find the hyperplane with the greatest margin - that is, the greatest distance from the closest vectors, called the support vectors.

#### 3.3.1. Parameters

- **Kernel:** 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.

Roughly speaking, a kernel  $K(x, y)$  is a real-valued function  $K : X \times X \rightarrow \mathbb{R}$  for which there exists a function  $\Phi : X \rightarrow Z$ , 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  $Z$ . In the SVM literature  $X$  and  $Z$  are called, respectively, input space and feature space. [3]

- **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.
- **Gamma:** The gamma value specifies how much influence individual training datapoints should have. The `scale` value of gamma means that gamma is scaled automatically to fit the size of the dataset.

#### 3.3.2. Results

TBA

### 3.4. Multilayer Perceptron

The Multilayer Perceptron (MLP) [4] 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:

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

where

- $a_j^{(l)}$  is the activation of the  $j$ -th neuron in the  $l$ -th layer,
- $\phi$  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:

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

where

- $\eta$  is the learning rate,
- $\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}}$  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) [5] 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 input image,  $h$  is the filter, and  $*$  denotes the convolution operation.

#### 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. It operates independently on every depth slice of the input and resizes it spatially. A common operation for pooling is max pooling:

$$f(x, y) = \max_{(a,b) \in D} g(x + a, y + b)$$

where  $D$  is the set of coordinates in the input, and  $g$  is the input image.

#### 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 is a softmax layer that outputs a probability distribution over the classes, indicating the likelihood of the input image belonging to each class.

### **3.5.2. Results**

TBA

## **3.6. Residual neural network**

As the number of layers in classic CNN grows, the problem of vanishing or exploding gradient appears. It means that gradients become very small (or very large in case of exploding) as they are propagated through the layers of the network.

Residual neural network (ResNet) is a type of CNN that is less subject to this problem. It was developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in 2015 [6]

### **3.6.1. ResNet Architecture**

ResNet uses a technique called skip connections. Skip connection bypasses several layers, so that input of one layer is added to the output of some further layer. Blocks of such connections are called residual and the network is formed by stacking residual blocks.

The idea is that instead of learning the original mapping of input to output, residual blocks learn the difference between output and input, which is called the residual function.

### **3.6.2. 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. [8]
- **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 [7].
- **Context length** is the maximal size of input tokens a large language model can process at any one time.

### 4.2. Overview

A Large Language Model (LLM) [9] 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. [10]. 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 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** (on the left):
  - Consists of  $N = 6$  layers.
  - Each layer has two sub-layers:
    - **Multi-Head Attention:** Applies attention mechanism over the input. Calculates the weight or importance of each token in its surrounding context (*Attention*). Does so independently for multiple tokens at once (*Multi-Head*). The results are concatenated and linearly transformed.
    - **Feed Forward:** Applies a fully connected feed-forward neural network.
  - **Add & Norm:** Residual connections followed by layer normalization after each sub-layer.
4. **Decoder** (on the right):
  - Also consists of  $N = 6$  layers.
  - Each layer has three sub-layers:
    - **Masked Multi-Head Attention:** As in Encoder, but masked - ensures that the predictions for position  $i$  can depend only on the known outputs at positions less than  $i$ .
    - **Multi-Head Attention:** Attends to the encoder's output.



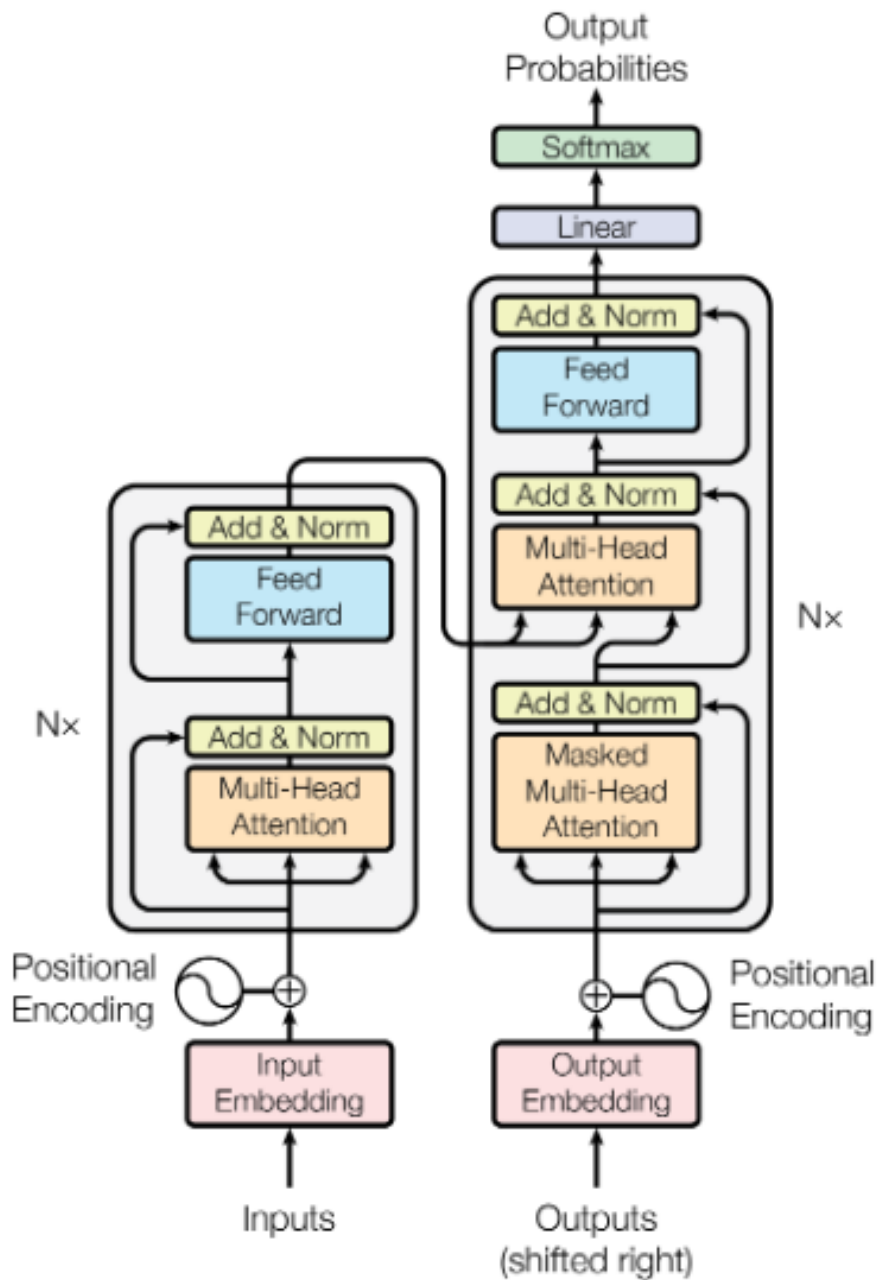


Figure 4.1: The Transformer model architecture

- **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. LLaMA model

### 4.4.1. Introduction

**LLaMA** or *Large Language Model Meta AI* [11] 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.
- **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 [11]). 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 [12].

### 4.4.3. LLaMA-2

**LLaMA-2** [13] 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 in the 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) [14].

What follows is a more detailed description. We consider the following problem: given a sequence of historical observations  $X \in \mathbb{R}^{N \times T}$  consisting of  $N$  different 1-dimensional 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) [15]. 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

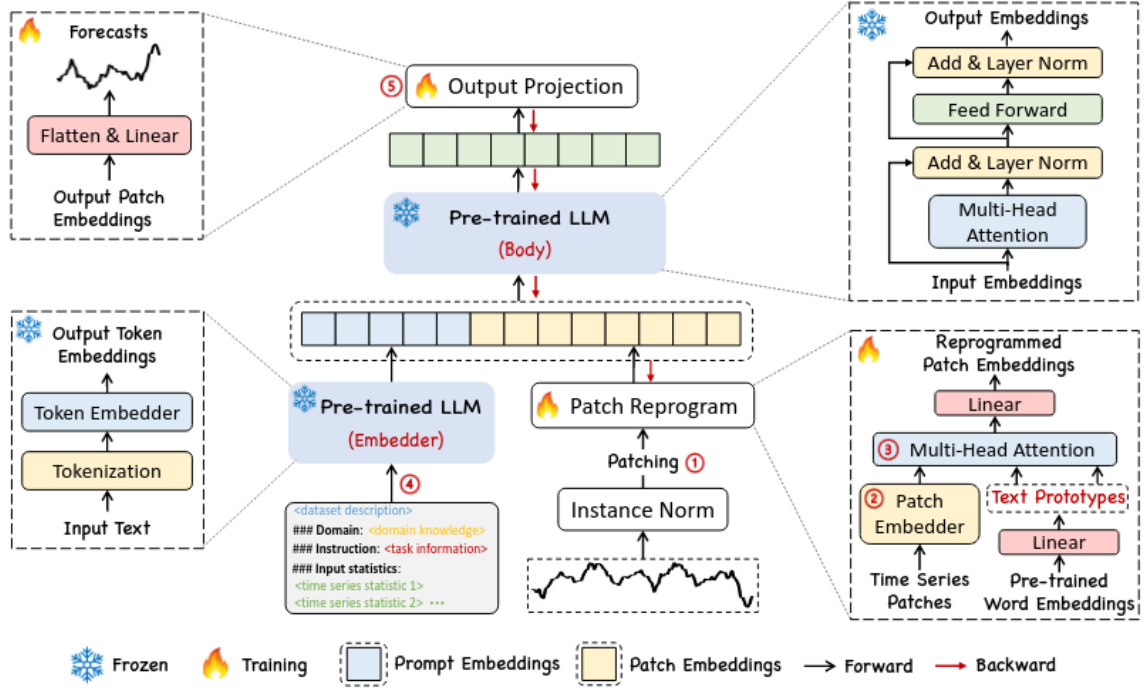


Figure 4.2: The model framework of TIME-LLM. Given an input time series, we first tokenize and embed it via (1) patching along with a (2) customized embedding layer. (3) These patch embeddings are then reprogrammed with condensed text prototypes to align two modalities. To augment the LLM’s reasoning 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.

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

## 4.7. Results



Chapter 5

Conclusion



# Bibliography

- [1] Scornet, E., Biau, G., & Vert, J. P. (2015). Consistency of Random Forests. arXiv preprint arXiv:1511.05741.
- [2] Meng, L., Cao, J., Zhang, C., Yu, S., & Yang, Q. (2020). Sufficient Dimension Reduction for Logistic Regression. arXiv preprint arXiv:2008.13567.
- [3] Steinwart, I., & Christmann, A. (2006). Estimating conditional quantiles with the help of the pinball loss. arXiv preprint math/0612817.
- [4] (Add complete citation when available).
- [5] Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1511.08458.
- [6] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv preprint arXiv:1512.03385.
- [7] Jurafsky, D., & Martin, J. H. (n.d.). Token Embeddings. Retrieved from <https://web.stanford.edu/~jurafsky/slp3/6.pdf>.
- [8] Li, Z., Li, J., & Liu, X. (2023). Efficient Language Models with Dynamic Token Dropping. arXiv preprint arXiv:2303.18223.
- [9] Zhang, Z., Li, X., & Yang, W. (2023). An Introduction to Large Language Models. arXiv preprint arXiv:2304.00612.
- [10] 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.
- [11] 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.
- [12] Meta AI. (2023). LLaMA GitHub Repository. Retrieved from <https://github.com/meta-llama/llama>.
- [13] 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.
- [14] Wang, Y., Xu, J., & Lin, J. (2023). Reprogramming Large Language Models with Synthetic Data. arXiv preprint arXiv:2310.01728.

- [15] 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.