

Using LLMs techniques for Time Series Prediction

Pierre Brugière and Gabriel Turinici

CEREMADE, University Paris Dauphine-PSL

brugiere@ceremade.dauphine.fr, turinici@ceremade.dauphine.fr

17th October 2024

- 1 Introduction
- 2 Objectives of The Model
- 3 Results for the Synthetic Dataset
- 4 Results for the S&P 500
- 5 Strategic Choices and Conclusion

- Large Language Models have demonstrated their capacity to answer complex questions.
- One of their capabilities is to predict the next word in a sentence, attaching a probability to each possible outcome.
- Based on this, the question arises whether LLMs can be used for time series prediction.

When asking ChatGPT-3.5 to predict the next number in a sequence, we get the following responses:

- Question: What is the next number in the sequence: 1, 3, 4, 7, 11, 18, 29?
Answer: 47 ✓
- Question: What is the next number in the sequence: 1, 2, 4, 5, 7, 8, 10, 11?
Answer: 13 ✓

The question is whether the LLM is recognizing a **known example** or if it has learned useful reasoning and **pattern recognition** during training that can be applied to new datasets.

- When asked to predict the number following a sequence of 30 i.i.d. observations $\mathcal{N}(5\%, 15\%)$, with a realized mean of 4.89%, ChatGPT applies a learned "method" for analysis and prediction, providing the following explanations and answers:

Step 1: Calculating the differences between consecutive terms.

Step 2: Detecting patterns: "The differences vary greatly, alternating between positive and negative values."

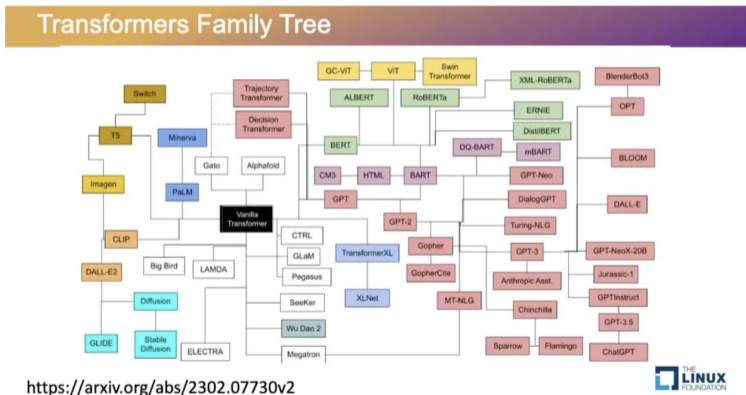
Step 3: Predicting the next value: "After a negative difference, we see a positive jump followed by a smaller negative one. Given this pattern, the next difference could likely be positive, reflecting the alternating nature of the sequence. Assuming the next difference is moderately positive, let's say around 0.1:" ✘

- Therefore, further work is needed before using ChatGPT-3.5 for time series prediction.

- The first step is to define the type of prediction desired:
 - a single value (maximum likelihood estimate, conditional expectation, or an MSE solution)?
 - a value with a confidence interval?
 - a distribution (probability buckets)?
- Here, we will predict a distribution for:
 - the next day's return, and
 - the next day's squared return (which can be used to forecast volatility or the gamma gain of an option position).
- For synthetic data (with a recognizable pattern), our model performs well. ✓
- For market data (using the S&P 500):
 - the predicted distribution for the next day's return is not very accurate ✗
 - the predicted distribution for the next day's squared return performs reasonably well. ✓

- To build the predictor, we will not use a ready-made LLM model but rather the Transformer architecture, which is at the core of ChatGPT and many other LLMs.
- The Transformer we have built corresponds to the one described in "Attention is All You Need" 2017 [1], but with a few adjustments.
- For text-to-text or text-to-image transformations, there are numerous different versions of transformers already in use (see Figure 1).

Figure: 1



- Developing LLM/transformer-based models to address the specific problem of time series prediction is currently an active area of research.
- One approach is to use "Universal LLMs," which have been trained on text, such as GPT-3 or LLaMA-2 [2] and to fine tune them to predict time series, like **LLMTime** [3].
- Another approach is to use transformer-based models dedicated to time series analysis and trained only on time series, like: Autoformer [4], Informer [5], PatchTST [6] and **TimesFM** [2].
- LLMTime [3] and TimesFM [2] have the interesting characteristic of being **zero-shot models**, i.e., they are able to predict time series they have not been trained on.
- Our model is time-series-specific and lightweight. It is trained on a portion of the studied dataset and tested on the remainder (zero-shot performance has not been tested).

Introduction: Two State-of-the-Art (SOTA) Protocols

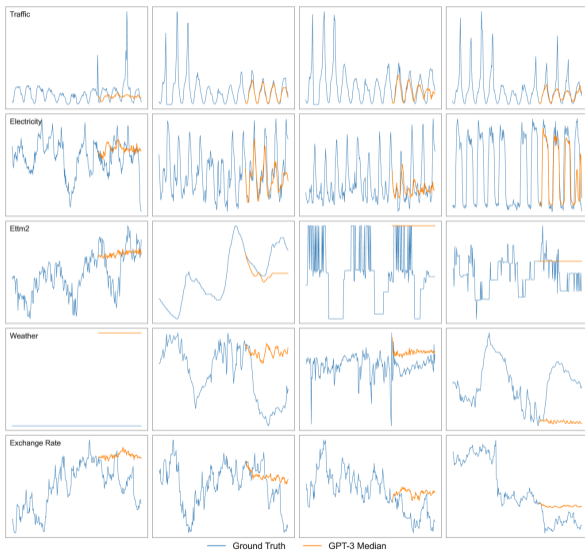
LLMTime (Team from NYU, NeurIPS 2023 Paper [3])

- Pre-processes data: tokenization-compatible transformation of time series, scaling.
- Feeds data into GPT-3 or LLaMA-2 (lesser performance with GPT-4).
- Can convert the discrete probabilities generated by the LLM into continuous densities.
- Can handle efficiently missing data (fills them more effectively than linear interpolation).

Results:

- Tested on 29 datasets (Darts, Monash, Informer).
- Benchmarked against several models (ARIMA, CatBoost, FFNN, etc.).
- LLMTime using GPT-3 or LLaMA-2 ranks first or second.
- However, its applicability to financial time series needs to be demonstrated.

Introduction : Two State of the Art (SOTA) Protocols: LLMTime



LLMTime [[3], fig 13.] The output is the median prediction: performs well with regular patterns (first row), but is not very useful for noisy financial data (last row).

Introduction: Two State-of-the-Art (SOTA) Protocols: TimesFM

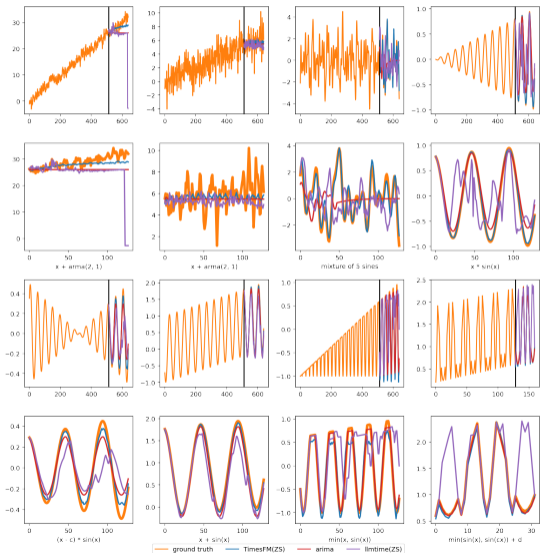
TimesFM (Google Team 2024 Paper [2])

- Trained only on time series ($O(100B)$ timepoints, 200m parameters).
- Can be adapted to predict probabilities, but the paper focuses on point prediction.
- Implements techniques such as scaling, patching, tokenization, and embedding.
- Fills in missing values more effectively than linear interpolation.

Results:

- Evaluated on 29 individual datasets (Darts, Monash, ETT).
- Performs well against other models (LLMTime, ARIMA, CatBoost, FFNN, etc.).
- Achieved first place on Monash and ETT, and third place behind LLMTime and ARIMA on Darts for Mean Absolute Error.
- However, its applicability to financial time series needs to be demonstrated.

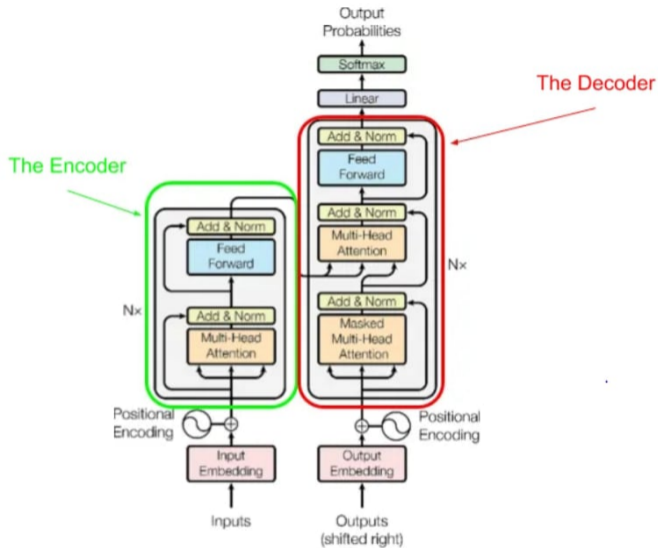
Introduction : two State of the Art (SOTA) protocols: TimesFM



TimesFM [[2], fig 7](Google team): a 'foundation model' usable on several datasets (zero-shot), trained mostly on wikipedia data; ok for periodic structures, less relevant for noisy data.

- The transformer described in "Attention is All You Need" [1] consists of an **encoder** and a **decoder**.
- In LLMs, the encoder is typically responsible for **discriminative tasks** (for example, detecting spam emails), while the decoder handles **generative tasks** (such as autoregressively generating a sequence of words) based on the encoder's output.
- The decoder builds the output sequence element by element, with each generated element used as input for generating the next element in the sequence.
- Here, to make predictions, we use only the encoder part of the model, as we have access to the entire sequence at once to analyze the situation and make the prediction.

Introduction



Objectives of the Model

- Our dataset is a time series of 24,130 observations $[y_1, y_2, \dots, y_{24130}]$.
- The dataset is split into training and test sets (80% for training, 20% for testing).
- Sequences of length $l = 30$ are extracted (we tried both overlapping and non-overlapping sequences).
- The aim of the model is to predict the distribution of y_{m+l} , given $[y_m, \dots, y_{m+l-1}]$.

We start with a synthetic series for which the distribution can be calculated theoretically.

We define seven buckets and compare:

- The theoretical probability of y_{m+l} belonging to these buckets, and
- The predicted probability estimated by the Transformer.

The Transformer performs remarkably well, and the results are explained below.

Results for the Synthetic Dataset

We use an Ornstein-Uhlenbeck process (h_i) defined as:

$$\forall n \in \mathbb{N}, \quad h_{n+1} = h_n + \theta(\mu - h_n)dt + \sigma\sqrt{dt}\epsilon_{n+1}, \quad \text{with } \epsilon_{n+1} \sim \mathcal{N}(0, 1) \quad (1)$$

We take as observations the variables (y_i) , which are the differences between consecutive h_i .

We define $x[i] = [y_i, y_{i+1}, \dots, y_{i+l-1}]$ as the sequence used to predict y_{i+l} .

From the training sample, we construct 7 buckets $]-\infty, \alpha_1], [\alpha_1, \alpha_2], \dots, [\alpha_6, +\infty[$, each containing an equal proportion of the y_i values.

The model estimates the probabilities $(q_1(x[i]), q_2(x[i]), \dots, q_7(x[i]))$ for y_{i+l} to belong to the seven buckets, based on the observation of $x[i]$.

Results for the Synthetic Dataset

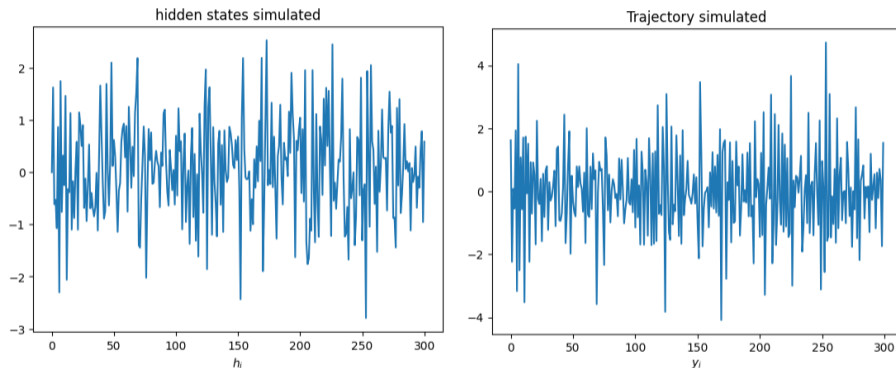


Figure: 5. Trajectory of the process (1) for the parameters $dt = 1$, $\theta = 1$, $\sigma = 1$, $\mu = 0$. **Left:** first 301 hidden values h_0, \dots, h_{300} . **Right:** first 300 values y_1, \dots, y_{300} .

Results for the Synthetic Dataset

To calibrate the model, the **loss function** used is the cross-entropy between the probability estimated by the model Q and the Dirac probability P^δ , which assigns a probability of 1 to the bucket $B(y_{i+l})$ containing y_{i+l} .

$$\text{Loss} = \text{Average on the training sample} \left(H(P^\delta(x[i]), Q(x[i])) \right)$$

with

$$H(P^\delta([x_i]), Q([x_i])) = -\ln(q_{B(y_{i+l})}(x[i]))$$

To evaluate the model, we also calculate its **categorical accuracy**, which is the proportion of y_{i+l} that falls into the bucket for which the model probability Q has the maximum value.

The results obtained are very good. ✓

We denote T as the real probability (calculated knowing the parameters of the OU process).

Number of observations	Train set: Loss $H(P^\delta, Q)$, Accuracy	Test set: Loss $H(P^\delta, Q)$, Accuracy	Train set $H(P^\delta, T)$, $H(T, T)$	Test set $H(P^\delta, T)$, $H(T, T)$
24131	1.681 30.33%	1.697 28.66%	1.626 1.630	1.636 1.628
241310	1.656 30.60%	1.656 30.74%	1.631 1.631	1.630 1.629

Table: 1. Analysis of the results obtained with simulated data.

- The loss $H(P^\delta, Q)$ is very close to the loss $H(P^\delta, T)$. ✓
- The categorical accuracy is approximately 30% with the model and approximately 32% when calculated with the real probability T . ✓
- For comparison, the baseline uniform prediction with probability $\frac{1}{7}$ would yield a loss of $-\ln(\frac{1}{7}) = 1.946$ and a categorical accuracy of $\frac{1}{7} = 14.28\%$.

Results for the Synthetic Dataset

We compare, for each sequence $x[i]$ (associated with the last hidden state h_{i+l-1}) and for each of the 7 buckets:

- The probability calculated by the transformer for y_{i+l} to belong to bucket j , and
- The theoretical values based on all the information about the parameters of the diffusion process and the hidden variables up to time $i + l - 1$.

The results are very good for this lightweight model, which converges after 30 epochs and takes less than 20 seconds to train on Google Colab.

We show in Figures 6 and 7 the following for each $x[i]$ (and associated h_{i+l-1}):

- The probability calculated by the transformer represented as an orange dot, and
- The theoretical probability calculated represented as a blue dot.

Results for the Synthetic Dataset

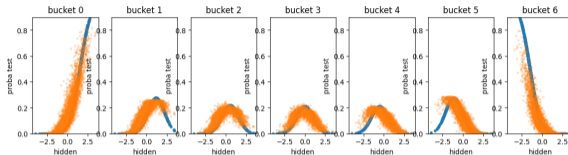


Figure: 6: Predictions for the synthetic stochastic process (1) with 24131 observations, 30 epochs.

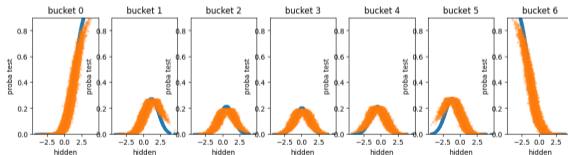


Figure: 7: Predictions for the synthetic stochastic process (1) with 241310 observations, 40 epochs.

Results for the S&P 500

We now make predictions using the closing prices for the S&P 500 Index from December 30, 1927, to February 1, 2024 (24,131 price observations). Here, the variables h_i correspond to the logarithm $\ln(p_i)$ of the observed prices, and we build predictions for the daily log returns defined as

$$y_i = \ln(p_i) - \ln(p_{i-1}) \quad (2)$$

(see Figure 8).

- In the first program, we predict the bucket for y_i , i.e., we try to estimate the probability distribution of the daily log return for the next business day.
- In the second program, we predict the bucket for y_i^2 (the quadratic variation of the log prices) for the next business day.

Results for the S&P 500

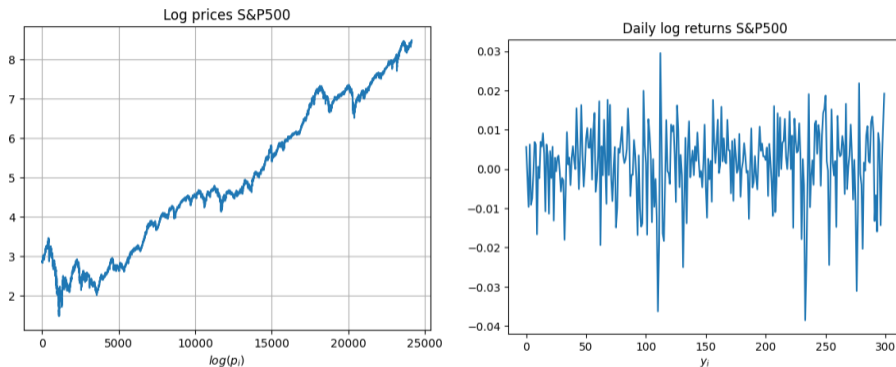
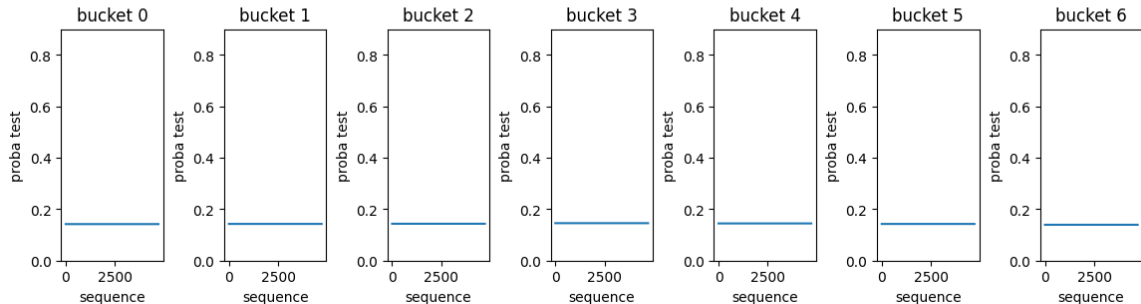


Figure: 8. **Left:** Log prices of the S&P500 over the period. **Right:** first 300 daily log returns y_1, \dots, y_{300} .

Results for the S&P 500

For the bucket prediction of y_{i+l} unfortunately the Transformer is not very helpful. ✘

- The model get stuck in predicting constant probabilities for each bucket.
- The model sees no causality.
- The conditional probability is the same as the unconditional probability.
- The prediction is just the one of a "rational" observer which give to each realization its historical probability.



Results for the S&P 500

For the prediction of y_{i+l}^2 the results are encouraging ✓

Number of epochs	Number of observations	Train set Loss $H(P, Q)$ Accuracy	Test set Loss $H(P, Q)$ Accuracy
50	24131	1,861 21.92%	1.876 22.84%

Table: 2. Bucket predictions for y_i^2 .

The model outperforms:

- The naive prediction, which uses the historical probabilities of occurrence for the buckets (for this model, the loss is 1.9459, and the categorical accuracy is 14.28%).
- The simple prediction, which assigns y_{i+l}^2 to the same bucket as $\frac{1}{l} \sum_{j=i}^{i+l-1} y_j^2$ (for this model, the accuracy is 19.27%).

Results for the S&P 500

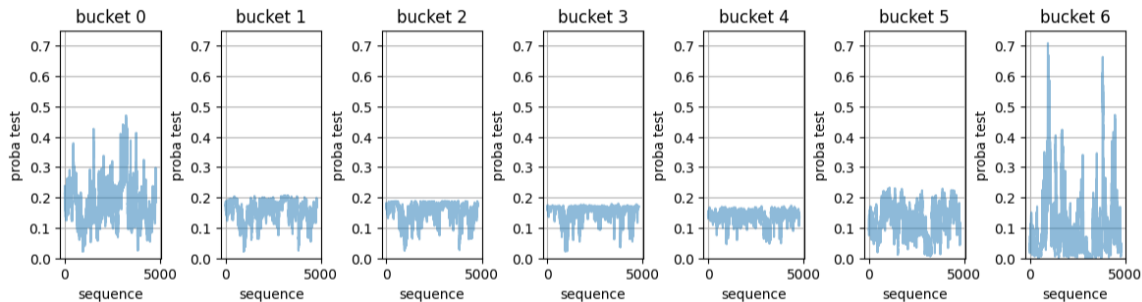


Figure: 10. Bucket predictions for y_{i+1}^2 on the test set for the S&P500.

Strategic Choices and Conclusion

- The architecture of the model we have implemented follows the guidelines of the "Attention is All You Need" [1] Transformer and adheres to the logic of its hyperparameters.
- We did not torture the model to make it work. ✓
- We implemented the layers one by one using Python Keras instead of using a pre-made Transformer model, which may be more difficult to analyze.
- As intended, in our model, the encoder feeds into a two-layer classifier (instead of feeding a decoder).
- The processes that could not translate in a straightforward way from NLP to time series were adjusted, such as the **embedding process**.

We embedded each number of a sequence (in a personal way) in \mathbb{R}^{16} based on some ideas discussed in our paper [7].

$$\phi : x \longrightarrow \left(x, \frac{x^2}{2}, \dots, \frac{x^{16}}{16!} \right)$$

Strategic Choices and Conclusion

Base case NLP encoder	Base case times series prediction
length sequence: $l \sim 1024$	$l = 32$
embedding dimension: $d = 512$	$d = \frac{l}{2} = 16$
Positional Encoding: sinus and cosinus	sinus and cosinus
number of heads: $h = 8$	$h = 8$
head size: $d_k = \frac{d}{N_h} = 64$	$d_k = 64$
number of block iterations: $N = 6$	$N = 6$
number of units FFN: $d_{ff} = 4 \times d = 2048$	$d_{ff} = 4 \times 16 = 64$
.	Dense layer classifier $mlp_units = [10]$

Table: 3. Parameters of the model

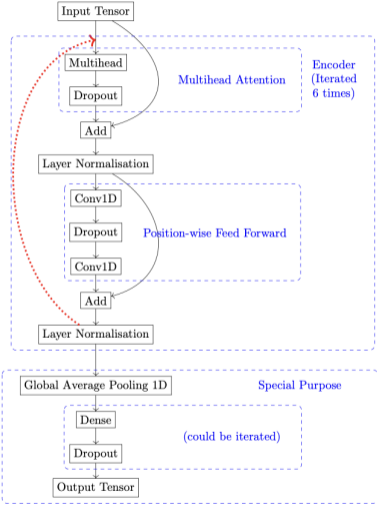
Strategic Choices and Conclusion

- Even though significant progress has been made in predicting certain types of time series, financial time series seem more challenging to handle.
- Some quantities (like volatilities) may also be more endogenous to predict than others (such as the return itself).
- Instead of universal Transformer solutions or Transformer solutions for all time series, we may end up with specific Transformer classes for financial time series prediction.










Thus, further research could integrate ideas such as:

- The optimal way of tokenizing and encoding numbers in a time series.
- The incorporation of other time series to aid in prediction.
- The exploration of transferability and zero-shot prediction for financial time series.
- The integration of a predictive model into a trading strategy.

Appendix: Our model Architecture [7]



Bibliography

-  [1] 2017 : <https://arxiv.org/abs/1706.03762>, *Attention Is All You Need*.
-  [2] 2023 : <https://arxiv.org/abs/2310.10688>, *A decoder-only foundation model for time-series forecasting*.
-  [3] 2023 : <https://arxiv.org/abs/2310.07820>, *Large Language Models Are Zero-Shot Time Series Forecasters*.
-  [4] 2021 : <https://arxiv.org/abs/2106.13008>, *Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting*.
-  [5] 2021 : <https://arxiv.org/abs/2012.07436>, *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting*.
-  [6] 2023 : <https://arxiv.org/abs/2211.14730>, *A Time Series is Worth 64 Words: Long-term Forecasting with Transformers*.
-  [7] 2024 : <https://arxiv.org/abs/2403.02523>, *Transformer for Times Series: an Application to the S&P500*.
-  [8] 2023 : <https://arxiv.org/abs/2302.07730>, *Transformer models: an introduction and catalog*.
-  [9] 2017 : <https://arxiv.org/abs/2302.13971>, *LLaMA: Open and Efficient Foundation Language Models*.

The End