MSc Dissertation Submission Cover Page

Student Name: FILIP TOPIC (BLOCK CAPITALS)

Programme: MSc Banking and Digital Finance *(e.g. MSc GIS)*

Supervisor: Prof. Ramin Okhrati

Dissertation Title:

Time-series Foundation Models for Uni-variate Prediction in Finance

DECLARATION OF OWNERSHIP

- I confirm that I have read and understood the guidelines on plagiarism, that I understand the meaning of plagiarism and that I may be penalised for submitting work that has been plagiarised.
- I declare that all material presented in the accompanying work is entirely my own work except where explicitly and individually indicated and that all sources used in its preparation and all quotations are clearly cited.
- I have submitted an electronic copy of the project report through Moodle/turnitin.

Should this statement prove to be untrue, I recognise the right of the Board of Examiners to recommend what action should be taken in line with UCL's regulations.

Signature: Filip Topić **Date: 12/09/2024**

Time-Series Foundation Models for Univariate Prediction in Finance

by

Filip Topic

September 2024

Dissertation Supervisor: Ramin Okhrati

Submitted in partial fulfilment of the requirements of the degree of MSc in Banking and Digital Finance, University College London

ABSTRACT

This dissertation explores the application of Time-series Foundation models in predicting financial time-series data and compares their performance with traditional time-series forecasting methods such as ARIMA, Prophet, and the simple naive autoregressor. Time-series Foundation Models, built on large-scale machine learning architectures, have shown potential in capturing complex temporal dependencies and patterns, making them promising candidates for financial forecasting. The study evaluates these models across multiple financial datasets across different types of data such as Stock indices, Commodity, Exchange rate and card-spending volumes. Through rigorous empirical analysis, this research provides insights into the strengths and limitations of Time-series Foundation Models, offering guidance on their practical use in financial forecasting. The results highlight the relative performance of these models in varying time-series contexts, ultimately contributing to the broader understanding of Time-series Foundation Model performance in the Finance domain. The results of this research serve as a guide to forecasters in Financial institutions as they provide insight into what type of time-series data do the Time-series Foundation Models perform well.

Among the conclusions of this research is that a Time-series Foundation Model Lag-Llama greatly benefits from fine-tuning and very often outperforms traditional time-series prediction models. Research reveals that fine-tuned Lag-Llama performs best when the underlying time-series being predicted is stationary and doesn't contain linear trend or cyclical patterns. Research also highlights peculiarities around fine-tuning behaviour of this model and contrasts it with another prominent Time-series Foundation model TimeGPT-1, essentially concluding that the zero-shot version of TimeGPT-1 and fine-tuned version of Lag-Llama could be complementary.

Keywords: Time-series Foundation Models, **Finance**, **time-series prediction**

ACKNOWLEDGEMENTS

I want to thank my University Supervisor Prof. Ramin Okhrati and Andrea Bassani from Natwest UK for providing me critical guidance throughout the entire research process as well as all the professors and teaching staff at UCL Institute of Finance and Technology for equipping me with knowledge and skills which were greatly needed to tackle such a complex topic. Finally I extend deep gratitude towards Nixtla.io team for being so kind and allowing extra access to their TimeGPT-1 API which was crucial for this research.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Time-series forecasting is a vital aspect of financial analysis, underpinning decision-making and resource allocation within companies. Accurate predictions of future values based on historical data enable businesses and policymakers to mitigate risks, optimize strategies, and make informed decisions on asset purchases and capital allocations. In particular, time-series forecasting plays a crucial role in portfolio management, hedging strategies, monetary policy formulation, and consumer credit risk assessments.

Historically, extensive efforts have been made to develop time-series prediction methods, many of which have seen significant application in finance. As hardware technology advanced throughout the 20th and 21st centuries, so did the sophistication of time-series models. Although the lines between them may blur, traditional time-series forecasting models can generally be categorized into several groups: statistical models such as ARIMA [5], which rely on regression over time-series lags; machine learning models like XGBoost [8], which employ algorithms such as gradient boosting; Artificial Neural Network (ANN)-based models; and deep learning models such as DeepAR [48].

The paradigm of machine learning and AI fundamentally shifted in 2017 with the introduction of the Transformer architecture, which revolutionized the field of deep learning, initially making its mark in natural language processing (NLP). Unlike recurrent architectures such as RNNs, LSTMs, and GRUs, Transformers employ self-attention mechanisms, allowing them to process entire sequences simultaneously rather than sequentially. This ability to model long-range dependencies and capture intricate patterns has sparked interest in their application beyond NLP, including time-series forecasting.

Time-series Foundation Models (TSFMs), derived from these Transformer architectures, are now emerging as powerful tools for financial forecasting. They have the potential to outperform traditional models by more effectively capturing the non-linear and dynamic nature of financial markets. To understand the motivation for focusing on TSFMs in this research, it is essential to contextualize their development in relation to the challenges of financial time-series forecasting. While traditional models have been effective for certain data types, they often struggle with the non-linearities and volatility characteristic of financial markets. TSFMs, with their ability to capture long-term

dependencies and process data in parallel, are particularly well-suited to modeling the complex, dynamic patterns observed in financial data.

This research compares TSFMs with traditional models, aiming to evaluate whether TSFMs can over-perform established methods, offering a more robust approach to financial forecasting.

1.1 Research Objectives

This dissertation aims to evaluate the performance of Time-series Foundation Models in predicting financial time-series across prominent datasets such as stock index prices, oil prices, exchange rates, and bank card spending volumes. The key research objectives are:

- To assess the predictive accuracy of Time-series Foundation Models compared to traditional models such as ARIMA, Prophet, and the naive autoregressor.
- To analyze the ability of these models to capture complex patterns, seasonality, cyclicality, and trends inherent in financial time-series.
- To provide insights into the implications of using these models in financial forecasting.

1.2 Dissertation Organization

This dissertation is structured into six chapters, with this chapter serving as the introduction. Chapter 2 provides an in-depth exploration of the Transformer architecture, Transformer-based models, and the transformative impact of this architecture on AI. Chapter 3 reviews the relevant literature, focusing on two prominent Time-series Foundation Models central to this research. These models are examined in terms of their architecture and capabilities. Chapter 4 outlines the methodology, describing the financial datasets used, preprocessing steps, and the overall experimentation process. Chapter 5 presents the experimental results and offers a detailed discussion on the performance of TSFMs in comparison to traditional methods. This chapter also analyzes the findings in relation to the research objectives, providing insights into their practical implications. Finally, Chapter 6 concludes the dissertation by summarizing the key contributions and discussing potential avenues for future research, including improvements and expansions of this work.

2. BACKGROUND

Appendix sections A.1, A.2.1, and A.2.2 briefly explain the history of univariate time-series forecasting, along with the development of the first statistical and machine learning time-series prediction models.

2.1 The Transformer

In 2017, Vaswani et al. [62] introduced the ground-breaking "Transformer" model, designed to solve sequence-to-sequence mapping problems. In such problems, both input and output are sequences consisting of tokens (see Appendix B.1.1 for details on tokens) that belong to a specific vocabulary, and both sequences have a specific order. The task of a sequence-to-sequence model is to learn the correct relationship between these input and output sequences so that, when given an unseen input, it can accurately predict the corresponding output. In essence, the model's goal is to capture the "meaning" of the relationship between the sequences. Examples of sequence-to-sequence tasks include:

1. **Language translation** - translating a sentence from one language (input) into another language (output).
2. **Chatbots** - where the input is a question and the output is the appropriate response.
3. **Time-series forecasting** - where the input is a historical time-series, and the output is the predicted future values of that series.

Before the Transformer, several machine learning models attempted to solve these tasks. Sutskever et al. (2014) [56] used multilayer LSTM[1] networks for sequence-to-sequence modeling. While LSTMs [25] and GRUs [10] were considered state-of-the-art at the time, they had several limitations: they required large memory[2], were inherently sequential[3], and suffered from information bottlenecks due to the fixed size of the hidden state (see Appendix B.1.2 for more on hidden states). The Transformer addressed these limitations.

[1] LSTM stands for Long Short-Term Memory cells, a type of RNN (Recurrent Neural Network) architecture.

[2] This made it difficult to parallelize the training process across training examples, necessitating smaller batch sizes.

[3] This made parallelizing computations within each training example impossible.

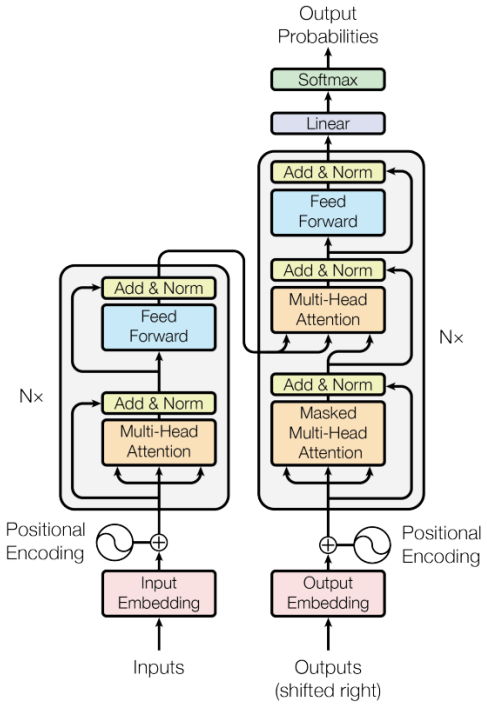

Figure 2.1: Transformer architecture schema

The Transformer consists of two main components: the Encoder and the Decoder. Figure 2.1 illustrates the architecture, with the Encoder on the left and the Decoder on the right.

Encoder

The Encoder is composed of a stack of multiple identical layers[4]. Each layer contains two sub-layers: the Multi-head Self-attention layer and a Feed-forward Neural Network. The Multi-head Self-attention layer consists of N "attention heads"[5]. Each head performs a series of operations on the input sequence. Consider a sequence of tokens $S = \{s_1, s_2, \ldots, s_n\}$, where the dimensionality of each token is $1 \times d_{\text{model}}$[6].

Each attention head uses three learnable weight matrices W^Q, W^K, and W^V[7], each of size $d_{\text{model}} \times d_k$, where $d_k = \frac{d_{\text{model}}}{N}$. Multiplying these matrices by each token s_i yields vectors Q_i, K_i, and V_i, each of dimension $1 \times d_k$. The dot product between Q_i and K_j (for

[4]In the original paper, the Encoder has 6 layers.
[5]In the original paper, $N = 8$.
[6]In the original paper, $d_{\text{model}} = 512$.
[7]Q, K, and V stand for Query, Key, and Value, respectively.

all $j \in \{1, 2, \ldots, n\}$) is then computed, resulting in dot products $\text{dot}_{i,1}, \text{dot}_{i,2}, \ldots, \text{dot}_{i,n}$.

These dot products are scaled[8] and passed through a softmax layer[9]. Each $\text{dot}_{i,j}$ is then multiplied by V_j, yielding a series of vectors $V_{i,1}, V_{i,2}, \ldots, V_{i,n}$, which are summed element-wise to produce Z_i (of size $1 \times d_k$), the self-attention output for token i. This process is repeated for all n tokens, yielding vectors $Z_1, Z_2, \ldots, Z_n$, which are then vertically stacked to form matrix Z of dimension $n \times d_k$. This matrix represents the output of a single attention head.

When vectorizing this process[10], the attention mechanism is expressed as:

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

(See Figure 2.2).

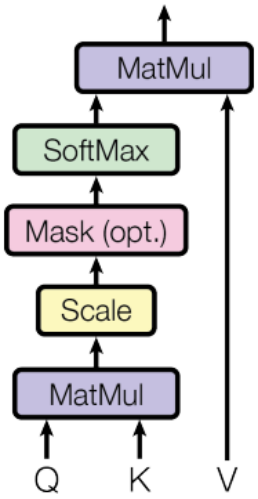

Figure 2.2: Scaled Dot Product Attention

This process runs in parallel across all attention heads in the Multi-head Self-attention layer. The outputs $Z_1, Z_2, \ldots, Z_N$ from all heads are concatenated to form the final output of the Multi-head Self-attention layer, which is then multiplied by a weight matrix W_O of size $Nd_k \times d_{\text{model}}$, producing the final output of the multi-head self attention layer.

[8]The scaling factor is $1/\sqrt{d_k}$, to stabilize weights during training.
[9]Softmax normalizes the numbers so they sum to 1.
[10]Vectorization allows performing the same operations across different vectors simultaneously.

Before this output is passed to the Feed-forward Neural Network[11], a residual connection[12] [23] and layer normalization[13] [3] are applied. These residual connections and normalization steps are applied after every sub-layer in both the Encoder and Decoder.

Decoder

The Decoder shares a similar structure to the Encoder, with a few key differences. It also consists of N layers, but each layer has three sub-layers: Masked Self-attention, Encoder-Decoder Attention, and a Feed-forward Neural Network. The Masked Self-attention layer functions like the Multi-head Self-attention layer in the Encoder, but the attention is only applied to the current and previously generated tokens (i.e. for the currently generated token i, masked self-attention head only calculates $Z_1, Z_2, \ldots, Z_i$), making it suitable for autoregressive tasks.

The Encoder-Decoder Attention layer works similarly to the Multi-head Self-attention layer in the Encoder, but with a crucial difference: Q comes from the Decoder's previous self-attention layer, while K and V come from the Encoder, allowing the Decoder to attend to all positions in the input sequence. (See Figure 2.1.)

Summary

Although attention mechanisms had been used previously [4], the self-attention model introduced in the Transformer was revolutionary because it solved the issue of long-range dependencies[14]. This enabled the model to learn the underlying structure of sequences, moving beyond simple "fitting" to a deeper understanding of the input.

Another breakthrough was the use of positional encoding, which provides the model with information about the relative and absolute positions of tokens in a sequence.

[11]The Feed-forward Neural Network has two hidden layers: the first with ReLU activation, and the second with linear activation. In the original paper, dimensionality of hidden layers is 2048.

[12]A residual connection adds the input of a layer to its output before passing it to the next layer, addressing the vanishing gradient problem and stabilizing training.

[13]Layer normalization scales features within each sample, as opposed to batch normalization, which normalizes across a batch of samples.

[14]Earlier attention models struggled to relate distant tokens due to attention being applied to hidden states rather than directly to inputs.

Positional encoding is achieved by adding a sinusoidal function to each token's vector representation, enabling the model to learn the importance of token order.

Finally, the Transformer's parallelization capability allows for significant scalability, enabling the construction of very large models that can learn more intricate patterns and the use of richer vector representations of input data[15]. This ability to scale has led to widespread adoption of Transformer-based architectures in various fields, including time-series forecasting.

2.2 Transformer-based Models

NLP Transformer-based Models

Since the introduction of the Transformer, many models have been built upon its success, retaining core concepts from the original architecture. The field that has most widely adopted the Transformer is Natural Language Processing (NLP). Among the most notable NLP Transformer-based models are BERT (2018) [15] and T5 (2019) [46], both of which have inspired many variations, including alBERT (2019) [32], roBERTa (2019) [35], distilBERT (2019) [49], mT5 (2020) [68], and flan-T5 (2024) [11]. However, OpenAI's GPT[16] series and the LLaMA series [61] have become the most widely recognized and influential Transformer-based models in the NLP domain.

Computer Vision Transformer-based Models

The Transformer architecture has also made significant strides in the field of Computer Vision (CV). Notable Transformer-based models in CV include ViT (2020) [18], DETR (2020) [7], and the Swin Transformer (2021) [37], each of which has expanded the application of self-attention mechanisms to image recognition and related tasks.

[15] In the original Transformer, vector representations had a dimension of 512, but modern models support much larger dimensions.

[16] GPT stands for Generative Pre-trained Transformer.

Time-series Transformer-based Models

It was only a matter of time before Transformer-based architectures were applied to time-series forecasting tasks. Li et al. (2019) [33] made pioneering contributions with their work on the LogSparse Transformer, which addressed the issue of the Transformer's quadratic space complexity[17] by employing a heuristic approach to reduce the storage complexity to $O(L \log(L))$. They also introduced a more task-specific version of self-attention[18].

Further advancements were made by Zhou et al. (2021) [74], who developed the Informer model. Informer reduced computational and space complexity to $O(L \log(L))$ by utilizing the "ProbSparse" self-attention mechanism[19] and self-attention distilling[20].

Zhou et al. (2022) [75] introduced the FEDformer, a Transformer-based model that incorporates a seasonal-trend decomposition approach [12], [63] and Fourier analysis into the self-attention mechanism[21]. This model demonstrated substantial improvements in time-series forecasting, both in terms of lower prediction errors and the closer alignment of prediction distributions with ground-truth values, as measured by the Kolmogorov-Smirnov test [40].

2.3 Time-series Foundation Models (TSFM)

Foundation Models (FM) are a class of deep learning models pre-trained on vast amounts of data, which enables them to generalize well[22] across different tasks. FMs have shown great success in both Computer Vision and NLP, and it is only natural that their development has begun in the realm of time-series forecasting. Since 2022, more than 50 models classified as Time-series Foundation Models (TSFMs) have been released. According to the taxonomy proposed by Liang et al. (2024) [34], TSFMs can be categorized based on

[17] Quadratic space complexity refers to the quadratic growth in memory usage as the input size increases, making it problematic for longer sequences.

[18] They proposed "convolutional self-attention," which incorporates local context awareness into the Query-Key matching process.

[19] ProbSparse self-attention employs Query Sparsity Measurement to identify the most "dominant" queries, allowing the model to focus on calculating attention for these queries only. This represents an improvement over LogSparse Transformers, which used a heuristic approach to select Query-Key pairs.

[20] Self-attention distilling reduces the number of calculations in each subsequent layer by shrinking the size of the outputs relative to the inputs.

[21] FED stands for Frequency Enhanced Decomposition.

[22] Generalization refers to the model's ability to perform effectively on unseen data.

several criteria:

- **Type of time-series data**, which can be:
  1. **Standard time-series** - a simple sequence of any number of data points, each associated with a time-stamp, ordered chronologically.
  2. **Spatial time-series** - a standard time-series with an additional spatial dimension.
  3. **Trajectory** - a sequence of time-stamped locations that describe the movement of an object through space.
  4. **Event sequence** - a sequence of events arranged in chronological order within a specific context.
- **Model architecture**, which may be:
  1. **Transformer-based**.
  2. **Non-Transformer-based** models (usually MLP, RNN, or CNN-based architectures).
  3. **Diffusion-based models** [53], [24], which are self-supervised models typically used for image generation. These models are trained by adding Gaussian blur to the input sample in a Markov process, followed by using a CNN-based model to learn how to "un-blur" the sample.
- **Nature of pre-training**, which can be:
  1. **Pre-trained LM** (Large Model). These models leverage a pre-trained language model, typically trained on other sequential data, and adapt it for time-series tasks by modifying the tokenization process.
  2. **Self-supervised models**, which can be:
     - **Generative models** - trained to reconstruct input data, making them well-suited for tasks like time-series forecasting.
     - **Contrastive models** - trained to distinguish between similar and dissimilar data pairs, commonly used in anomaly detection tasks.
     - **Hybrid models** - combining both generative and contrastive approaches.
  3. **Fully supervised models**.
- **Adaptation capabilities**, which include:
  1. **Fine-tuning**.
  2. **Zero-shot learning**.

3. **Prompt engineering**.
4. **Tokenization**.

In this dissertation, I will focus on a subset of TSFMs that belong to the class of standard time-series, Transformer-based, self-supervised (Generative) models with fine-tuning capability. This class of TSFMs is quite large, as highlighted by Liang et al. (2024) [34], and includes models such as PatchTST [43], MOIRAI [66], Lag-Llama [45], TimeSiam [17], Timer [36], TimesFM [14], UniTS [19], TimeGPT-1 [20], Chronos [1], and MTSMAE [58]. However, the focus of this research will be on two pioneering models: Lag-Llama and TimeGPT-1, both of which have been trained on vast collections of time-series data spanning multiple domains. The success of Transformer-based models in NLP and CV has generated interest in their application to financial time-series forecasting. Financial time-series data present unique challenges, such as volatility, irregular intervals, and frequent outliers, which traditional models often struggle to handle. TSFMs, leveraging the Transformer's self-attention mechanisms, provide a promising alternative by capturing both short-term fluctuations and long-term dependencies. Additionally, TSFMs can adapt to various time-series characteristics without requiring explicit feature engineering, making them particularly valuable in the financial domain.

3. LITERATURE REVIEW

3.1 TimeGPT-1

TimeGPT-1 [20] is a closed-source model, which means that details about its architecture, parameters, and training data are not publicly available.

Architecture

TimeGPT-1 employs an encoder-decoder architecture with multiple layers, each incorporating residual connections and layer normalization. It uses a self-attention mechanism, similar to the original Transformer architecture [62].

A unique feature of TimeGPT-1 is its ability to perform multivariate time-series forecasting. This allows the model to account for "special events" and exogenous features when predicting the target time-series. Special events refer to time-series assumed to influence the target, such as bank holidays affecting the number of flights per day. Exogenous features, on the other hand, are variables external to the target series but potentially impactful, such as petrol prices influencing the number of cars on the road.

However, forecasting based on exogenous variables presents challenges, as it requires knowledge of their future values—something that is nearly impossible. TimeGPT-1 addresses this by first predicting the future values of exogenous variables and then using these forecasts to make predictions for the target time-series. While innovative, this method introduces additional layers of uncertainty. The accuracy of the target forecast depends not only on the prediction of the exogenous variables but also on the correctness of the relationship between the exogenous and target variables. Essentially, this creates a forecast based on another forecast, which may not always be reliable. In my opinion, this approach should only be applied when there is a high degree of confidence in both the future values of the exogenous variables and the relationships between them and the target variable.

Training Data

The authors claim that TimeGPT-1 was trained on a dataset containing over 100 billion data points, making it the model trained on the largest ammount of time-series data, according to their knowledge. The data spans various domains, including finance, economics, demographics, healthcare, weather, IoT sensor data, energy, web traffic, sales, transport, and banking. Due to the diversity of these domains, the training set includes time-series with a wide range of characteristics such as seasonality, cycles, trends, noise, and outliers. As a result, TimeGPT-1 exhibits strong robustness and generalization capabilities across various types of time-series data.

3.2 Lag-Llama

Tokenization

Lag-Llama [45] constructs "lagged features" from past time-series values based on a frequency-dependent set of lag indices, $L_{\text{indices}} = \{1, \ldots, N\}$. For a data point y_t at timestamp t, the set of lagged features is $\{y_{t-L_{\text{indices}}[i]}\}$, where $0 < i < N + 1$. In addition to lagged features, Lag-Llama constructs "date-time features" F, which for a point y_t, provide information about the time components such as second-of-the-minute, minute-of-the-hour, hour-of-the-day, and month-of-the-year. The final tokenization is achieved by concatenating both the lagged features and date-time features into a single vector.

One of Lag-Llama's key hyperparameters, "Context Length" (cl), determines how many previous time points are considered when making a prediction. For instance, when predicting y_{t+1}, Lag-Llama will use the previous $\{y_{t-cl+1}, y_{t-cl+2}, \ldots, y_t\}$. Important to note, even though Lag-Llama explicitly uses only the past cl points, due to how the tokenization process works - specifically the lagged features, tokens contain information from timestamps older than cl time-points in the past.

Architecture

Lag-Llama is an open-source, decoder-only Time Series Foundation Model (TSFM) built upon the LLaMA architecture [61], but with a reduced number of parameters[23] (2.5 million). Similar to LLaMA, Lag-Llama incorporates key techniques such as RMSnorm [72], RoPE [55], and the SwiGLU activation function [50] [42] (see Appendix C.1).

In addition to its N decoder layers, Lag-Llama features a "parametric distribution head," which is responsible for predicting the distribution parameters of the next time point, given a predefined distribution. The authors chose the Student-t distribution for the distribution head, meaning that during inference, the model predicts the degrees of freedom, mean, and scale of the t-distribution [54]. It then randomly draws n samples from this distribution to provide a probabilistic view of the model's predictions. Since the t-distribution is symmetric, the mean of the distribution is used as the prediction, as it represents the most likely outcome. See Figure 3.1:

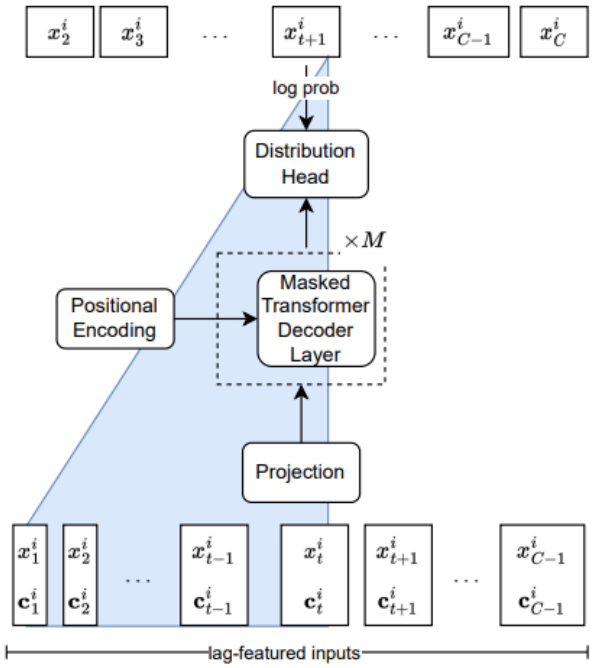

Figure 3.1: Lag-Llama architecture

[23]The original LLaMA models come in parameter sizes of 6.7B, 13.0B, 32.5B, and 65.2B.

Training Data

Lag-Llama was pre-trained on a diverse set of time-series domains, including energy, transportation, economics, nature, air quality, and cloud operations. However, the only finance-related data used in its training was exchange rate data up until 2016.

3.3 Time Series Foundation Models in Finance

There has been significant work on time-series prediction using Foundation Models in finance. However, none of this research has focused on univariate time-series; most approaches have relied on some form of exogenous data, and none specifically utilized dedicated Time Series Foundation Models (TSFMs).

Yu et al. (2023) [69] successfully applied GPT-4 and LLaMA 13B language models with instruction-based fine-tuning, one-shot, and few-shot inference to predict NASDAQ-100 returns. Their approach combined company profiles, financial and economic news, and index price data. However, instead of predicting actual values, they focused on predicting bins and the movement direction of future values. Chen et al. (2023) [9] used ChatGPT-informed Graph Neural Network (GNN), enhanced with prompt engineering and financial news headlines, to predict stock price movement direction. Similarly, Xie et al. (2023) [67] utilized ChatGPT with chain-of-thought-enhanced zero-shot prompting, using Twitter data to forecast stock price movements. Wimmer et al. (2023) [64] employed the CLIP model (pre-trained on computer vision tasks) [44] in combination with an LSTM to predict movements in the German stock market, utilizing Open, High, Low, and Close prices. To the best of my knowledge, this dissertation represents the first work specifically examining the performance of TSFMs for univariate time-series prediction in finance.

4. DATA AND METHODOLOGY

This research aims to answer three main questions:

1. How effective are TSFMs, namely Lag-Llama and TimeGPT-1, on financial time-series data?
2. In which cases do they perform better or worse?
3. How can the performance of TSFMs be improved?

The methodology is designed to address these questions in a systematic and scientific manner.

4.1 Time-series Prediction Evaluation

Time-series predictions are evaluated using traditional regression metrics due to the continuous nature of the target variables. All regression metrics are based on some variation of prediction error, which is the difference between the model's predicted value and the actual (ground-truth) value. From this prediction error, the following metrics are derived and commonly used in time-series forecasting tasks in finance [26]: RMSE (Root Mean Square Error), MAE (Mean Absolute Error), MAPE (Mean Absolute Percentage Error), and R^2.

Additionally, this research will use Mean Directional Accuracy (MDA) (see Appendix D.1.1) as a performance metric. For the calculation of directional accuracy, see [13]. Furthermore, I propose a new metric, Mean Equal Sign (MES), to be used specifically in scenarios where the time-series being predicted represents financial returns. For a set of n time-series predictions $\{y^1_{\text{predicted}}, y^2_{\text{predicted}}, \ldots, y^n_{\text{predicted}}\}$ and ground-truth values $\{y^1_{\text{actual}}, y^2_{\text{actual}}, \ldots, y^n_{\text{actual}}\}$, MES is defined as:

$$\text{MES} = \text{mean}\{\text{sign}(y^1_{\text{predicted}} \times y^1_{\text{actual}}), \ldots, \text{sign}(y^n_{\text{predicted}} \times y^n_{\text{actual}})\}$$

This metric is introduced because MDA alone does not fully capture the model's ability to predict the direction of asset value movements (see Appendix D.1.2). Both MDA and MES are especially important in the field of finance, as the primary interest

is not only in predicting exact asset values but also in forecasting the direction of price movements in the next time step.

Evaluation metrics based on prediction errors are scale-dependent, meaning that the errors themselves are inherently influenced by the scale of the data. Since this research uses datasets of varying frequencies and types, each experiment operates on a different scale. Therefore, raw evaluation metrics cannot be directly used to compare models across different time-series.

It could be argued that MAPE accounts for the difference in scales as it calculates absolute errors as a percentage of ground-truth actual values, however, it doesn't account for the fact that some time-series are inherently more difficult to predict than the others hence the MAPE would be on a different scale on different time-series that reason. A similar argument applies to R^2, which scales the sum of square explained (SSE) by the total sum of squares (TSS), but does not account for variations in predictability across time-series.

For instance, models tend to achieve high R^2 scores on CHAPS data (with some models scoring over 0.7), while most models score negative R^2 on stock index data. For these reasons, each model in an experiment is assigned a rank from 1 to 7 (as there are seven models in each experiment: two zero-shot TSFMs, two fine-tuned TSFMs, and three benchmark models) for each evaluation metric, reflecting its relative performance. This ranking system is a common practice in time-series forecasting evaluations and has been widely used, notably in the M-series competitions [39] [38]. An example of the ranking system is as follows: if a model had the lowest MSE among all other models on a certain experiment on a certain time-series, the model would get rank 1. If the model has the 2nd lowest MSE, it would get rank 2, and so on. Since the ranking scale is constant (1-7), this system can be used to compare the relative performance of models across different time-series. However, a limitation of this approach is that it does not quantify how much better or worse one model is compared to another.

Other scale-invariant metrics such as RMSSE[24] and MASE[25] [28] were considered. However, due to the way these metrics are calculated, they could introduce unfair

[24]Root Mean Square Scaled Error
[25]Mean Absolute Scaled Error

comparisons when applied to non-stationary time-series or time-series with heteroskedastic errors.

4.2 Data

This project utilizes four types of financial time-series data (see Table 4.1). Stock index data was sourced from Yahoo Finance using the 'yfinance' Python package, while commodity and exchange rate data were obtained via the Alpha Vantage Python API. CHAPS[26] data was sourced from the UK Office for National Statistics (see Appendix D.2.1 for more details). Depending on the frequency and type of financial data, different levels of availability (in sufficient ammount) were encountered. Table 4.2 outlines the frequencies available for each type of data, while Table 4.3 specifies the time periods used for each frequency. For exceptions to these tables, refer to D.2.2. In total, 56 distinct time-series were used for this research.

Stock Index	**Commodity**	**Exchange Rate**	**CHAPS**
S&P 500	WTI	USD/GBP	Aggregate
FTSE 100			Delayable
DOWJ			Social
NASDAQ			Staple
			Work-related

Table 4.1: Types of data used

Frequency	**Type of Data**			
	Stock Index	**Commodity**	**Exchange Rate**	**CHAPS**
Daily	YES	YES	YES	YES
Weekly	YES	YES	YES	YES
Monthly	YES	YES	NO	NO

Table 4.2: Available frequencies of different types of data

Daily	**Weekly**	**Monthly**
2018-01-01 to 2020-01-01	2015-01-01 to 2020-01-01	1987-01-01 to 2024-01-01
2020-01-01 to 2022-01-01	2017-01-01 to 2022-01-01	
2022-01-01 to 2024-01-01	2019-01-01 to 2024-01-01	

Table 4.3: Time periods used for different frequencies of data

[26]CHAPS represents UK debit and credit card spending data.

Data Pre-processing

In finance, the concept of "return" is commonly used to represent asset price changes over time. At a given time T, the return R_T of an asset A_T is calculated as:

$$R_T = \frac{A_T - A_{T-1}}{A_{T-1}}$$

This formula captures the percentage change in the asset's value between two consecutive time points $T-1$ and T. In this research, financial time-series (such as stock indices and exchange rates) are transformed into return time-series, a common practice in financial prediction tasks [26]. This transformation is not applied to CHAPS as it wouldn't make sense given the nature of that data. Working with returns rather than raw values also has practical benefits, such as making the time-series stationary, which can enhance model performance.

Time-series Data Characteristics

Time-series data exhibit several key characteristics that are essential for effective analysis and forecasting **Trend** is a long-term increase or decrease in the data, which may take various forms (none, linear, exponential, polynomial). This research uses Kendall's Tau coefficient [30] to detect trends. Kendall's Tau measures the strength of the relationship between two ordinal variables. If a time-series of length L has a statistically significant Kendall's Tau correlation (p-value < 0.05) with a monotonically increasing sequence $\{1, 2, ..., L\}$, this indicates the presence of a linear trend. Similar logic applies for quadratic trends (using square roots of values) and exponential trends (using log-transformed values). However, as most of the time-series in this study contain negative values, quadratic and exponential trends were not tested. **Seasonality** is the presence of recurring patterns at regular intervals in the data. Seasonality is detected by examining partial autocorrelation (PAC) values (see D.2.3 for details). A seasonal pattern is considered present if two or more significant PAC values (> 0.3 or $<$ -0.3) are observed in the data. **Stationarity**: A time-series is said to be stationary if its statistical properties (such as mean and variance) remain constant over time. Stationarity is assessed using the Dickey-Fuller test [16]. Other characteristics such as **Noise** and **Volatility** are also important but are more challenging

to quantify.

Data exploration

56 distinct time-series were used for this research. 38% of all time-series contain linear trend, 62% contain no trend. 46% of time-series contain cyclical patterns and 54% contain no discernable cyclical pattern. 88% of time-series are stationary whereas only 12% are non-stationary (see Figure 4.1).

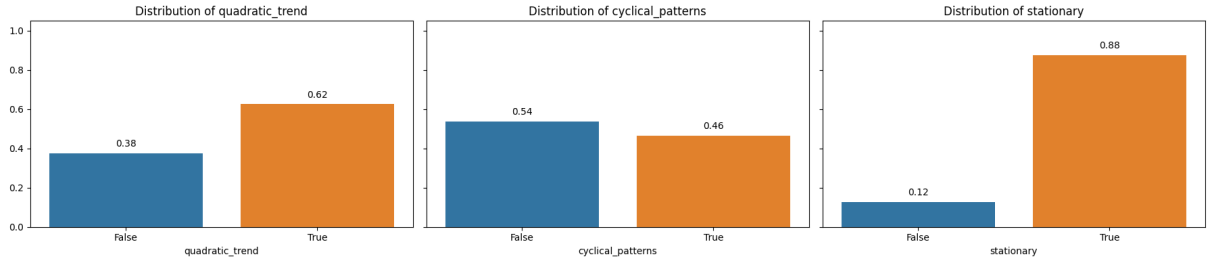

Figure 4.1: Breakdown of all the time/series used by trend, presence of cyclical patterns and stationarity

Half of all time-series used are Stock indices, 27% are CHAPS, 12% Commodity and 11% Exchange rate. 50% of time-series used had daily frequency, 41% had weekly frequency and 0.09% were monthly See Figure 4.2).

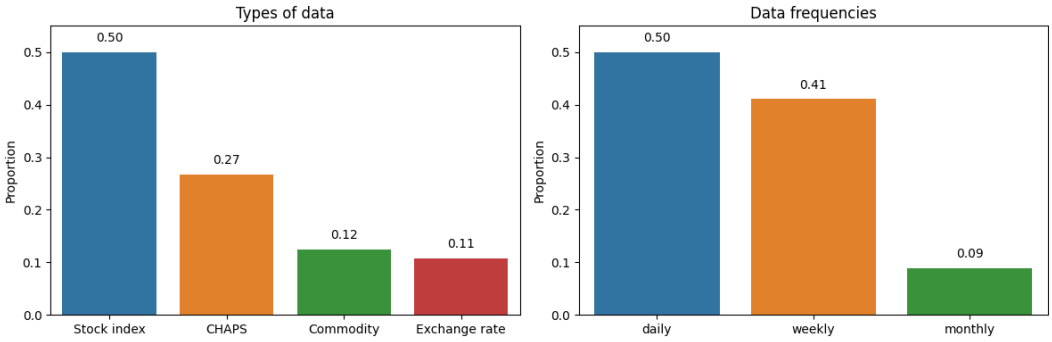

Figure 4.2: Breakdown of all time-series by type of data and frequency.

See Figure 4.3 for breakdown of time-series features by Type of data and Frequency of data. Expected observations are that Commodity, Exchange rate and Stock index (apart from one) data are all stationary and contain no linear trend, and CHAPS data is the only type which contains some non-stationary time-series and some linear trends. This is due to the pre-processing method applied on the non-CHAPS data.

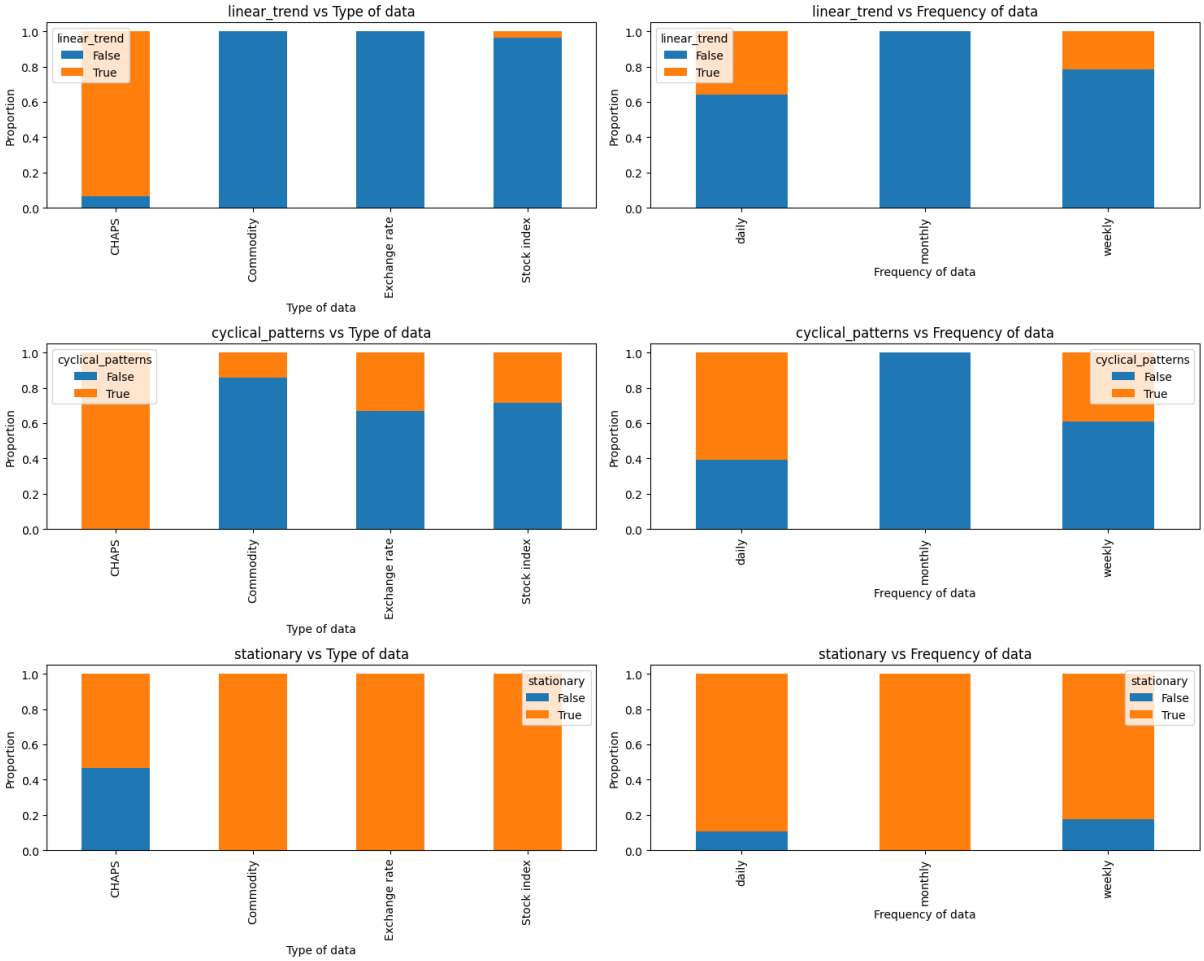

Figure 4.3: Breakdown of all time-series by time-series features, type of data and frequency.

4.3 Experiment Design

Time-series Cross Validation (TSCV)

In traditional machine learning, K-fold cross-validation (CV) is a statistical technique used to evaluate model performance by dividing the dataset into K equally sized subsets, or "folds." The model is trained on K-1 folds and tested on the remaining fold. This process repeats K times, with each fold serving as the test set once, and the results are averaged to

provide a more reliable estimate of model performance. K-fold CV is important because it provides multiple evaluations of the model's ability to generalize to unseen data.

However, in time-series prediction, where data points are temporally dependent, standard K-fold cross-validation cannot be applied directly because randomly partitioning the data would break the temporal order and dependencies. To address this, K-fold cross-validation is modified for time-series by partitioning the data in a way that respects its temporal structure, allowing for a valid evaluation of time-series models while still leveraging the benefits of cross-validation.

Time-series Cross Validation in this Research

In this research, TSCV works as follows: consider a time-series S of length l: $S = \{y_1, y_2, \ldots, y_l\}$. We define n as the length of the training sequence and h as the prediction horizon. The K-fold TSCV method will sample K time-series segments $\{F_1, F_2, \ldots, F_K\}$ from S, where each fold F_i (for $1 \leq i \leq K$) consists of a "train" part and a "test" part. A fold F_i^j starting from timepoint j is defined as:

$$F_i^j = \{y_j, y_{j+1}, \ldots, y_{j+n-1}, y_{j+n}, \ldots, y_{j+n+h-1}\}$$

The model is trained on $\{y_j, y_{j+1}, \ldots, y_{j+n-1}\}$ and tested on $\{y_{j+n}, \ldots, y_{j+n+h-1}\}$, with the size constraint $K \leq \frac{l-n}{h}$. This research focuses on next-point prediction, meaning we use $h = 1$, which aligns with the nature of financial markets where the primary interest is predicting the next value in the time-series.

The TSCV approach used here is a "rolling window" TSCV, where the length l is fixed throughout the TSCV process. Folds are grouped and sampled based on two additional cross-validation parameters: f (number of folds per group) and r (number of groups). Folds within a group are sequential, and the sampling procedure is repeated r times, ensuring that the groups are uniformly distributed across the time-series. The parameters used in this research are $r = 6$ (except for monthly data where $r = 8$) and $f = 5$, meaning five consecutive folds are sampled six times, spread out over the entire time-series. See Figure 4.4 for a schematic view of the TSCV strategy:

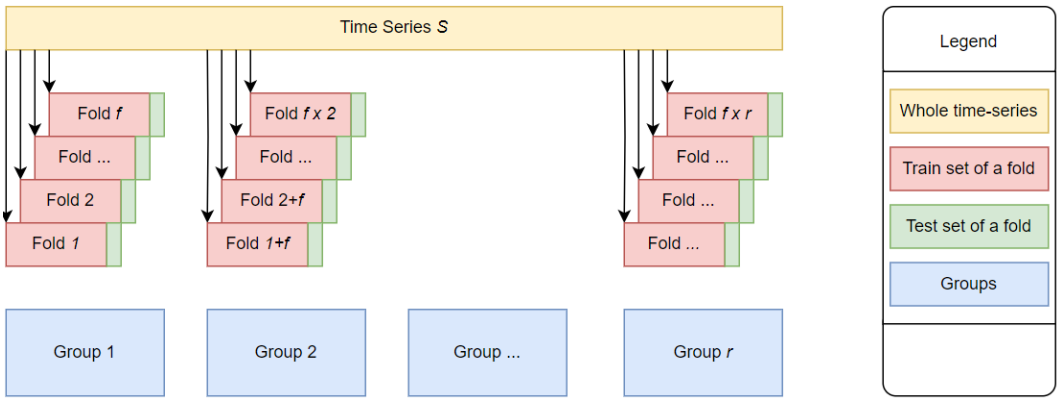

Figure 4.4: Schematic view of the TSCV strategy

Time Series Foundation Models

The central models in this research are Lag-Llama [45] and TimeGPT-1 [20]. Both models have fine-tuning capabilities with associated fine-tuning hyperparameters.

Lag-Llama allows users to configure batch size (BS), maximum epochs (ME), and learning rate (LR). Batch size refers to the number of training examples processed before the model updates its parameters. A smaller batch size results in more frequent updates and more "granular" learning, while a larger batch size processes more data per update. An epoch represents a single pass through the entire dataset during fine-tuning. Before each fine-tuning epoch E_i (for $1 < i \leq ME$), the model records its parameters. After each epoch, the loss L_i is calculated. If $L_{i+1} < L_i$, the updated parameters from epoch E_{i+1} are retained; otherwise, the model reverts to the previous epoch's parameters. This process continues for all $i \leq ME$, ensuring that only the best-performing parameters are retained. A higher number of epochs gives the model more opportunities to learn, but it also increases the risk of overfitting. The learning rate controls how much the model adjusts its weights in response to the loss gradient, with smaller learning rates resulting in slower learning and larger rates potentially causing instability.

TimeGPT-1, on the other hand, allows users to configure only the number of "Fine-tune Steps", which corresponds to the number of epochs. TimeGPT-1 uses the Adam optimizer, but details about other fine-tuning hyperparameters (such as learning rate and batch size) are not publicly available. Therefore, we cannot assume that the same number of epochs would be optimal for both Lag-Llama and TimeGPT-1. According to the

authors of TimeGPT-1 [20], its performance improves with the number of Fine-tune Steps, reaching a plateau at around 100 steps, whereas in this research, Lag-Llama achieves peak performance with only 4 epochs, as we will see later on. To thoroughly investigate model performance, this research evaluates both the zero-shot and fine-tuned versions of Lag-Llama and TimeGPT-1 to determine how fine-tuning impacts their predictive capabilities.

Benchmark Models

Simply calculating evaluation metrics for time-series predictions is insufficient to determine whether a model performs well, as some time-series are inherently more or less predictable than others. Therefore, it is essential to compare the performance of TSFMs against widely-used benchmark models to contextualize their effectiveness. The chosen benchmark models are: ARIMA (with automatic hyperparameter selection) [27][27], Naive Simple Autoregressor (NSA)[28], and Meta's Prophet model [59].

Experiment

"Running an experiment" on a specific time-series S involves applying the TSCV technique, as described earlier. Each fold F_i (where $1 \leq i \leq rf$) represents a slice of S, with a train set of length n and a prediction horizon $h = 1$, defined as[29]:

$$F_i = \{y_1, y_2, \ldots, y_n, y_{n+1}\}$$

The train set T_i of fold F_i is given by:

$$T_i = \{y_1, y_2, \ldots, y_n\}$$

and the test set is:

[27]Originally designed for R, we used the Python equivalent implementation: https://github.com/alkaline-ml/pmdarima.

[28]The NSA predicts the value at time T as the previous value at $T - 1$.

[29]For the sake of simplicity, in the following description, the starting index of each fold is 1.

$$TE_i = \{y_{n+1}\}$$

Initially, the entire train set TR_i is used only for fine-tuning the TSFMs. Subsequently, only the last cl (Context Length, see Section 3.2.1) points of T_i:

$$\{y_{n-cl+1}, y_{n-cl+2}, \ldots, y_n\}$$

are used as the context window for the fine-tuned TSFMs, zero-shot TSFMs, and as the actual train set for the benchmark models (see Section 5.2.1). A schematic view of a single fold in the TSCV process is shown in Figure 4.5.

Once all models have made their predictions for fold F_i, the predictions and the corresponding ground-truth values TE_i are recorded. This process is repeated for every fold in the TSCV process. A noteworthy nuance is that the fine-tuned Lag-Llama is fine-tuned only on the first fold of every group of folds (i.e., every 5 folds), while TimeGPT-1 is fine-tuned on every fold (see Section 5.2.7). After predictions are recorded for all folds, evaluation metrics for each model are calculated, each model is given a rank for each metric, (see Section 4.1) and all this information is documented.

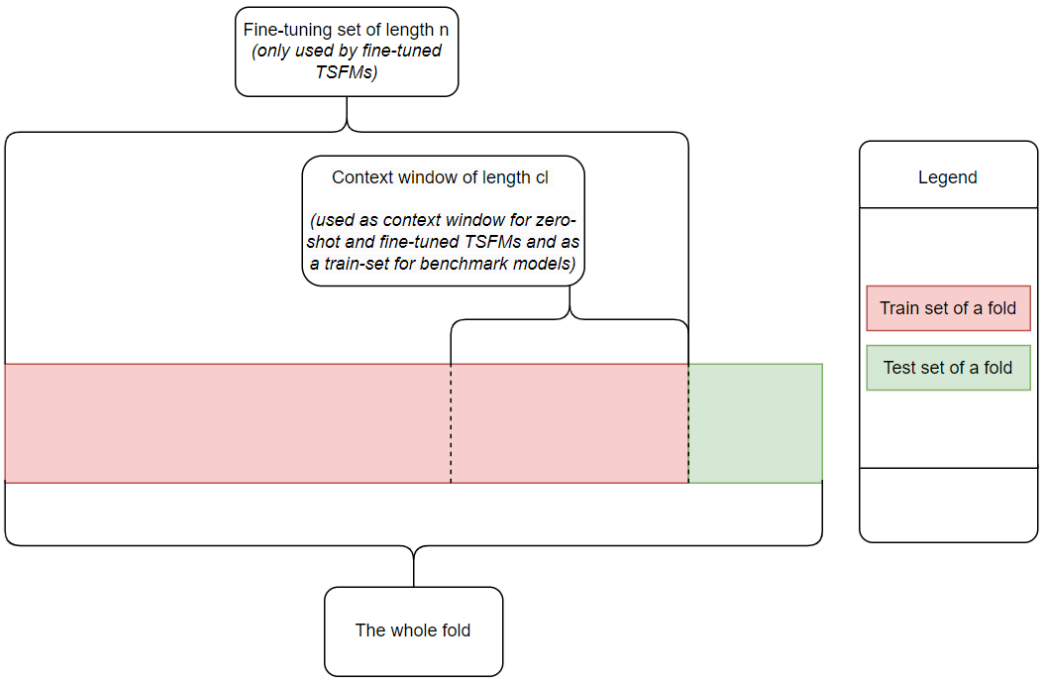

Figure 4.5: Schematic view of a single fold from the TSCV technique

Experiment Configurations

This research consists of numerous experiments (as described above) run across different experiment configurations. An experiment configuration refers to a unique combination of experiment parameters. These parameters can be categorized into two groups: data parameters and fine-tuning parameters.

Data parameters include:

- *Type of Data*
- *Frequency of Data*
- *Context Length*

Fine-tuning parameters include:

- *Fine-tuning Length*: The length of the fine-tuning set used for TSFM fine-tuning.
- *Max Epochs*, *Batch Size*, and *Learning Rate*: Applicable only to Lag-Llama.
- *Fine-tune Steps*: Applicable only to TimeGPT-1.

Refer to Table 4.4 for details on the parameter values used. Running the experiments across a wide range of configurations allows us to analyze the results in terms of different experiment parameters.

Breaking down the results by data parameters answers one of the main research questions: "In which cases do TSFMs perform better or worse?" Analyzing results based on fine-tuning parameters helps answer the second main question: "How can TSFMs' performance be improved?" Investigating the results by multiple parameters provides more granular insights, which is particularly relevant because models may perform differently on time-series of the same frequency but different types, or vice versa.

In total, we have one experiment parameter with 4 values and six parameters with 3 values, resulting in $4x3^6=2916$ different experiment configurations[30], making it computationally unfeasible to run an experiment with every single parameter combination, given that we run all the models apart from TimeGPT-1, locally.

[30]This does not account for the fact that the Stock index data consists of four distinct time-series and CHAPS contains five distinct time-series.

Type of experiment-parameter	Experiment-parameters	Parameter values			
Data-parameters	Type of data	Stock index	Commodity	Exchange rate	CHAPS
	Frequency of data	Daily	Weekly	Monthly	
	Context length	32	64	128	
Fine-tuning parameters	Fine-tuning length	200	128	64	
	Batch size	5	10	20	
	Max epochs	4	8	16	
	Learning rate	0.0005	0.005	0.00005	
	Fine-tune steps	100			

Table 4.4: Different experiment parameters and their values

Data-parameter Experimentation Phase

Due to computational constraints, we divided the experimentation process into phases. The first phase is the Data-parameter experimentation phase, where all fine-tuning parameters (from the last 5 rows of Table 4.4) are kept fixed at their default values (as in the first column of these rows), while the experiments are run with different combinations of data-parameters. See Figure 4.6 for a schematic view.

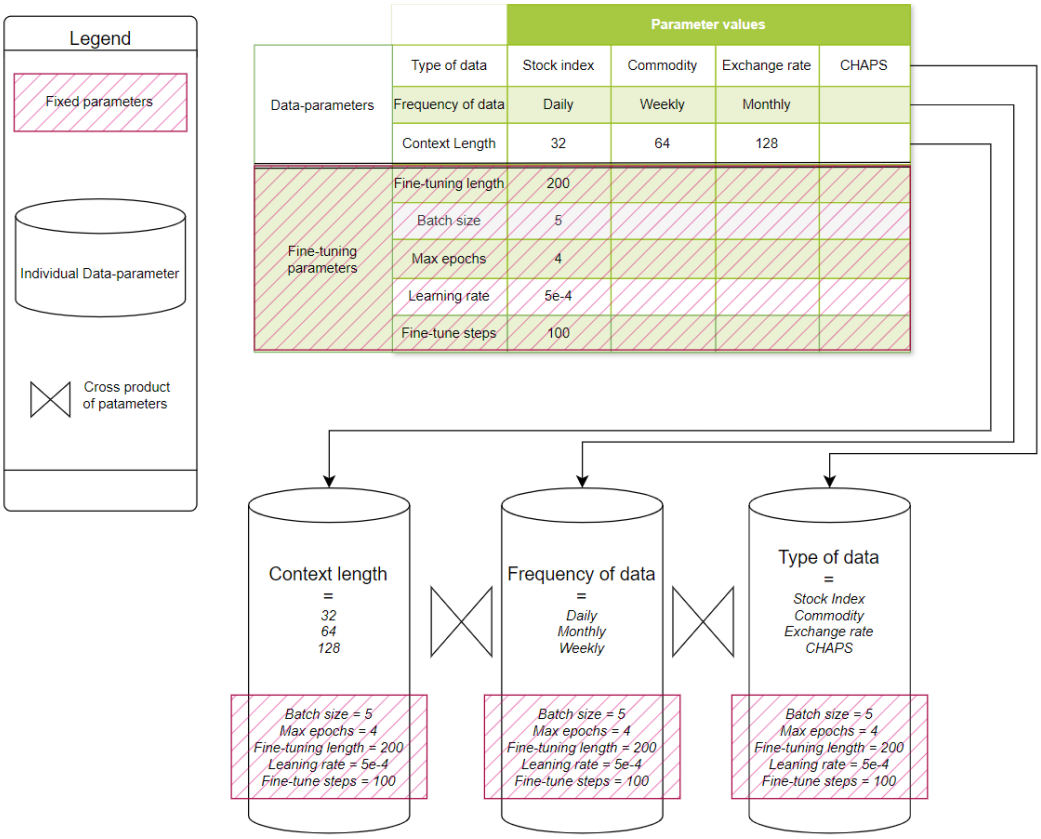

Figure 4.6: Schematic view of the Data-parameter experimentation phase

Aggregation Phase

After recording and aggregating the results from the Data-parameter experimentation phase, the results are grouped by one or more data-parameters, and the mean evaluation metric ranks (as described in Section 4.1) are calculated. This phase constitutes the main body of the research and allows us to compare model performance across different data-parameters.

One limitation of this research is that it is conducted on a relatively small collection of time-series data across a few types (see limitations in Sections 5.2.6 and 5.2.3). To mitigate this, we classify the time-series used according to underlying characteristics such as the presence of trends, stationarity, and cyclical patterns. This classification can help generalize the findings to new datasets with similar characteristics, allowing future users to predict TSFM performance based on the underlying structure of their time-series.

Fine-tuning Parameter Experimentation Phase

Penultimate phase involves fixing a data-configuration and varying the fine-tuning parameters to explore their impact. The data-configuration is chosen based on where the TSFM performed well but with room for improvement. Conceptually, this process is identical to the process in the Data-parameter experimentation phase, but this time the data-parameters are fixed, and we vary the fine-tuning parameters in each experiment. This phase applies only to Lag-Llama due to API limitations for TimeGPT-1 (see Section 5.2.4).

Model-parameter Aggregation Phase

Similar to the Data-parameter Aggregation Phase, the results of the Fine-tuning parameter experimentation phase are grouped by fine-tuning parameters, and the mean evaluation metric ranks are calculated. This analysis enables us to answer the research question: "How to improve TSFMs' performance?" by identifying the fine-tuning parameters that yield the best results.

4.4 Implementation

The experiment framework was implemented in python. See :https://github.com/filip-topic/MSc_dissertation.git

5. RESULTS AND DISCUSSION

5.1 Results

As outlined in Section 4.1, the results of the Data-parameter experimentation phase (Section 4.3.7) are evaluated using the metric ranking system and aggregated as described in Section 4.3.8. These results are presented in Figures 5.1 to 5.10. The results are broken down by various Data-parameters and time-series characteristics. For each combination of Data-parameters and time-series characteristics, the best-performing model for each evaluation metric is highlighted in purple based on the average rank. In some cases, highlights are missing from the MES column where multiple models performed equally well in that group, to avoid confusion.

The **Data-parameter exploration phase results** sub-section presents the main body of the research findings. However, it is important to note that results from the Model-parameter experimentation phase are not included in this section. This is because the time-series chosen for the Model-parameter experimentation phase was specifically selected based on Lag-Llama's strong performance on that time-series, which would bias the overall results. Therefore, the results from this phase are discussed separately in Section 5.1.2.

Data-parameter Experimentation Phase Results

Across all experiments in this phase (see Table 5.1), the ARIMA model performs best overall according to all metrics except for MDA, where Lag-Llama outperforms other models. This pattern remains consistent when breaking down the results by Context Length (see Table 5.2).

Context Length: Across all context lengths (128, 32, and 64), ARIMA performs best in all metrics except MDA. Fine-tuned Lag-Llama excels in MDA when the context length is 128 (mean rank 2.79) or 32 (mean rank 2.95), while zero-shot Lag-Llama performs best with a context length of 64 (mean rank 2.84).

Frequency: Breaking down the results by data frequency (see Table 5.3), ARIMA

performs best on daily data across all metrics (mean rank usually between 1.11 and 2.44), except for MDA. Fine-tuned Lag-Llama, however, leads on monthly (mean rank between 1.33 and 2.00) and weekly (mean rank between 2.94 and 3.01) data in R^2, MSE, MAE, and RMSE, with zero-shot Lag-Llama outperforming in MDA (mean rank 2.52 on weekly and 2.20 on monthly data). ARIMA remains the best for MES.

Type of Data: When results are broken down by data type (see Table 5.4), fine-tuned Lag-Llama tends to be the best model on Commodity, Exchange rate, and Stock index data according to R^2, MSE, MAE, and RMSE (mean best ranks between 1.67 and 2.78). However, ARIMA is equally strong or better in terms of MAE, MAPE, and MES for Stock index data (mean best ranks between 1.15 and 2.49) and performs best in MAPE (mean rank 2.11) and MES (mean rank 1.33) for Exchange rate data.

Daily Frequency by Data Type: In the daily frequency breakdown (see Table 5.5), Naive Simple Autoregressor (NSA) performs best on CHAPS data (mean best ranks between 1.43 and 1.90), fine-tuned Lag-Llama performs best on Commodity data (mean best ranks between 2.11 and 2.89), and ARIMA dominates Stock index data (mean best ranks between 1.17 and 2.14). For Exchange rates, the best-performing model is less clear, with both ARIMA and Prophet showing strong results.

Weekly Frequency by Data Type: For weekly frequency data (see Table 5.6), fine-tuned Lag-Llama consistently outperforms other models on Commodity (mean best ranks between 1.00 and 2.00), Exchange rate (mean best ranks between 1.22 and 1.67), and Stock index data (mean best ranks between 1.83 and 1.94). ARIMA, however, remains the top model for MES. On CHAPS data, it is unclear whether fine-tuned TimeGPT-1 or Naive Simple Autoregressor is the best, as both excel in different metrics.

Monthly Frequency by Data Type: For monthly data (see Table 5.7), no single model stands out for Commodity data, as multiple models perform well across different metrics. Fine-tuned Lag-Llama, however, significantly outperforms on Stock index data (mean best ranks between 1.17 and 2.17), with zero-shot Lag-Llama tied for first place in MDA, and ARIMA beating Lag-Llama only in MES.

Stationarity: When breaking down results by stationarity (see Table 5.8), the Naive Simple Autoregressor performs best on non-stationary data (mean best ranks between

1.76 and 2.95), while ARIMA performs best on stationary data (mean best ranks between 1.27 and 2.65). Fine-tuned Lag-Llama achieves the best MDA score (2.78) on stationary data. Interestingly, fine-tuned Lag-Llama performs relatively better on stationary data, while fine-tuned TimeGPT-1 performs relatively better on non-stationary data.

Presence of a Linear Trend: When a linear trend is present (see Table 5.9), the Naive Simple Autoregressor dominates (mean best ranks between 1.87 and 2.71). However, fine-tuned Lag-Llama performs best on time-series without trends (mean best ranks between 2.28 and 2.35), with ARIMA outperforming in MAPE (2.25) and MES (1.32).

Cyclical Pattern: For time-series without cyclical patterns (see Table 5.10), fine-tuned Lag-Llama is the best model (mean best ranks between 2.26 and 2.42), with ARIMA only outperforming in MAPE (1.97) and MES (1.36). For time-series with cyclical patterns, ARIMA generally performs best (mean best ranks between 1.09 and 2.95), except for fine-tuned TimeGPT-1, has the best MDA (3.26).

If we analyse the results by the **joint effect of time-series characteristics** (see Appendix E, Tables E.1, E.2, E.3), we cannot make judgement on which time-series characteristics are more important as we don't have sufficient combinations of characteristics in order to be able to perform a differential analysis. However, we can say that trend seems to be more important than cyclicality for fine-tuned Lag-Llama performance as the model did better on time-series with no trend and with cyclical patterns than it did on time-series with trend and no cyclicality.

Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
arima	2.71	2.71	2.68	2.71	2.33	5.09	1.23
autoregressor	5.40	5.40	5.32	5.40	5.06	3.94	3.79
ft_lag_llama	3.38	3.38	3.45	3.38	3.50	2.92	1.92
ft_timeGPT	3.57	3.57	3.64	3.57	3.65	3.58	3.93
lag_llama	4.52	4.52	4.58	4.52	4.67	3.15	2.93
prophet	4.48	4.48	4.52	4.48	4.83	3.32	4.44
timeGPT	3.94	3.94	3.82	3.94	3.96	4.25	3.52

Table 5.1: Overall results of the exploration phase

Context length	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
128	arima	2.68	2.68	2.64	2.68	2.18	4.88	1.50
	autoregressor	5.46	5.46	5.43	5.46	5.23	4.11	3.73
	ft_lag_llama	3.30	3.30	3.23	3.30	3.52	2.79	1.80
	ft_timeGPT	3.71	3.71	3.82	3.71	3.77	3.71	3.91
	lag_llama	4.34	4.34	4.39	4.34	4.50	2.91	2.91
	prophet	4.43	4.43	4.57	4.43	4.82	3.54	3.82
	timeGPT	4.07	4.07	3.91	4.07	3.98	4.39	3.43
32	arima	2.91	2.91	2.91	2.91	2.55	5.46	1.04
	autoregressor	5.38	5.38	5.20	5.38	4.95	3.59	3.79
	ft_lag_llama	3.21	3.21	3.34	3.21	3.34	2.95	2.00
	ft_timeGPT	3.23	3.23	3.32	3.23	3.48	3.38	3.95
	lag_llama	5.05	5.05	5.09	5.05	5.02	3.70	3.09
	prophet	4.52	4.52	4.54	4.52	4.73	3.05	4.80
	timeGPT	3.70	3.70	3.61	3.70	3.93	4.00	3.57
64	arima	2.55	2.55	2.48	2.55	2.25	4.93	1.16
	autoregressor	5.38	5.38	5.32	5.38	5.00	4.13	3.86
	ft_lag_llama	3.61	3.61	3.79	3.61	3.64	3.04	1.96
	ft_timeGPT	3.75	3.75	3.77	3.75	3.71	3.64	3.95
	lag_llama	4.18	4.18	4.25	4.18	4.48	2.84	2.80
	prophet	4.48	4.48	4.45	4.48	4.93	3.38	4.70
	timeGPT	4.05	4.05	3.95	4.05	3.98	4.36	3.55

Table 5.2: Breakdown of results by different Context lengths

Frequency of data	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
daily	arima	2.44	2.44	2.39	2.44	2.32	4.68	1.11
	autoregressor	5.15	5.15	4.98	5.15	4.65	3.83	3.63
	ft_lag_llama	4.04	4.04	4.13	4.04	4.42	3.25	1.71
	ft_timeGPT	3.62	3.62	3.42	3.62	3.62	3.65	3.32
	lag_llama	4.02	4.02	4.25	4.02	3.90	3.83	3.31
	prophet	5.15	5.15	5.27	5.15	5.51	2.93	4.67
	timeGPT	3.57	3.57	3.56	3.57	3.57	4.15	3.29
monthly	arima	2.20	2.20	2.40	2.20	2.07	5.53	1.33
	autoregressor	6.33	6.33	6.33	6.33	6.27	4.80	4.33
	ft_lag_llama	1.33	1.33	2.00	1.33	1.13	2.73	2.07
	ft_timeGPT	3.73	3.73	3.53	3.73	3.27	4.40	5.13
	lag_llama	3.13	3.13	2.47	3.13	4.73	2.20	2.20
	prophet	5.93	5.93	5.93	5.93	5.73	2.60	5.93
	timeGPT	5.33	5.33	5.33	5.33	4.80	4.27	4.60
weekly	arima	3.16	3.16	3.09	3.16	2.39	5.49	1.36
	autoregressor	5.51	5.51	5.51	5.51	5.29	3.88	3.87
	ft_lag_llama	3.01	3.01	2.94	3.01	2.90	2.57	2.14
	ft_timeGPT	3.46	3.46	3.93	3.46	3.78	3.30	4.42
	lag_llama	5.43	5.43	5.43	5.43	5.58	2.52	2.64
	prophet	3.33	3.33	3.29	3.33	3.80	3.96	3.84
	timeGPT	4.09	4.09	3.81	4.09	4.26	4.36	3.57

Table 5.3: Breakdown of results by different Frequency of data

Type of data	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
CHAPS	arima	2.96	2.96	2.78	2.96	2.76	2.78	1.00
	autoregressor	2.11	2.11	1.67	2.11	1.62	2.78	1.00
	ft_lag_llama	6.04	6.04	6.18	6.04	6.27	4.56	1.00
	ft_timeGPT	2.87	2.87	2.93	2.87	2.93	3.60	1.00
	lag_llama	5.36	5.36	5.71	5.36	5.64	4.80	1.00
	prophet	5.87	5.87	5.78	5.87	5.82	4.13	3.67
	timeGPT	2.80	2.80	2.96	2.80	2.96	2.98	1.00
Commodity	arima	3.19	3.19	3.14	3.19	3.00	5.48	1.95
	autoregressor	6.14	6.14	6.29	6.14	5.81	3.90	3.90
	ft_lag_llama	2.38	2.38	2.24	2.38	2.00	3.14	2.29
	ft_timeGPT	4.05	4.05	4.10	4.05	4.33	4.29	4.48
	lag_llama	4.00	4.00	3.57	4.00	4.19	2.71	3.71
	prophet	4.76	4.76	4.81	4.76	4.81	2.95	5.62
	timeGPT	3.48	3.48	3.86	3.48	3.86	3.81	4.00
Exchange rate	arima	3.17	3.17	2.78	3.17	2.11	6.00	1.33
	autoregressor	6.50	6.50	6.39	6.50	6.44	4.50	4.28
	ft_lag_llama	2.78	2.78	2.56	2.78	2.28	2.50	3.33
	ft_timeGPT	2.83	2.83	3.17	2.83	4.11	2.78	3.83
	lag_llama	5.11	5.11	5.28	5.11	4.94	2.94	4.39
	prophet	3.28	3.28	3.94	3.28	4.22	2.94	4.44
	timeGPT	4.33	4.33	3.89	4.33	3.89	4.89	4.22
Stock index	arima	2.37	2.37	2.49	2.37	1.98	6.04	1.15
	autoregressor	6.75	6.75	6.80	6.75	6.42	4.45	5.15
	ft_lag_llama	2.32	2.32	2.49	2.32	2.65	2.08	2.02
	ft_timeGPT	3.98	3.98	4.00	3.98	3.77	3.56	5.39
	lag_llama	4.08	4.08	4.07	4.08	4.20	2.42	3.46
	prophet	3.92	3.92	3.89	3.92	4.43	3.06	4.56
	timeGPT	4.58	4.58	4.26	4.58	4.55	4.90	4.60

Table 5.4: Breakdown of results by different Type of data

Frequency of data	Type of data	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
daily	CHAPS	arima	2.67	2.67	2.43	2.67	2.43	2.23	1.00
		autoregressor	1.90	1.90	1.50	1.90	1.43	2.33	1.00
		ft_lag_llama	5.57	5.57	5.77	5.57	5.90	5.03	1.00
		ft_timeGPT	3.33	3.33	3.27	3.33	3.27	3.77	1.00
		lag_llama	5.03	5.03	5.57	5.03	5.47	5.30	1.00
		prophet	6.57	6.57	6.47	6.57	6.53	3.87	5.00
		timeGPT	2.93	2.93	3.00	2.93	2.97	3.10	1.00
	Commodity	arima	3.00	3.00	3.11	3.00	3.44	6.67	1.22
		autoregressor	6.78	6.78	6.67	6.78	5.22	3.67	2.89
		ft_lag_llama	2.89	2.89	2.78	2.89	3.33	2.11	2.22
		ft_timeGPT	3.67	3.67	3.22	3.67	4.11	3.44	4.67
		lag_llama	3.67	3.67	3.33	3.67	3.22	3.56	5.56
		prophet	4.67	4.67	4.56	4.67	5.44	2.56	5.56
		timeGPT	3.33	3.33	4.33	3.33	3.22	5.00	4.56
	Exchange rate	arima	3.22	3.22	2.56	3.22	1.89	6.00	1.11
		autoregressor	7.00	7.00	6.78	7.00	6.89	3.89	4.22
		ft_lag_llama	3.89	3.89	3.67	3.89	3.33	2.22	2.22
		ft_timeGPT	3.56	3.56	3.44	3.56	4.56	4.11	5.33
		lag_llama	3.22	3.22	3.56	3.22	2.89	3.56	4.33
		prophet	3.11	3.11	4.67	3.11	5.22	2.00	4.56
		timeGPT	4.00	4.00	3.33	4.00	3.22	4.56	4.78
	Stock index	arima	1.92	1.92	2.14	1.92	2.06	5.89	1.17
		autoregressor	7.00	7.00	7.00	7.00	6.64	5.11	5.86
		ft_lag_llama	3.08	3.08	3.22	3.08	3.72	2.31	2.06
		ft_timeGPT	3.86	3.86	3.58	3.86	3.56	3.50	4.42
		lag_llama	3.47	3.47	3.56	3.47	3.03	2.75	4.42
		prophet	4.61	4.61	4.61	4.61	4.75	2.47	4.19
		timeGPT	4.06	4.06	3.89	4.06	4.25	4.72	4.50

Table 5.5: Breakdown of "daily" results by different Types of data

Frequency of data	Type of data	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
weekly	CHAPS	arima	3.53	3.53	3.47	3.53	3.40	3.87	1.00
		autoregressor	2.53	2.53	2.00	2.53	2.00	3.67	1.00
		ft_lag_llama	7.00	7.00	7.00	7.00	7.00	3.60	1.00
		ft_timeGPT	1.93	1.93	2.27	1.93	2.27	3.27	1.00
		lag_llama	6.00	6.00	6.00	6.00	6.00	3.80	1.00
		prophet	4.47	4.47	4.40	4.47	4.40	4.67	1.00
		timeGPT	2.53	2.53	2.87	2.53	2.93	2.73	1.00
	Commodity	arima	4.11	4.11	3.44	4.11	2.56	5.33	2.44
		autoregressor	5.44	5.44	5.89	5.44	6.11	3.78	4.22
		ft_lag_llama	2.00	2.00	1.67	2.00	1.00	3.56	2.67
		ft_timeGPT	4.33	4.33	4.89	4.33	5.22	4.56	4.56
		lag_llama	4.56	4.56	4.56	4.56	5.22	2.00	2.67
		prophet	4.56	4.56	4.67	4.56	3.89	3.67	5.33
		timeGPT	3.00	3.00	2.89	3.00	4.00	3.11	3.22
	Exchange rate	arima	3.11	3.11	3.00	3.11	2.33	6.00	1.56
		autoregressor	6.00	6.00	6.00	6.00	6.00	5.11	4.33
		ft_lag_llama	1.67	1.67	1.44	1.67	1.22	2.78	4.44
		ft_timeGPT	2.11	2.11	2.89	2.11	3.67	1.44	2.33
		lag_llama	7.00	7.00	7.00	7.00	7.00	2.33	4.44
		prophet	3.44	3.44	3.22	3.44	3.22	3.89	4.33
		timeGPT	4.67	4.67	4.44	4.67	4.56	5.22	3.67
	Stock index	arima	2.78	2.78	2.86	2.78	1.94	6.08	1.19
		autoregressor	6.64	6.64	6.75	6.64	6.28	3.69	4.86
		ft_lag_llama	1.94	1.94	1.94	1.94	2.08	1.83	1.92
		ft_timeGPT	4.22	4.22	4.64	4.22	4.08	3.47	6.33
		lag_llama	5.03	5.03	5.03	5.03	5.14	2.17	2.86
		prophet	2.53	2.53	2.50	2.53	3.67	3.75	4.53
		timeGPT	4.86	4.86	4.28	4.86	4.81	5.14	4.69

Table 5.6: Breakdown of "weekly" results by different Types of data

Frequency of data	Type of data	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
monthly	Commodity	arima	1.00	1.00	2.33	1.00	3.00	2.33	2.67
		autoregressor	6.33	6.33	6.33	6.33	6.67	5.00	6.00
		ft_lag_llama	2.00	2.00	2.33	2.00	1.00	5.00	1.33
		ft_timeGPT	4.33	4.33	4.33	4.33	2.33	6.00	3.67
		lag_llama	3.33	3.33	1.33	3.33	4.00	2.33	1.33
		prophet	5.67	5.67	6.00	5.67	5.67	2.00	6.67
		timeGPT	5.33	5.33	5.33	5.33	5.33	2.33	4.67
	Stock index	arima	2.50	2.50	2.42	2.50	1.83	6.33	1.00
		autoregressor	6.33	6.33	6.33	6.33	6.17	4.75	3.92
		ft_lag_llama	1.17	1.17	1.92	1.17	1.17	2.17	2.25
		ft_timeGPT	3.58	3.58	3.33	3.58	3.50	4.00	5.50
		lag_llama	3.08	3.08	2.75	3.08	4.92	2.17	2.42
		prophet	6.00	6.00	5.92	6.00	5.75	2.75	5.75
		timeGPT	5.33	5.33	5.33	5.33	4.67	4.75	4.58

Table 5.7: Breakdown of "monthly" results by different Types of data

Stationary data	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
FALSE	arima	3.19	3.19	3.00	3.19	3.00	3.38	1.00
	autoregressor	2.24	2.24	1.76	2.24	1.76	2.95	1.00
	ft_lag_llama	6.57	6.57	6.57	6.57	6.57	3.90	1.00
	ft_timeGPT	2.48	2.48	2.71	2.48	2.71	3.71	1.00
	lag_llama	5.81	5.81	5.86	5.81	5.86	4.19	1.00
	prophet	5.29	5.29	5.24	5.29	5.24	4.29	2.71
	timeGPT	2.43	2.43	2.86	2.43	2.86	3.05	1.00
TRUE	arima	2.65	2.65	2.63	2.65	2.23	5.33	1.27
	autoregressor	5.86	5.86	5.82	5.86	5.53	4.08	4.19
	ft_lag_llama	2.92	2.92	3.01	2.92	3.06	2.78	2.05
	ft_timeGPT	3.72	3.72	3.77	3.72	3.79	3.56	4.35
	lag_llama	4.34	4.34	4.39	4.34	4.50	3.00	3.21
	prophet	4.36	4.36	4.41	4.36	4.77	3.18	4.69
	timeGPT	4.16	4.16	3.96	4.16	4.12	4.42	3.88

Table 5.8: Breakdown of results by stationarity

Trend	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
linear	arima	2.64	2.64	2.51	2.64	2.53	2.98	1.00
	autoregressor	2.22	2.22	1.89	2.22	1.87	2.71	1.29
	ft_lag_llama	6.18	6.18	6.20	6.18	6.09	4.69	1.07
	ft_timeGPT	2.91	2.91	3.04	2.91	3.11	3.64	1.13
	lag_llama	5.38	5.38	5.56	5.38	5.53	4.73	1.29
	prophet	5.84	5.84	5.64	5.84	5.67	3.98	3.58
	timeGPT	2.82	2.82	3.16	2.82	3.20	2.93	1.29
no trend	arima	2.74	2.74	2.74	2.74	2.25	5.86	1.32
	autoregressor	6.57	6.57	6.57	6.57	6.23	4.39	4.71
	ft_lag_llama	2.35	2.35	2.45	2.35	2.55	2.28	2.24
	ft_timeGPT	3.80	3.80	3.85	3.80	3.85	3.55	4.96
	lag_llama	4.21	4.21	4.22	4.21	4.35	2.57	3.54
	prophet	3.98	3.98	4.11	3.98	4.52	3.08	4.76
	timeGPT	4.35	4.35	4.07	4.35	4.24	4.73	4.33

Table 5.9: Breakdown of results by trend

Cyclical patterns	Model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
FALSE	arima	2.51	2.51	2.56	2.51	1.97	5.87	1.36
	autoregressor	6.53	6.53	6.59	6.53	6.40	4.14	4.82
	ft_lag_llama	2.42	2.42	2.41	2.42	2.26	2.26	2.39
	ft_timeGPT	3.77	3.77	3.96	3.77	3.84	3.86	5.04
	lag_llama	4.40	4.40	4.36	4.40	4.62	2.56	3.58
	prophet	4.07	4.07	4.03	4.07	4.38	3.13	4.68
	timeGPT	4.30	4.30	4.10	4.30	4.53	4.53	4.21
TRUE	arima	2.95	2.95	2.82	2.95	2.74	4.19	1.09
	autoregressor	4.10	4.10	3.85	4.10	3.51	3.71	2.60
	ft_lag_llama	4.47	4.47	4.65	4.47	4.94	3.69	1.38
	ft_timeGPT	3.33	3.33	3.27	3.33	3.44	3.26	2.65
	lag_llama	4.67	4.67	4.83	4.67	4.72	3.83	2.19
	prophet	4.95	4.95	5.08	4.95	5.35	3.54	4.17
	timeGPT	3.53	3.53	3.50	3.53	3.31	3.92	2.72

Table 5.10: Breakdown of results by presence of cyclical patterns

Model-parameter Experimentation Phase Results

The time-series selected for Lag-Llama fine-tuning hyperparameter optimization was a Commodity time-series with weekly frequency, spanning the period from 01-01-2019 to 01-01-2024, with a context length of 32. Since there are four Lag-Llama fine-tuning parameters being tested (see Table 4.4), each with three possible values, a total of $3^4 = 81$ experiments were conducted. In almost all fine-tuning cases, there is a significant improvement over the zero-shot version.

Because we are working with a single time-series, we can examine the actual performance metrics of fine-tuned Lag-Llama rather than using ranking. When breaking down the fine-tuning results by *Max Epochs* (see Table 5.12), the model performs best with lower and medium epochs of 4 and 8. The reason for this is that a Max Epoch value of 16 increases the likelihood of overfitting, leading to worse predictions on the test sets.

Breaking down the results by *Fine-tune Length* (see Table 5.13), Lag-Llama shows its best performance with a fine-tune length of 200. However, fine-tune length of 128 performs best for R^2 and MDA, while 64 is best for MES, suggesting there may be an inflection point where increasing the amount of fine-tuning data begins to hurt model performance. This could be due to the model learning outdated patterns from longer fine-tune sets, causing it to make incorrect predictions.

Regarding the *Learning Rate* (see Table 5.14), there is no clear optimal value, as both the lowest and highest learning rates produce nearly identical performance. A similar phenomenon is observed with *Batch Size* (see Table 5.11), where the model exhibits nearly identical performance across all batch sizes. This phenomenon, along with the broader implications of these findings, is discussed further in Section 5.3.5.

BS	model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
20	ft_lag_llama	-0.0099	0.1149	0.0710	0.1938	1.0000	0.5597	0.5128
10	ft_lag_llama	-0.0099	0.1149	0.0710	0.1938	1.0000	0.5597	0.5128
5	ft_lag_llama	-0.0099	0.1149	0.0710	0.1938	1.0000	0.5119	0.5128
NA	zs_lag_llama	-0.3148	0.1143	0.0752	0.1987	6.0655	0.5243	0.5111

Table 5.11: Lag-Llama batch size length optimization

ME	model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
16	ft_lag_llama	-0.0113	0.1652	0.0883	0.2625	1.0000	0.5517	0.5333
8	ft_lag_llama	-0.0095	0.0997	0.0658	0.1732	1.0000	0.5540	0.5067
4	ft_lag_llama	-0.0095	0.0997	0.0658	0.1732	1.0000	0.5287	0.5067
NA	zs_lag_llama	-0.3045	0.1209	0.0774	0.2076	5.8807	0.5223	0.5136

Table 5.12: Lag-Llama max epoch optimization

FTL	model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
200	ft_lag_llama	-0.0069	0.0015	0.0321	0.0392	1.0000	0.5435	0.4667
128	ft_lag_llama	-0.0018	0.0022	0.0333	0.0470	1.0000	0.5632	0.5000
64	ft_lag_llama	-0.0251	0.4919	0.1995	0.7014	1.0000	0.5249	0.6333
NA	zs_lag_llama	-0.2384	0.1641	0.0913	0.2658	4.4970	0.5253	0.5309

Table 5.13: Lag-Llama fine-tune length optimization

LR	model	R^2	MSE	MAE	RMSE	MAPE	MDA	MES
5.00E-05	ft_lag_llama	-0.0069	0.0015	0.0321	0.0392	1.0000	0.5575	0.4667
0.0005	ft_lag_llama	-0.0113	0.1652	0.0883	0.2625	1.0000	0.5364	0.5333
0.005	ft_lag_llama	-0.0069	0.0015	0.0321	0.0392	1.0000	0.5632	0.4667
NA	zs_lag_llama	-0.3964	0.0562	0.0564	0.1202	7.8557	0.5162	0.4881

Table 5.14: Lag-Llama Learning rate optimization

5.2 Limitations

Information Disparity Between Fine-tuned TSFMs and Benchmark Models + Zero-shot TSFMs

Theoretically, fine-tuned TSFMs are systematically advantaged over benchmark models and zero-shot TSFMs because they have access to more information. Fine-tuned TSFMs use the entire train set TR_i for fine-tuning, whereas benchmark models and zero-shot TSFMs only access the last *context length* data points from TR_i. This disparity arises from the technical implementation of fine-tuning, where the fine-tuning set must be larger than the context window of zero-shot models. While this information disparity is unavoidable, allowing benchmark models to train on the full TR_i would disadvantage zero-shot TSFMs. Conversely, letting zero-shot TSFMs use a context length equal to the fine-tuning set length would lead to unfair comparisons with fine-tuned TSFMs due to different context lengths and varying direct information inputs.

Statistical Significance

Due to the nature of this research, it is challenging to quantify the statistical significance of results within each time-series. Statistical significance in time-series prediction is typically measured by analyzing performance across multiple time-series and calculating metrics such as mean, variance, and confidence intervals. In this case, however, the time-series used are highly diverse in terms of their characteristics, and there are not enough similar time-series (see Section 5.2.3) to enable robust hypothesis testing. Instead, we attempted to achieve statistical significance through the use of TSCV (see Figure 4.4), spreading the 6 groups of 5 predictions uniformly across each time-series. This approach minimizes the likelihood of obtaining localized results by covering the largest period possible.

Computational Resource Limitations

A significant limitation of this research is the lack of computational resources. Running the experiments is time-consuming, with each experiment taking between 10 to 45 minutes

depending on Max Epochs and Batch Size, since Lag-Llama is run and fine-tuned locally. This constraint necessitated several compromises:

- Limiting the number of time-series used.
- Reducing the parameter space explored in the experiments.
- Employing a heuristic approach by fixing model-parameters and varying data-parameters, and vice versa.
- Fine-tuning Lag-Llama only every five predictions, rather than before each prediction (see Section 5.2.7).
- Avoiding multiple repeats of the same experiment, which could have provided more statistical confidence in the results.
- Limiting each time-series to 30 predictions.

TimeGPT-1 API Call Limitations

Another limitation is the restricted number of API calls available for TimeGPT-1. The paid subscription allows for a maximum of 10,000 calls per month, which was increased to 30,000 upon request. However, even 30,000 calls were insufficient to fully complete the research. Each TimeGPT-1 prediction requires 2 API calls, meaning that each experiment uses 4 API calls (2 for zero-shot and 2 for fine-tuned versions). With 30 predictions per experiment, this results in 120 API calls per experiment. The 30,000 limit allowed for approximately 250 experiments, but after code testing and failed attempts, the remaining calls were barely enough to complete the research. As a result, certain avenues of research, such as TimeGPT-1 fine-tuning parameter optimization and exogenous variables analysis, had to be excluded.

TimeGPT-1 Pre-training Data

A third limitation is the lack of transparency regarding TimeGPT-1's pre-training data. Without public information, we cannot be certain that TimeGPT-1 was not already trained on the data used for this research, which would result in data leakage. A query was sent to the TimeGPT-1 team (Nixtla Labs), but they refused to disclose this information. Based on its overall performance, it does not seem that TimeGPT-1 had prior exposure

to the dataset. One potential solution could have been using only the most recent data for the experiments, but this was limited by both API and data availability constraints.

Data Availability

Data availability posed another limitation. The AlphaVantage API does not provide monthly exchange rate data prior to 2015, which is insufficient for experiments. Similarly, CHAPS monthly data is only available from 2020 onwards, which also limited proper experimentation. Additionally, the AlphaVantage API only provides commodity price data for higher frequencies for WTI and BRENT, but not for other commodities.

Fine-tuning Disparity Between Lag-Llama and TimeGPT-1

Ideally, fine-tuned versions of TSFMs should be fine-tuned on every fold, as having some distance between the fine-tuning data and the prediction time-point increases the likelihood of learning outdated patterns. However, due to the computational time required for fine-tuning Lag-Llama, it was unfeasible to fine-tune the model before every prediction. As a compromise, Lag-Llama was fine-tuned every five predictions. Despite this, however, fine-tuned Lag-Llama perform excellent.

Data Issues

There are some discrepancies in the frequency of the data. For example, daily Stock Index, Commodity, and CHAPS data are only available on weekdays, while daily exchange rate data is available on weekends as well. Ideally, predictions for time-series with the same frequency should be evaluated on the same set of days, but this was not possible due to the varying data densities. Avoiding weekends entirely would have introduced another bias into the research. Additionally, certain instances of daily commodity data contained 2-3 missing values, which were imputed using the previous value. This method was chosen because some models in this research cannot handle irregular missing data, though it is not an ideal solution. Other imputation techniques, such as averaging or using the median, were deemed less suitable.

5.3 Discussion

Overall Results Discussion

Fine-tuned Lag-Llama consistently performs best on Commodity, Exchange rate, and Stock index data (see Table 5.4). This can be attributed to the characteristics of these data types. Figure 4.3 shows that these data types generally lack cyclical patterns, with Commodity data having the fewest. Notably, fine-tuned Lag-Llama outperforms the next best model by a significant margin on Commodity data (0.81 in RMSE), compared to Exchange rate (0.05) and Stock index (0.05).

A similar pattern emerges when analyzing results by data frequency. Lag-Llama performs best on monthly and weekly data (see Table 5.3), which also have fewer cyclical patterns (Figure 4.3). Interestingly, fine-tuned Lag-Llama shows greater improvement on monthly data (0.87 RMSE difference) compared to weekly data (0.15 RMSE difference), reinforcing that it excels with non-cyclical data. This is further evidenced by Table 5.10, where fine-tuned Lag-Llama is one of the top models for non-cyclical series but struggles with cyclical data.

When considering linear trends, fine-tuned Lag-Llama performs poorly on time-series with such trends, as seen with Stock index, Exchange rate, and Commodity data, which typically lack trends. Similarly, Lag-Llama excels on monthly frequency data, which generally lacks trends, but performs worse on daily frequency data, where nearly 40% of the series show linear trends. A similar case is observed with stationarity: fine-tuned Lag-Llama performs well on stationary data but struggles on non-stationary series.

TimeGPT-1, however, delivers somewhat disappointing results. It rarely outperforms other models, and when it does, the patterns seem arbitrary, suggesting that its strong performance may be coincidental. The only consistent success for fine-tuned TimeGPT-1 was with weekly CHAPS data, where it achieved an R^2 score of 1.93 across multiple metrics, with the second-best model scoring 2.53. When examining the effect of time-series characteristics (Tables 5.8 to 5.10), fine-tuned TimeGPT-1 behaves opposite to Lag-Llama: it performs better on non-stationary data and series with linear trends and cyclical patterns.

General Lag-Llama MDA Performance

Both the fine-tuned and zero-shot versions of Lag-Llama excel in terms of MDA (Mean Directional Accuracy) across various configurations, with fine-tuned Lag-Llama ranking as the best model overall (Table 5.1). Lag-Llama also dominates MDA rankings in all context lengths (Table 5.2), monthly and weekly frequencies (Table 5.3), and data types such as Commodity, Exchange rate, and Stock index (Table 5.4).

Given that Lag-Llama performs best on stationary data with no trend and no cyclical patterns, it is not surprising that it ranks highest in MDA across all these cases (Tables 5.8, 5.9, 5.10). Interestingly, zero-shot Lag-Llama consistently ranks second in terms of MDA in these cases, indicating that Lag-Llama architecture, in general, has a superior ability to predict the direction of a time-series' movement when these conditions are met.

Inter-model Fine-tuning Performance Discussion

Across most results, Lag-Llama benefits from fine-tuning, with few exceptions: daily and weekly CHAPS data, and daily Exchange rate data, where the zero-shot version performs better (Tables 5.5, 5.6). When breaking down by time-series characteristics, zero-shot Lag-Llama outperforms the fine-tuned version on non-stationary data (Table 5.8), data with linear trends (Table 5.9), and performs almost equally on cyclical series (Table 5.10).

This is noteworthy, as one would expect fine-tuning to generally improve a model's performance. However, these results suggest that Lag-Llama's fine-tuning may degrade performance on non-stationary, linear-trend, or cyclical data. This highlights Lag-Llama's strengths in learning from stationary, trendless, and non-cyclical patterns, and its weaknesses when faced with the opposite characteristics. Lastly, it is interesting that there is a degree of complimentarity between zero-shot and fine-tuned versions of the same model and the fact that fine-tuning is not strictly better in every single scenario[31].

TimeGPT-1 only shows marginal improvement from fine-tuning. The most notable improvements occur on weekly Exchange rate and monthly data (Tables 5.6, 5.7), while

[31]Although overall Lag-Llama does benefit from fine-tuning. Just not in every single case as we point out here.

the zero-shot version outperforms fine-tuned TimeGPT-1 on daily CHAPS and daily/weekly Commodity (Tables 5.5, 5.6). Additionally, on characteristics where fine-tuned TimeGPT-1 performs relatively well (non-stationary and linear-trend series), the zero-shot version actually slightly outperforms it according to a few metrics (Tables 5.8, 5.9), a contrast to Lag-Llama's behavior, where fine-tuned versions dominated on their preferred characteristics.

Lag-Llama Fine-tuning Results Discussion

Overall, Lag-Llama benefits most from fine-tuning on stationary, trendless, and non-cyclical data. On these time-series, it outperforms the zero-shot version. However, for non-stationary, linear-trend, and cyclical series, the zero-shot version performs better, suggesting that fine-tuning does not enhance its ability to handle these types of data.

Deeper Investigation of Lag-Llama Fine-tuning Results

An interesting observation is the similarity in many of the fine-tuning results. Tables 5.11 and 5.12 show near-identical outcomes, with only minor differences in the MDA column. Initially, this could suggest an error in the result recording process, but after thorough investigation, it became clear that no mistake occurred. Rather, an unusual phenomenon was at play.

In the majority of these experiments, Lag-Llama consistently predicted values close to 10^{-14} (near-zero), indicating it was making cautious predictions. These small differences, sometimes in the 14th, 15th, or 16th decimal place, resulted in almost identical evaluation metrics when rounded to four decimal places (see Tables 5.11 to 5.14). The reason for this behavior likely lies in the nature of the weekly WTI returns time-series, which is highly unpredictable. The model likely avoided large errors by sticking to near-median predictions.

Evidence for this hypothesis is that a period of high volatility occurred around the one-third mark of the time-series, where WTI returns sharply fell and rose to over 350% (see Figure 5.2). This volatile period was included in the fine-tuning process in many cases (depending on which fold and fine-tune length), likely causing the model weights to

get stuck in a deep local minimum, where the model deteriorated to making, near-median (median is near 0) predictions (see Figure 5.1). Despite this, the model still managed to achieve MDA greater than 50% in most configurations, meaning it could still predict the correct direction of the time-series movement more than half the time, even if the magnitude was inaccurate.

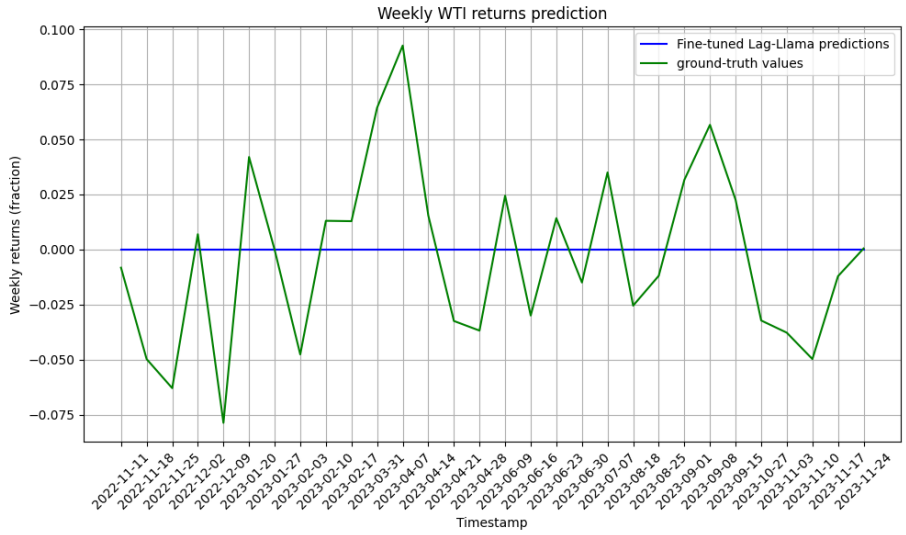

Figure 5.1: Predictions of Lag-Llama with Batch size 5, fine-tune length 200, Max epochs 8 and Learning rate 0.0005 on weekly WTI returns

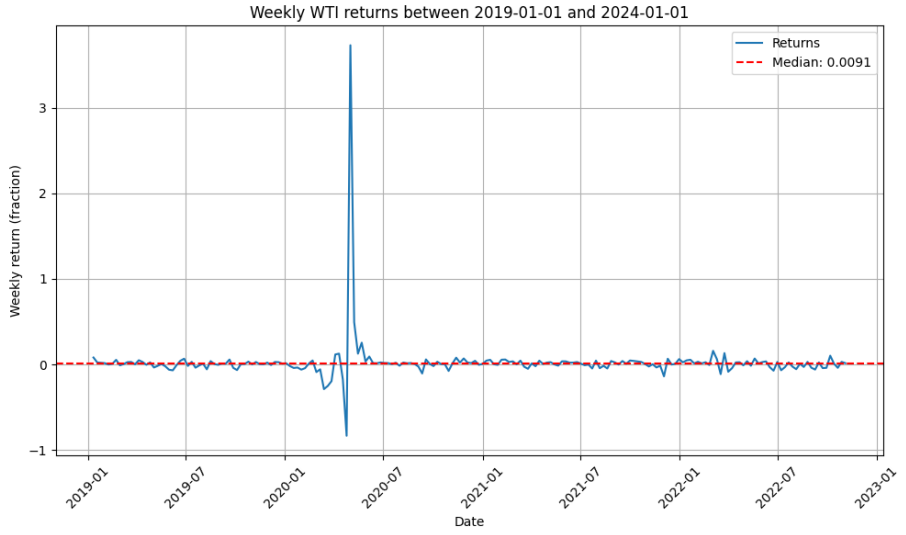

Figure 5.2: WTI weekly returns between 2019-01-01 and 2024-01-01

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This research explored the behavior of Time-Series Foundation Models (TSFMs), specifically Lag-Llama and TimeGPT-1, across various types of financial time-series, data frequencies, and context lengths. Through these experiments, we gained valuable insights into how these models perform on different types of data and identified key time-series characteristics that influence their performance. Additionally, we examined fine-tuning behavior by experimenting with different combinations of Lag-Llama fine-tuning parameters on a challenging time-series, characterized by extreme outliers. The model's tendency to predict near-median values in the face of such unpredictability led to the hypothesis that extreme outliers are responsible for this phenomenon.

Further analysis uncovered a fascinating pattern: time-series characteristics (stationary, no linear trend, no cyclicality) that cause fine-tuned Lag-Llama to perform poorly tend to improve the performance of its zero-shot version, and vice-versa. Interestingly, we have found that is is exactly the opposite time-series characteristics which make TimeGPT-1 perform better. Also we found that TimeGPT-1 displays opposite fine-tuning behavior, with the zero-shot version arguably outperforming the fine-tuned version on the same time-series characteristics beneficial to the fine-tuned version. This complementary nature of the these models suggests a potential benefit in leveraging both versions depending on the characteristics of the data. Finally, we discovered that both fine-tuned and zero-shot versions of Lag-Llama exhibited strong performance in terms of directional accuracy on many datasets.

In conclusion, fine-tuned Lag-Llama demonstrated excellent performance on financial time-series that are stationary and lack linear or cyclical patterns. On the other hand, TimeGPT-1 did not consistently perform well, excelling only in a small subset of experiments, such as with weekly CHAPS data.

6.2 Future Work

This research has opened up several avenues for future exploration. One direction is to expand the volume of experiments to include additional asset classes, such as options prices and stock trading volumes, and explore different frequencies (minutely, hourly, quarterly) across more diverse time periods. Additionally, future work could involve more detailed classification of time-series characteristics, including testing for quadratic, exponential, and damped trends, as well as distinguishing between additive and multiplicative seasonality. Further analysis could also focus on additional time-series features that were not tested in this research, such as volatility and error structures.

Another important direction is to optimize fine-tuning hyperparameters. A missing element in this research, due to computational limitations (Section 5.2.3), is an analysis of how to choose optimal fine-tuning hyperparameters based on the characteristics of the time-series. Expanding this analysis to include TimeGPT-1 fine-tuning optimization and exploring the use of exogenous variables would be valuable areas of research, particularly given the limitations of API calls in this study (Section 5.2.4).

Finally, an intriguing avenue for future research is the development of hybrid models. This approach could involve decomposing time-series into trend, cycle, and error components, using Lag-Llama to predict stationary components without trends or cyclical patterns, while employing a different model, such as zero-shot TimeGPT-1, to predict components with strong cyclical and trend components.

7. BIBLIOGRAPHY

REFERENCES

[1] A. F. Ansari, L. Stella, C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. S. Rangapuram, S. P. Arango, S. Kapoor, et al. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

[2] V. Assimakopoulos and K. Nikolopoulos. The theta model: a decomposition approach to forecasting. *International journal of forecasting*, 16(4):521–530, 2000.

[3] J. Ba. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] D. Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] G. Box and G. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day series in time series analysis and digital processing. Holden-Day, 1970.

[6] R. G. Brown. *Exponential smoothing for predicting demand*. Little, 1956.

[7] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[8] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[9] Z. Chen, L. N. Zheng, C. Lu, J. Yuan, and D. Zhu. Chatgpt informed graph neural network for stock movement prediction. *arXiv preprint arXiv:2306.03763*, 2023.

[10] K. Cho. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[11] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

[12] R. B. Cleveland, W. S. Cleveland, J. E. McRae, I. Terpenning, et al. Stl: A seasonal-trend decomposition. *J. off. Stat*, 6(1):3–73, 1990.

[13] M. Costantini, J. C. Cuaresma, and J. Hlouskova. Forecasting errors, directional accuracy and profitability of currency trading: The case of eur/usd exchange rate. *Journal of Forecasting*, 35(7):652–668, 2016.

[14] A. Das, W. Kong, R. Sen, and Y. Zhou. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688*, 2023.

[15] J. Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] D. A. Dickey and W. A. Fuller. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American statistical association*, 74(366a):427–431, 1979.

[17] J. Dong, H. Wu, Y. Wang, Y. Qiu, L. Zhang, J. Wang, and M. Long. Timesiam: A pre-training framework for siamese time-series modeling. *arXiv preprint arXiv:2402.02475*, 2024.

[18] A. Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[19] S. Gao, T. Koker, O. Queen, T. Hartvigsen, T. Tsiligkaridis, and M. Zitnik. Units: Building a unified time series model. *arXiv preprint arXiv:2403.00131*, 2024.

[20] A. Garza and M. Mergenthaler-Canseco. Timegpt-1. *arXiv preprint arXiv:2310.03589*, 2023.

[21] R. Gençay and M. Qi. Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. *IEEE transactions on neural networks*, 12(4):726–734, 2001.

[22] J. Hasanhodzic and A. W. Lo. Can hedge-fund returns be replicated?: The linear case. *The Linear Case (August 16, 2006)*, 2006.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[24] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[25] S. Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[26] Z. Hu, Y. Zhao, and M. Khushi. A survey of forex and stock price prediction using deep learning. *Applied System Innovation*, 4(1):9, 2021.

[27] R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for r. *Journal of statistical software*, 27:1–22, 2008.

[28] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[29] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of forecasting*, 18(3):439–454, 2002.

[30] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

[31] D. S. Kumar, B. Thiruvarangan, A. Vishnu, A. S. Devi, D. Kavitha, et al. Analysis and prediction of stock price using hybridization of sarima and xgboost. In *2022 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, pages 1–4. IEEE, 2022.

[32] Z. Lan. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[33] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.

[34] Y. Liang, H. Wen, Y. Nie, Y. Jiang, M. Jin, D. Song, S. Pan, and Q. Wen. Foundation models for time series analysis: A tutorial and survey. In *Proceedings of the 30th ACM*

SIGKDD Conference on Knowledge Discovery and Data Mining, pages 6555–6565, 2024.

[35] Y. Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[36] Y. Liu, H. Zhang, C. Li, X. Huang, J. Wang, and M. Long. Timer: Transformers for time series analysis at scale. *arXiv preprint arXiv:2402.02368*, 2024.

[37] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[38] S. Makridakis and M. Hibon. The m3-competition: results, conclusions and implications. *International journal of forecasting*, 16(4):451–476, 2000.

[39] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 38(4):1346–1364, 2022.

[40] F. J. Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[41] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[42] J. A. Miller, M. Aldosari, F. Saeed, N. H. Barna, S. Rana, I. B. Arpinar, and N. Liu. A survey of deep learning and foundation models for time series forecasting. *arXiv preprint arXiv:2401.13912*, 2024.

[43] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

[44] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from

natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[45] K. Rasul, A. Ashok, A. R. Williams, A. Khorasani, G. Adamopoulos, R. Bhagwatkar, M. Biloš, H. Ghonia, N. V. Hassen, A. Schneider, et al. Lag-llama: Towards foundation models for time series forecasting. *arXiv preprint arXiv:2310.08278*, 2023.

[46] A. Roberts, C. Raffel, K. Lee, M. Matena, N. Shazeer, P. J. Liu, S. Narang, W. Li, and Y. Zhou. Exploring the limits of transfer learning with a unified text-to-text transformer. *Google, Tech. Rep.*, 2019.

[47] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[48] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International journal of forecasting*, 36(3):1181–1191, 2020.

[49] V. Sanh. Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[50] N. Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[51] W. Shen, X. Guo, C. Wu, and D. Wu. Forecasting stock indices using radial basis function neural networks optimized by artificial fish swarm algorithm. *Knowledge-Based Systems*, 24(3):378–385, 2011.

[52] E. SLUTSKY. ön a case where the law of large numbers applies to mutually dependent quantities. the extension of a theorem by magoichirô watanabe. *Tohoku Mathematical Journal, First Series*, 28:26–32, 1927.

[53] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[54] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[55] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[56] I. Sutskever. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[57] I. Svetunkov, N. Kourentzes, and J. K. Ord. Complex exponential smoothing. *Naval Research Logistics (NRL)*, 69(8):1108–1123, 2022.

[58] P. Tang and X. Zhang. Mtsmae: Masked autoencoders for multivariate time-series forecasting. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 982–989. IEEE, 2022.

[59] S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[60] P. Temin. Price behavior in ancient babylon. *Explorations in Economic History*, 39(1):46–60, 2002.

[61] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[62] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[63] Q. Wen, J. Gao, X. Song, L. Sun, H. Xu, and S. Zhu. Robuststl: A robust seasonal-trend decomposition algorithm for long time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5409–5416, 2019.

[64] C. Wimmer and N. Rekabsaz. Leveraging vision-language models for granular market change prediction. *arXiv preprint arXiv:2301.10166*, 2023.

[65] H. Wold. *A study in the analysis of stationary time series*. PhD thesis, Almqvist & Wiksell, 1938.

[66] G. Woo, C. Liu, A. Kumar, C. Xiong, S. Savarese, and D. Sahoo. Unified training of universal time series forecasting transformers. *arXiv preprint arXiv:2402.02592*, 2024.

[67] Q. Xie, W. Han, Y. Lai, M. Peng, and J. Huang. The wall street neophyte: A zero-shot analysis of chatgpt over multimodal stock movement prediction challenges. *arXiv preprint arXiv:2304.05351*, 2023.

[68] L. Xue. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*, 2020.

[69] X. Yu, Z. Chen, Y. Ling, S. Dong, Z. Liu, and Y. Lu. Temporal data meets llm–explainable financial time series forecasting. *arXiv preprint arXiv:2306.11025*, 2023.

[70] G. U. Yule. On the time-correlation problem, with especial reference to the variate-difference correlation method. *Journal of the Royal Statistical Society*, 84(4):497–537, 1921.

[71] G. U. Yule. Vii. on a method of investigating periodicities disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 226(636-646):267–298, 1927.

[72] B. Zhang and R. Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

[73] C. Zhang and X. Zhou. Forecasting credit risk of smes in supply chain finance using bayesian optimization and xgboost. *Mathematical Problems in Engineering*, 2023(1):5609996, 2023.

[74] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.

[75] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International conference on machine learning*, pages 27268–27286. PMLR, 2022.

A. HISTORY OF TIME-SERIES FORECASTING

A.1 Brief History of time-series prediction in Finance

When the agricultural revolution took place (circa 10,000 BC), for the first time in history, humans have started producing more food and items than they required just to survive. This gave birth to the concept of "storing" things and eventually trading them for other things which were perceived to be of greater value. When sufficient amount of people started transacting goods in this manner, the concept of "market price" took hold. It did not take long before humans realized that they didn't have to work in the field the whole day to amass wealth, but that they could do it through trade alone. They could buy items from one person at a low price and sell to another at a high price (arbitrage) or buy it now at a low price and hopefully sell later at a high price (asset appreciation). Opportunities to exploit the first option became more rare and less profitable as more and more people started engaging in trade and eliminating these arbitrage opportunities. However, the 2nd option has captured the imagination of traders ever since 10,000BC as no one seemed to be able to tell with certainty whether the price of a good would indeed go up in the future or not. As humans learned to write and keep records, they starter noting down the prices of goods that were being traded along with the time when those prices were prevailing. We have records of ancient Babylonians keping historic records of enf-of-month barley, dates, cuscuta, sesame, cardamom and wool prices on clay tablets spanning some 400 years [22]. Investigating these prices, we can see that they follow a random walk pattern with exogenous shocks (such as the death of Alexander the Great) [60] and they seem almost impossible to predict with a naked eye, however, it is not impossible that someone might have tried and unknowingly became the first person in history to work on time-series prediction in the field of Finance.

Over the centuries, methods to solve the problem of time-series prediction in finance seem to have developed universally across many different cultures. Christopher Kurz, a 16th century merchant from Antwerp, thought about the idea of "trend" and "seasonality" in agricultural prices and claimed he could predict prices 20 days in advance. In 18th century Japan, Munehisa Honma - a man dubbed "God of the markets" by his contemporaries, developed a principle stating that when prices become extremely high, they must come

down again which we know today to be the principle of "mean reversion". In the late 19th century, Charles Dow, co-founder of Wall Street Journal and Dow Jones Company[32] popularized and coined the term "technical analysis" (a collection of methods that employ math and statistics to predict future prices) which later evolved into quantitative analysis - which is used today in institutions engaged in future price prediction.

A.2 Brief History of modern time-series prediction methods

Statistical models

Although the field of future price prediction has moved far beyond simply considering just the historical prices[33], there has been notable work on the general field of univariate[34] time-series prediction. It is hard to pinpoint the exact start of modern time-series prediction, but I think a good candidate would be Udny Yule's[35] 1927 [71] paper which is considered first to have used simple autoregressor model (AR)[36] to predict Wolfer's Sunspot numbers[37]. Even though Slutsky (1927) [52] and Udny (1921) [70] demonstrated the use of Moving Average as a way to show trend and cyclicality in time-series data, Herman Wold's work (1938)[38] [65] indirectly invented the Auto Regressive Moving Average (ARMA) model[39]. ARMA model is theoretically sound if the time-series being predicted is stationary[40], which is not the case with all time-series. In 1970, George Box and Gwilym Jenkins introduced probably the most famous time-series prediction model of all times: ARIMA [5]. ARIMA(p, i, q) model is an ARMA(p, q) model that employs the method of

[32] Dow Jones Company is the publisher of the prominent DOWJ stock market index

[33] Quantitative analysis now-adays considers wide palette of data and information

[34] Univariate time-series prediction is a practice of using only the past values of a certain time-series in order to predict its future value(s).

[35] Udny studied at UCL under a well-known (and controversial) professor Karl Pearson - father of mathematical statistics and generally one of the most well-known personas in statistics.

[36] An AR(p) time-series prediction model predicts future values based on a linear combination of the previous p observations in the series, assuming that the current value depends on its own past values and a stochastic error term.

[37] Certain time-series data from the field of Astronomy.

[38] Wold's Decomposition Theorem states that any stationary time series can be decomposed into a deterministic part (AR component) and a stochastic part (MA component)

[39] The MA(q) part of the ARMA(p, q) model forecasts future values by modeling the current value as a linear combination of the past q error terms (shocks) and a stochastic error term.

[40] A time series is said to be stationary if its statistical properties, such as mean, variance, and autocorrelation, remain constant over time

differencing[41] on the date before training and prediction.

Another direction of innovation in the field of univariate time-series prediction was towards "Exponential Smoothing" (ES). Concept of ES originates from the work of Brown 1956 [6] and has been thoroughly built upon since. Most notable invention was by Hyndman et al. in 2002 [29] - implementing exponential smoothing within a state-space framework, giving us the ETS[42] model. Most recent culmination in the ES family of models was by Ivan Svetunkov in 2022 [57] with the CES model[43]. Final honorable mention among the time-series prediction models of "statistical" nature is the "Theta" model. Vassilis Assimakopoulos, and Konstantinos Nikolopoulos (2000) [2] Introduced the Theta model which decomposes a time-series into its' cyclical and trend components which are then forecasted separately and finally combined[44].

Machine Learning (ML) models

As the hardware technology advanced in the 21st century, so did the time-series models as the scientists have finally gotten machines with enough computational power to run more complex algorithms. The most famous of these algorithms being the Artificial Neural Network (ANN). Although the idea (in its primitive form[45]) dates back all the way to McCulloch and Pitts (1943) [41], ANN first real ancestor was the "Perceptron"[46], invented by a psychologist - Frank Rosenblatt in 1958 [47]. The perceptron had ability to take continuous inputs and proper learning algorithm - known even to this day as the "Rosenblatt algorithm". Through the rest of the 20th century, there have been general

[41]Differencing is a technique in time-series analysis used to transform a non-stationary series into a stationary one by deducting the preceding observation from each observation. The order "i" of the differencing refers to the ammount of times this procedure is applied to a time-series.

[42]ETS stands for Error Trend Seasonality. ETS is a family of 18 models. ETS model can account for errors being additive or multiplicative, and trend or seasonality being none, additive or multiplicative. ETS models with no multiplicative errors or seasonality have their equivalent withing the ARIMA family of models

[43]CES stands for Complex Exponential Smoothing model. CES method models the time-series as a series of complex numbers where the real part is the prediction and imaginary parts are errors. CES models can also be expressed in state-space form to adress seasonality. CES advantages are that it is flexible and has been empirically shown to perform well.

[44]These components are called "theta lines". Theta model can use exponential smoothing or an AR model to predict these two lines. Predictions are finally combined using weighted average where weights could be equal or adjusted to favor either the trend line or the cycle line.

[45]The original vision of an ANN was based on simple binary inputs with fixed thresholds activation - a far cry from modern ANNs

[46]Perceptron can be thought of as a single-neuron ANN. ANNs are comprised of one or more neuron layers, each layer containing one or more neurons.

periods of great interest and disinterest in ANNs for many reasons, but ANNs finally came on their own in the 21st century when computers become powerful enough to be able to train larger versions of these models[47] in reasonable time[48]. Naturally it was not long before people realized ANN-type models can also be used for time-series prediction[49] which led to many variations of the ANN architecture to be invented and used in financial time-series forecasting, such as: Bayesian regularization ANN [21] and Radial Basis Function ANN (RBFANN) [51]. Among non-ANN - based ML models, the most famous is XGboost [8] which also saw some use in Finance [31] [73].

[47]Size of an ANN matters when dealing with very complex tasks (tasks with many input features).

[48]Another reason for explosion in popularity was the beginning of the internet era and sudden availibility of large ammounts of data which are needed for ANN training

[49]It is important to note that univariate time-series prediction ANNs are simply regression ANNs where lags of the data we're trying to predict are explanatory features (inputs) to the model

B. BACKGROUND

B.1 The transformer

Token

A token is a multi-dimensional numeric vector representation of an element in the sequence. Reason why sequence of tokens is used as input to a sequence-to-sequence model is because ML models can only "understand" numbers - therefore they can't work with some type of sequences (eg. human language sentences). Reason why tokens are vectors rather than simple nubers is because they are designed to represent the semantic meaning of a real-world element they represent and the semantic meaning of a real-world element could change depending on the context. Therefore different dimensions of a token account for different meaning of that element in different contexts.

LSTM and GRU Hidden state

The hidden state is an intermittent sequence within a sequence-to-sequence model. It is a product of applying the encoder on the input sequence. The output sequence is generated by applying the decoder to the hidden state. Having a hidden state of fixed size means that some information will inherently get lost when a very long input sequence is fed into the model.

C. LAG-LLAMA

C.1 LLaMA

RMSnorm is a layer normalization technique. Zhang and Sennrich (2019) demonstrate that centering component of the LayerNorm technique is dispensable (and computationally expensive) and propose their own layer normalization strategy called RMSnorm which delivers similar benefits (more stable training and better model convergence) but with lower computational cost.

RoPE stands for Rotary Positional Encoding. This is a method for positional encoding. RoPE enables valuable properties, including the flexibility of sequence length, decaying inter-token dependency with increasing relative distances, and the capability of equipping linear self-attention with relative position encoding.

SwiGLU activation function is a combination of swish and GLU activation functions. With beta hyperparameter equal to 1, it is of similar shape as ReLU but completely smooth (continuous) - therefore "behaving better" during model training.

D. METHODOLOGY

D.1 Time-series prediction

MDA

MDA is time-series specific. Since time-series data is inherently ordered (as opposed to regular regression data), we can measure the direction in which a time-series is moving at each time-step. Hence we can compare the direction of movement of our predicted time-series against the actual time-series and we can calculate in what percentage of cases did the two time-series move in the same direction (up or down).

MES

Reason why MDA doesn't fully capture the model's ability to predict direction of underlying asset value movement is because the models are working with returns time-series, therefore MDA measures whether the model accurately predicts the directional change of returns and not the actual direction of movement of the underlying values. Imagine scenario where $y_{actual}{}^{T} = 1\%$, $y_{actual}{}^{T+1} = 0.5\%$ and $y_{predicted}{}^{T} = 0.7\%$, $y_{predicted}{}^{T+1} = -0.5\%$. In this case MDA would account this as a correct directional guess as the model predicted y would go down in period T+1, and y actually did go down at T+1. However, if we inspect this case in greater detail, at the moment T+1, the value of the underlying asset which y represents, went up at T+1, as the return in that period is still positive, however the model actually implicitly predicted the price to go down. This is a nuance which MDA doesn't capture, but MES does. This metric makes sense if the values of the underlying time-series are centered around 0 (mean = 0), which returns in financial markets are in the short-term. However, this metric can be generalized to for time-series of any mean *mu* if we define it as MES = mean$\{\text{sign}(y_{predicted}{}^{1} \text{ x } (y_{actual}{}^{1}\text{-}mu))$, $\text{sign}(y_{predicted}{}^{2} \text{ x } (y_{actual}{}^{2}\text{-}mu))$, ..., $\text{sign}(y_{predicted}{}^{n} \text{ x } (y_{actual}{}^{n}\text{-}mu))\}$

D.2 Data

CHAPS data

CHAPS dataset used in this research is: "ukspendingoncreditanddebitcards" dataset at https://www.ons.gov.uk/economy/economicoutputandproductivity/output/datasets/

Data sourcing

Exceptions:

- Daily CHAPS data not available 2018-01-01 to 2020-01-01.
- Weekly CHAPS data not available 2017-01-01 to 2022-01-01 and 2015-01-01 to 2020-01-01.
- Weekly CHAPS data used was available only 2020-01-01 to 2024-01-01.
- Monthly CHAPS and Exchange rate data was not available in sufficient quantity to conduct experiment.

Partial autocorrelation

Partial autocorrelation measures the correlation between a time series and its lagged values, while controlling for the influence of intermediate lags. In contrast, autocorrelation simply measures the correlation between the time series and its lagged versions without accounting for the effects of other lags. Partial autocorrelation provides a clearer picture of the direct relationship between observations and their lagged counterparts by isolating the specific influence of each lag. This makes it particularly useful for detecting seasonality in time-series data because it helps identify the direct impact of observations at seasonal intervals. By highlighting the most relevant lags for seasonality, partial autocorrelation allows for a more precise analysis of recurring patterns, which is harder to achieve using autocorrelation alone.

E. GRANULAR TIME-SERIES CHARACTERISTICS RESULTS

cyclical_patterns	trend	model	r2	mse	mae	rmse	mape	mda	mes
FALSE	linear	arima	1.00	1.00	1.00	1.00	1.33	7.00	1.00
		autoregressor	7.00	7.00	7.00	7.00	7.00	5.67	5.33
		ft_lag_llama	5.00	5.00	5.33	5.00	3.00	3.67	2.00
		ft_timeGPT	3.33	3.33	2.67	3.33	3.67	2.67	3.00
		lag_llama	2.67	2.67	3.33	2.67	4.00	2.67	5.33
		prophet	4.67	4.67	3.67	4.67	3.33	1.33	3.67
		timeGPT	4.33	4.33	5.00	4.33	5.67	4.00	5.33
	no_trend	arima	2.56	2.56	2.61	2.56	1.99	5.83	1.37
		autoregressor	6.52	6.52	6.57	6.52	6.38	4.09	4.80
		ft_lag_llama	2.33	2.33	2.31	2.33	2.23	2.21	2.40
		ft_timeGPT	3.78	3.78	4.00	3.78	3.85	3.90	5.11
		lag_llama	4.46	4.46	4.39	4.46	4.64	2.55	3.52
		prophet	4.05	4.05	4.05	4.05	4.41	3.20	4.71
		timeGPT	4.30	4.30	4.07	4.30	4.49	4.55	4.17
TRUE	linear	arima	2.76	2.76	2.62	2.76	2.62	2.69	1.00
		autoregressor	1.88	1.88	1.52	1.88	1.50	2.50	1.00
		ft_lag_llama	6.26	6.26	6.26	6.26	6.31	4.76	1.00
		ft_timeGPT	2.88	2.88	3.07	2.88	3.07	3.71	1.00
		lag_llama	5.57	5.57	5.71	5.57	5.64	4.88	1.00
		prophet	5.93	5.93	5.79	5.93	5.83	4.17	3.57
		timeGPT	2.71	2.71	3.02	2.71	3.02	2.86	1.00
	no_trend	arima	3.17	3.17	3.06	3.17	2.89	5.94	1.19
		autoregressor	6.69	6.69	6.56	6.69	5.86	5.11	4.47
		ft_lag_llama	2.39	2.39	2.78	2.39	3.33	2.44	1.83
		ft_timeGPT	3.86	3.86	3.50	3.86	3.86	2.72	4.58
		lag_llama	3.61	3.61	3.81	3.61	3.64	2.61	3.58
		prophet	3.81	3.81	4.25	3.81	4.78	2.81	4.86
		timeGPT	4.47	4.47	4.06	4.47	3.64	5.17	4.72

Table E.1: Breakdown of results by cyclicality and trend

stationary	cyclical_patterns	model	r2	mse	mae	rmse	mape	mda	mes
FALSE	TRUE	arima	3.19	3.19	3.00	3.19	3.00	3.38	1.00
		autoregressor	2.24	2.24	1.76	2.24	1.76	2.95	1.00
		ft_lag_llama	6.57	6.57	6.57	6.57	6.57	3.90	1.00
		ft_timeGPT	2.48	2.48	2.71	2.48	2.71	3.71	1.00
		lag_llama	5.81	5.81	5.86	5.81	5.86	4.19	1.00
		prophet	5.29	5.29	5.24	5.29	5.24	4.29	2.71
		timeGPT	2.43	2.43	2.86	2.43	2.86	3.05	1.00
TRUE	FALSE	arima	2.51	2.51	2.56	2.51	1.97	5.87	1.36
		autoregressor	6.53	6.53	6.59	6.53	6.40	4.14	4.82
		ft_lag_llama	2.42	2.42	2.41	2.42	2.26	2.26	2.39
		ft_timeGPT	3.77	3.77	3.96	3.77	3.84	3.86	5.04
		lag_llama	4.40	4.40	4.36	4.40	4.62	2.56	3.58
		prophet	4.07	4.07	4.03	4.07	4.38	3.13	4.68
		timeGPT	4.30	4.30	4.10	4.30	4.53	4.53	4.21
	TRUE	arima	2.86	2.86	2.75	2.86	2.65	4.49	1.12
		autoregressor	4.79	4.79	4.61	4.79	4.16	3.98	3.19
		ft_lag_llama	3.70	3.70	3.95	3.70	4.33	3.61	1.53
		ft_timeGPT	3.65	3.65	3.47	3.65	3.70	3.09	3.26
		lag_llama	4.25	4.25	4.46	4.25	4.30	3.70	2.63
		prophet	4.82	4.82	5.02	4.82	5.39	3.26	4.70
		timeGPT	3.93	3.93	3.74	3.93	3.47	4.25	3.35

Table E.2: Breakdown of results by stationarity and cyclicality

stationary	trend	model	r2	mse	mae	rmse	mape	mda	mes
FALSE	linear	arima	3.19	3.19	3.00	3.19	3.00	3.38	1.00
		autoregressor	2.24	2.24	1.76	2.24	1.76	2.95	1.00
		ft_lag_llama	6.57	6.57	6.57	6.57	6.57	3.90	1.00
		ft_timeGPT	2.48	2.48	2.71	2.48	2.71	3.71	1.00
		lag_llama	5.81	5.81	5.86	5.81	5.86	4.19	1.00
		prophet	5.29	5.29	5.24	5.29	5.24	4.29	2.71
		timeGPT	2.43	2.43	2.86	2.43	2.86	3.05	1.00
TRUE	linear	arima	2.17	2.17	2.08	2.17	2.13	2.63	1.00
		autoregressor	2.21	2.21	2.00	2.21	1.96	2.50	1.54
		ft_lag_llama	5.83	5.83	5.88	5.83	5.67	5.38	1.13
		ft_timeGPT	3.29	3.29	3.33	3.29	3.46	3.58	1.25
		lag_llama	5.00	5.00	5.29	5.00	5.25	5.21	1.54
		prophet	6.33	6.33	6.00	6.33	6.04	3.71	4.33
		timeGPT	3.17	3.17	3.42	3.17	3.50	2.83	1.54
	no_trend	arima	2.74	2.74	2.74	2.74	2.25	5.86	1.32
		autoregressor	6.57	6.57	6.57	6.57	6.23	4.39	4.71
		ft_lag_llama	2.35	2.35	2.45	2.35	2.55	2.28	2.24
		ft_timeGPT	3.80	3.80	3.85	3.80	3.85	3.55	4.96
		lag_llama	4.21	4.21	4.22	4.21	4.35	2.57	3.54
		prophet	3.98	3.98	4.11	3.98	4.52	3.08	4.76
		timeGPT	4.35	4.35	4.07	4.35	4.24	4.73	4.33

Table E.3: Breakdown of results by stationarity and trend