Overall more complex design than Costa et al, but git with code is provided (but not the fine tuning dataset)

Financial Fine-tuning a Large Time Series Model

Xinghong Fu
Massachusetts Institute of Technology
Cambridge, MA, United States
fxh@mit.edu

Masanori Hirano
Preferred Networks, Inc.
Tokyo, Japan
research@mhirano.jp

Kentaro Imajo
Preferred Networks, Inc.
Tokyo, Japan
imos@preferred.jp

***Abstract*—Large models have shown unprecedented capabilities in natural language processing, image generation, and most recently, time series forecasting. This leads us to ask the question: treating market prices as a time series, can large models be used to predict the market? In this paper, we answer this by evaluating the performance of the latest time series foundation model TimesFM on price prediction. We find that due to the irregular nature of price data, directly applying TimesFM gives unsatisfactory results and propose to fine-tune TimeFM on financial data for the task of price prediction. This is done by continual pre-training of the latest time series foundation model TimesFM on price data containing 100 million time points, spanning a range of financial instruments spanning hourly and daily granularities. The fine-tuned model demonstrates higher price prediction accuracy than the baseline model. We conduct mock trading for our model in various financial markets and show that it outperforms various benchmarks in terms of returns, sharpe ratio, max drawdown and trading cost.**

***Index Terms*—quantitative finance, deep learning, foundation models, time-series forecasting, price prediction**

I. Introduction

Predicting the market has long been of interest to researchers. Under the more general task of time-series forecasting, countless research attempts date back to simple moving averages [1], various types of models has been developed, including autoregressive [2], global univariate models like N-BEATS [3], and long-term forecasting models [4].

Following the development of large language models [5]–[7], researchers have also attempted to directly make use of the zero-shot forecasting capabilities of LLMs [8]–[12]. Benefits of using pre-trained LLMs include the availability of text context together with numeric data to improve forecasting prediction accuracy [11], and access to a strong encoder/decoder such that tuning the LLM outputs to time-series forecasting only requires aligning the embedding layer for numeric time series data [9], [12]. However, recent work [13] raises question on the usefulness of the LLM backbone by comparing it to basic attention layers trained from scratch. A similar concern turns our attention to TimesFM [14], a foundation time-series model trained from scratch, specifically for the task of time-series forecasting. Detailed more in Section II, TimesFM achieves state-of-the-art performance of multiple forecasting benchmarks. However, these benchmarks most oftenly include regular and seasonal data, much unlike financial data that we are interested in. In this work, we answer the following research question: **can a foundation time series model perform well on price data in the financial markets?**

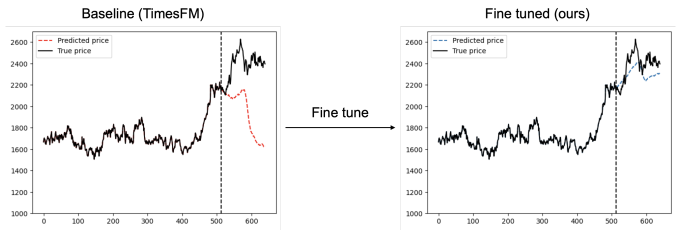

Fig. 1. We show that the baseline foundation time-series model TimesFM fails at the task of financial market price prediction. Fine-tuning TimesFM on financial data siginificantly improves its prediction accuracy.

In order to address this research question, we begin by evaluating the performance of TimesFM on the task of price prediction. We find the baseline TimesFM shows extremely undesirable performance. In particular, for the price trajectory prediction task shown in Figure 1, baseline TimesFM fails drastically. Nonetheless, we show that fine-tuning TimesFM through continual pre-training allows significant improvement over multiple standard benchmarks across several markets. We conduct a mock trading experiment with our fine-tuned TimesFM against multiple standard benchmarks to demonstrate its performance in the following markets: S&P500 stocks, TOPIX500 stocks, forex market, cryptocurrencies.

We list our main contributions in this paper below:

1) Dataset curation for fine-tuning on price data
2) Fine-tuning TimesFM to give a **financial times series foundation model**
3) Modification of training methods (loss and masking) to stabilize training on financial time series, specifically price data
4) Testing of fine-tuned model by mock trading experiments to demonstrate that our fine-tuned model outperforms multiple standard benchmarks across a variety
5) The code and model weights are published at https://github.com/pfnet-research/timesfm_fin for reproducibility of results and to accelerate further research

The following paper discusses (in order): *related work* (section II) of transformers to time series predictions and a description of TimesFM, *fine-tuning method* (section III) including our modifications, details about our *experiments* (section IV) including training dataset, hyperparameter settings, evaluation metrics and mock trading. We follow this by the next section on *results* (section V), showing how fine-tuning TimesFM improves its performance over standard baseline a range of experiments and financial markets. We finally suggest some future work possibilities in *discussions* (section VI).

II. Related Work

"Buy low, sell high" - one of the fundamental principles of the market that traders rely on to profit from their trades. But how do we know, what is 'low' and how much is 'high'? In essence, the problem reduces to accurate prediction of the price of a financial asset.

Historically, quantitatively modelling prices have founded upon methods such as autoregressive [2] combined with moving averages [1] and conditional variances [15], Kalman filtering [16] and hidden Markov models [17] to name a few. Much of these models rely on well calibrated mathematical concepts to model the underlying dynamics of the market and fit a trend for the prices. Advancements in neural network architectures such as the RNN [18] and LSTM [19] have inspired price analysis [20]. More recently, increased compute capabilities have shown that with large amounts of data and a sufficient model capacity to absorb the data, models can be trained to capture underlying trends and generalize as well, or even better than previous models. [21] A particular architecture which made this possible was the transformer.

Transformers, first introduced in [5], revolutionized natural language processing through large language models [6], [7], [22], [23], and computer vision fields of image classification [24], video classification [25] and image generation [26]. For our task of financial market price prediction, we examine some related works of utilizing transformers in time series predictions. [4], [8], [9], [11], [14], [27]

Industry standard of using transformers for time series forecasting have made use mainly of LLMs [8], [9], [11], [27], [28]. Recent works [4], [10], [13] question the necessity and relevance of LLMs in forecasting, and instead focus on training a time series foundation model with transformer building blocks specifically for the task of time series forecasting [14], to which we turn our attention.

TimesFM [14] is a 200-million parameter decoder-only model trained a time series datasets, containing 100-billion time points, with the task of next-value forecasting. We refer readers to the original paper [14], but we provide a brief summary of this related work, including its methods and main findings in section III.

Evaluation of TimesFM done on the Darts [29], Monash [30] and Informer [31] benchmarks demonstrate strong performance of TimesFM in terms of the mean-average-error(MAE) metric in comparison to many previous SOTA methods, including N-BEATS [3], LLM-Time [8], ARIMA [1] and PatchTST [4].

Our continual pre-training recipe is mostly aligned to that in TimesFM. More details are provided in Section III

III. Financial Fine-tuning method

We first describe TimesFM, the existing model on which we base our fine-tuning. In TimesFM, input time series data is patched into $input\_patch\_len = l_i$ length patches, these are processed by stacked transformer layers from which an output block containing $output\_patch\_len = l_o$ time points are predicted. Mean-squared error (MSE) loss is computed on these l_o points:

$$Train\_Loss = \frac{1}{N}\sum_{j=1}^{N} MSE(\hat{y}_{l_i j+1:l_i j+l_o}, y_{l_i j+1:l_i j+l_o}) \tag{1}$$

The authors typically set $l_i = 32$ and $l_o = 128$ and recommends $l_o > l_i$ to train the model in a decoder-only mode and also minimizes the number of autoregressive steps needed at inference time. Random masking is also applied to train the model going through all possible context lengths.

During inference time, the model reads in the l_o points it generates as input and repeatedly generates new time points until all have been autoregressively generated. Masking is not applied at inference time.

Data used during pre-training of TimesFM mainly included Google trends, Wiki page views, and many other publicly available time series data sources. The authors also showed that a mix of synthetic data improved the performance of the model.

In the remaining of this section, we introduce modifications to the original TimesFM for fine-tuning on financial data (specifically, price data). The method we employ is continual pre-training: restarting training from the pre-trained weights of TimesFM, continuing stochastic gradient descent on financial data. We restart training with a linear warmup to a learning rate of 5e-4 followed by a cosine decay. The specific training recipe is listed in Table II in section IV. Model architecture follows from the publicly available TimesFM checkpoint. We list two of our contributions for adapting TimesFM for continual pre-training on financial data.

A. Loss

The original MSE loss (Equation 1) comes with its set of pitfalls when trained on price data:

1) Biases towards large scale values, e.g. a stock index with average values of USD1000 will receive much more weight in training than cryptocurrencies averaging BUSD0.0001
2) Instability due to market crash events. Especially when high price stocks experience a rapid crash of more than 99% of its original value, instability in a single step results in NaN loss and failure of convergence.

In this section, we tackle these problems by describing a small modification to the loss computation. Namely, we apply a log transformation to the original time series, make predictions based on these transformed sequences. The MSE loss is then computed on these log-ed sequences. What we do explicitly is

$$z \leftarrow \log(y) \tag{2}$$

Binance is for cryptos

where z is used as the input to the model, then followed by

$$Train\_Loss = \frac{1}{N}\sum_{j=1}^{N} MSE(\hat{z}_{l_i j+1:l_i j+l_o}, z_{l_i j+1:l_i j+l_o}) \tag{3}$$

For small changes in y, computing the MSE of $z = \log(y)$ is equivalent to computing the percentage MSE loss. But for large changes in y, the tapering of the $\log$ function results in a less than proportionate change in z, which in turn stabilizes training.

B. Masking

We employ a similar masking strategy to that described in [14], where we want to randomly sample the start and end points of the time series. This is done with the following method:

For training efficiency, time series are broken up into sequences of length of at most $max\_context\_length + output\_length$. We then randomly sample a random t_{end} from $[min\_context\_length, max\_context\_length]$ then sample a random t_{start} from $[0, t_{end} - min\_context\_length]$. The points between $[t_{start}, t_{end}]$ are then taken as input, where the model outputs the next $output\_len$ many points during training and loss is evaluated on those points.

Typically, we set $min\_context\_len = 128$ to ensure that the model is trained on meaningful (sufficiently long) examples. Our masking strategy fine tunes TimesFM to be able to predict any price data sequence of length from $min\_context\_length$ to $max\_context\_length$. These random masks change between batches and training steps, preventing overfitting by training the model to forecast from a variety of segments of the time series.

Through the strategies describe in this section, we are able to complete fine tuning of TimesFM on 80M time points within 1 hour without any NaN loss.

IV. Experiments

In this section, we build upon our method described in section III and set up computational experiments to address our original research question: **can a foundation time series model perform well on price data in the financial markets?** We first begin by detailing the data and settings used to run our experiments: to build a fine-tuned TimesFM and compare it on several experiments against previous benchmarks (including the original TimesFM). These experiments, later explained more thoroughly, includes comparing the accuracy and F1-score of price prediction across various prediction horizons. From a financial perspective, we wish to understand the profitability of the model beyond evaluation metrics such as accuracy and F1-score which do not capture intricacies such as magnitudes of price movements, cost of trading among other considerations when deployed in the market. To that end, we propose a mock trading set up, devising an executable trading strategy based on our fine-tuned model, to compare foundation time series models (original and fine-tuned TimesFM) to a by-chance model and an AR1 model.

A. Data

Consisting of price time series in stocks, indices, foreign currencies and crypto currencies, data span granularities of hourly and daily. Main source used include Yahoo Finance and Binance, from which data is obtained using publicly available API endpoints. A detailed description of thecontinual pre-training data used can be found in Table I. Our dataset totals more than 100K time series and 90M time points.

To avoid look-ahead bias, data from year 2023 onwards is reserved for testing. During the training process, we use a 75-25 split between train and validation dataset, randomly sampled from the same subset of time series ending before 1 Jan 2023.

TABLE I
Summary of data used for fine-tuning.

Dataset	**Granularity**	**# Time Series**	**# Time Points**
Topix500 stocks	Daily	3513	2,248,320
S&P500 stocks	Daily	3173	2,030,720
Currencies	Daily	1092	698,880
Japan Investment Trusts	Daily	6698	4,286,720
Commodities	Daily	29	18,560
Stock Indices	Daily	216	138,240
Stock Indices	Hourly	847	542,080
Stock prices	Hourly	31,756	20,323,840
Cryptocurrencies	Daily	1680	1,075,200
Cryptocurrencies	Hourly	79,153	50,657,920

As opposed to the original TimesFM, we do not use any synthetic data in training and do not conduct any reweighting to sample each granularity evenly. We acknowledge the possibilities of future work in this area where some suggestions are offered in Section VI. Nonetheless, we observe that while the training process included more of hourly granularity data, the model shows better performance over longer prediction horizons, as demonstrated in Section V.

B. Hyperparameters

Table II list out the settings used for fine-tuning TimesFM. Notably, we use SGD with linear warmup and cosine decay with a peak learning rate of 5e-4.

TABLE II
Hyperparameters and architecture settings.

Hyperparameter/Architecture	**Setting**
Optimizer	SGD
Linear warmup epochs	25
Total epochs	100
Peak learning rate	5e-4
Momentum	0.9
Gradient clip (max norm)	1.0
Batch size	1024
Max context length	512
Min context length	128
Input length	32
Output length	128
Layers	20
Hidden dimensions	1280

Following the training recipe listed out in Table II and using the data in I, we are able to complete training on 8 V100s within 1 hour without any NaN loss. Training curves are shown in Figure 2.

C. Testing

To explore whether fine-tuning TimesFM does indeed lead to a performance gain when deployed in financial market, we run several experiments detailed in this section. The first metric we compare is the accuracy of price predictions, over various prediction horizons (equivalent to holding period of the asset). This is done on the test set (data from 2023 onwards, not used in training and validation). We also introduce a more robust metric: Macro F1-score, allowing more fair comparisons of the models even under class-imbalanced situations. Lastly, we conduct mock trading in various markets: S&P500 stocks, TOPIX500 stocks, currencies, cryptocurrencies, to verify that performance in accuracy and macro F1 is translated into Profit and Loss (PnL).

1) Metric: Accuracy: Recall that at training time, the model is given $input\_length <= max\_context\_length = 512$ data points (with random masking) and tasked to always predict the next $output\_len = 128$ many points. Loss is evaluated on these $output\_len$ many points.

At inference time, the model is consistently given $context\_length = c$ many points (without masking, where $c <= 512$) and tasked to predict the following points. However, we might wish to generate an arbitrary number of future points, not necessarily 128.

At each step, the model predicts the next h many points. For this single step, accuracy is evaluated on the last output point y_{c+h}, where the model is tasked to classify whether the price moves up or down. At the next step, the model then reads in the next ground truth h points, and computes the accuracy again with $\hat{y}_{c+2h}$. Accuracy calculation is based on this classification over every inference step, i.e.

$$Accuracy(\hat{y}_{c+kh}, y_{c+kh} | y_{1:c+(k-1)h}) \tag{4}$$

for all $1 \leq k \leq K$ such that Kh is longer than the desired total prediction horizon H. In our experiments, we fix $H = 128$ and vary $h \in \{2, 4, 8, \ldots, 128\}$. We note that the model is only capable of processing the last $max\_context\_length = 512$ points in $y_{1:c+(k-1)h}$.

Results comparing our fine-tuned TimesFM against the baseline TimesFM are shown in Figure 3.

2) Metric: F1 score: Accuracy scores may not always be a reasonable metric of interest. As an example, on a biased dataset with 90% positive samples and 10% negative. a model classifying every sample as positive will attain a 90% accuracy. Nonetheless, we should ask the question: did the model truly learn the underlying distribution.

An alternative to accuracy is the F1-score, taken as the harmonic mean of precision and recall. However, the F1-score also suffers from problems with class imbalance [32], which points us to adopt the Macro F1-score [33], which is computed as the arithmetic mean of F1-scores taking each class as the 'positive' class. Specifically, in the example above we obtain a Macro F1-score of 0.474, showing that the model fails to learn the 10% negative rate well.

3) Mock trading: Here, we develop trading strategies based on fine-tuned TimesFM and analyze our profits from our trades.

We outline below the first trading strategy, which we shall term *basic strategy*. The trader begins by selecting a holding period, denoted as h (where $h =$ horizon_len). Additionally, we define the context length as c (where $c = 512$, using maximum context length throughout). h and c are analogous to that in Equation 4, since the trades are made based on the prediction h-steps ahead.

After trading day i, the trader inputs the time series $P_{i-c-1:i} = \{P_{i-c-1}, P_{i-c}, \ldots, P_i\}$ into a model to obtain a prediction for $P_{i+1:i+h}$. The trader places a buy or sell order on day $i+1$ and $i+h$ based on the following conditions:

- If $P_{i+h} > P_{i+1}$, place a buy order on day $i+1$ and a sell order on day $i+h$.
- If $P_{i+h} < P_{i+1}$, place a sell order on day $i+1$ and a buy order on day $i+h$.

This strategy is repeated for all trading days i.

If the trading basket contains a total of T assets, all orders placed will be worth $\frac{1}{(h-1)T}$ each. This ensures that the ℓ_1-norm of the orders placed on a given day does not exceed $\frac{1}{h-1}$, and over the entire holding period does not exceed 1 (which represents the total capital). We limit the norm of our orders to ensure that, even in the extreme case where every order during the holding period is a 'buy', sufficient capital is available to place all orders. As an example, our initial budget is \$1, and the holding period $h = 100$. If for all $0 \leq i \leq 99$, we predict $P_{i+h} > P_{i+1}$, then for we will place buy orders for all $0 \leq i \leq 99$. Hence the maximum we can trade in one day will be limited to $\$\frac{1}{99}$.

An alternative strategy we compare is the *market neutral strategy*. A concern of the *basic strategy* is how our portfolio positions depend on the overall market bias, and will be affected by the total market movement. This is also illustrated in Figure 5. To construct a market neutral strategy such that our returns are independent of the overall market movement, the mean position for each day is subtracted. As an example, if a trade a basket of three stocks *A, B, C*, and our *basic strategy* positions were $-1/3, 1/3, 1/3$ respectively, then our *market neutral strategy* positions will be $-4/9, 2/9, 2/9$, i.e. this mean subtraction is done on top of the *basic strategy*. Hence, instead of limiting our dailu budget, we limit our daily exposure to $1/(h-1)$. The results for this is shown in Figure 5.

V. Results

This section gives a comprehensive analysis of the fine-tuned TimesFM results following the experiments described in section IV and its comparison to the original version as well as some popular past benchmarks.

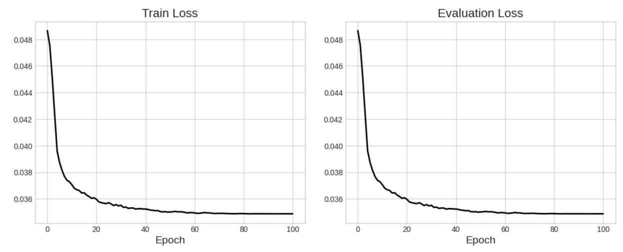

Fig. 2. Training and Validation loss curves for fine tuning TimesFM following the recipe in Table II. Training generally asymptotes at around 70% of the original loss value.

A. Training results: Loss curves

As shown in Figure 2, training usually asymptotes at around 70% of the original loss value. Noise in training is present due to the random masking augmentation described in the previous section. Note that extending training past 100 epochs, or using a larger learning rate gives preliminary signs of overfitting. We recommend, for future work, the use of larger training set, stronger data augmentation or early stopping for better generalization.

We note that the loss values in Figure 2 display the loss after performing the log transformation, and does not immediately translate into the same performance when computing MSE loss evaluated on the original samples. We verified that MSE loss also certainly decreases.

While this demonstrates learning capabilities of the model, MSE loss can be easily reduced by many methods. Due to the similarities between train and validation sets (both drawn from data before 2023), overall market trends and high correlation between prices can incentivize a model to learn by memorization of seen patterns.

In the following sections, we evaluate the performance of this model on the test set: data from 2023 onwards.

B. Metric: Accuracy

We observe, in Figure 3, that through fine-tuning, we are able to see consistently better performance over the vanilla pre-trained TimesFM across horizon lengths from 2 to 128. As a benchmark, we provide the chance rate, calculated as the accuracy obtained by a random model. For example, if 53% of the price changes in the test set is up, the random model guesses up 53% of the time, and down 47% of the time. Our fine-tuned TimesFM is also able to outperform this benchmark, providing statistical confidence that the improvements we see are not due to random chance.

As an answer to our original research question, the following conclusions can be drawn from Figure 3,

1) Original TimesFM underperforms random chance on 4 out of 7 of the prediction horizons, suggesting that TimesFM cannot be used it its original state for price prediction
2) Fine-tuned TimesFM outperforms original TimesFM on all prediction horizons, showing how fine-tuning on financial data significantly improves performance.

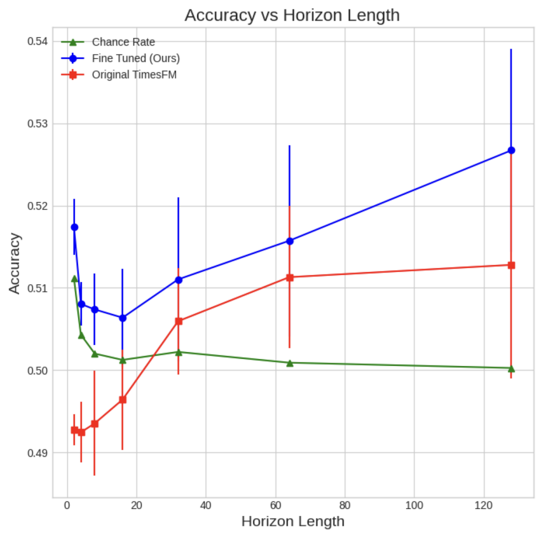

Fig. 3. Accuracy score of fine-tuned versus original TimesFM and a chance-rate model, when evaluated on the full test set.

3) Fine-tuned TimesFM outperforms random chance on all prediction horizon, hinting at statistically significant performance.

C. Metric: F1-score

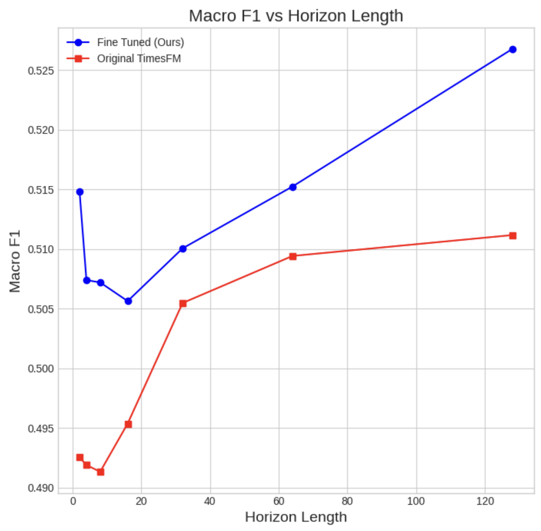

Fig. 4. Macro F1 score of fine-tuned versus original TimesFM, when evaluated on the full test set.

A more rigorous of the model performance with the Macro F1-score shown in Figure 4 show identical trends to Figure 3, where the fine-tuned TimesFM consistently outperforms random chance and the baseline model. This suggests a conclusive answer to our original research question: **foundation time series models can perform well on price data in the financial markets after fine-tuning.**

D. Mock Trading

In the previous section, we have demonstrated that our fine-tuned model is able to outperform standard benchmarks (original TimesFM and a chance-rate model) in price prediction tasks by a reliable and significant margin. Using the strategy outlined in section IV-C3, we conduct mock trading of the fine-tuned TimesFM on daily data from the S&P 500 index starting from January 1, 2023. The returns generated from executing this strategy (in a zero cost setting) are presented in Figure 5.

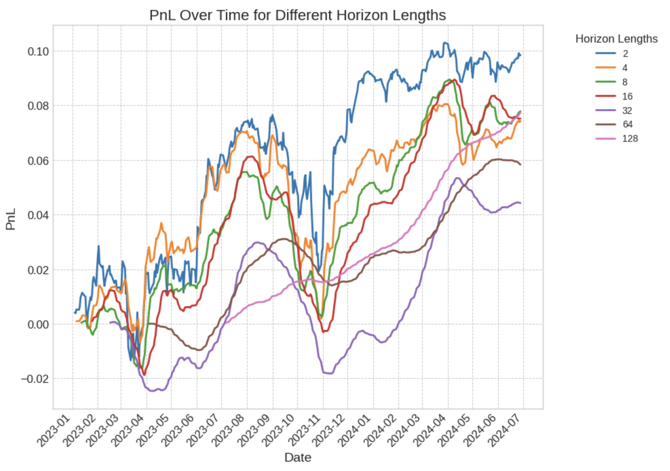

Fig. 5. Realized PnL using fine-tuned TimesFM traded on S&P500 stocks using the basic strategy, assuming no trading costs.

Using the basic strategy, we see consistently positive gains over each horizon length at the end of the trading period. Note that using a horizon length of H, the first returns will only be realized on day H, so the start points for each horizon length is different.

While we are able to observe maximum returns of up to 10% (using horizon length of 2), this basic strategy is highly volatile, due to its dependence on the overall market movement. In contrast, results in Figure 6 show that the market neutral strategy is effective in reducing the overall volatility while ensuring positive returns for majority of horizon lengths.

Furthermore, we provide additionally useful metrics for evaluating the performance of the market neutral strategy.

TABLE III
Performance Metrics by Horizon

Horizon	Ann Sharpe	Max Drawdown	Ann Returns	Ann Volatility	Neutral Cost (%)
2	0.516	-0.015	0.013	0.024	0.003
4	-0.483	-0.028	-0.009	0.019	-0.006
8	0.227	-0.017	0.005	0.022	0.007
16	0.003	-0.019	0.000	0.024	0.000
32	0.420	-0.015	0.014	0.034	0.080
64	1.285	-0.002	0.033	0.026	0.347
128	1.679	-0.001	0.036	0.021	0.600

In general, we observe the more desirable performance for larger horizon lengths, with $H = 128$ achieving annual returns of 3.6% and an annual sharpe of 1.68. Neutral cost, defined as the cost basis to zero out returns at the end of the trading period, increases with larger horizon lengths as we average the

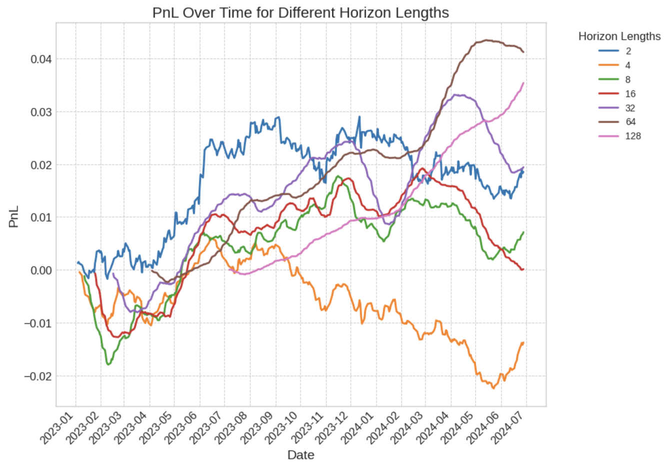

Fig. 6. Realized PnL using fine-tuned TimesFM traded on S&P500 stocks using the market neutral strategy, assuming no trading costs.

price movements out over slower moving strategies. Again, the largest horizon length of 128 can be traded up to a cost of 0.60%.

Next, we focus on a horizon length of 128 and compare our proposed model against several others. We focus on the market neutral strategy, comparing our proposed fine-tuned TimesFM against the original TimesFM, as well as a random model and an AR(1) model [2].

For the construction of a random model, we proceed according to the chance rate calculation presented in Figure 3. To recap, this model is created by first calculating the ratio up:down within the whole dataset, then at each day i, guess the sign of $P_{i+H} - P_{i+1}$ according to this ratio.

The AR(1) model is an autoregressive model fitted with only the single-difference lagged term. To implement this, on each time series (in the training period) we fit an AR(1) model to obtain the coefficients, then predict on the test dates. Then, mean subtraction is done to turn this into a market neutral strategy.

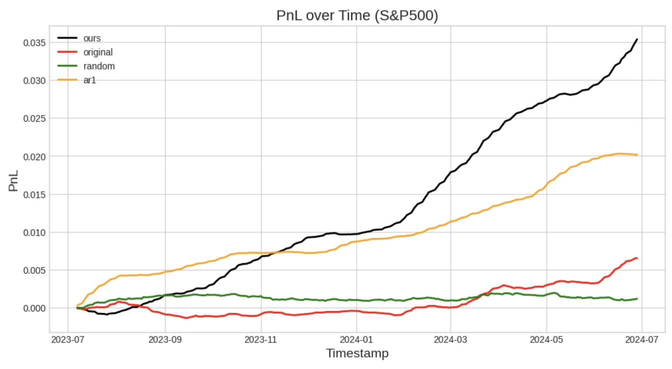

Fig. 7. Realized PnL comparison between various models traded on S&P500 stocks using the market neutral strategy in a cost-free setting.

Results shown in Figure 7 illustrate the stronger performance of our fine-tuned TimesFM over other models. The random model performs poorly in a market neutral setting, while the AR1 model also shows lower returns than fine-tuned TimesFM.

Rigorous testing results across different markets, evaluated on sharpe ratio and neutral cost, are shown in Tables IV and V respectively.

TABLE IV
COMPARISON OF SHARPE RATIO ACROSS MODELS/MARKETS

	Ours	Original TimesFM	Random	AR1
S&P500	1.68	0.42	0.03	1.58
TOPIX500	1.06	-1.75	0.11	-0.82
Currencies	0.25	-0.04	-0.03	0.88
Crypto Daily	0.26	-0.03	0.01	0.17

Our model outperforms the original TimesFM on all benchmarks, and the random model cannot make any reliable predictions under a market neutral situation.

However, performance of our model on currencies and crypto is left to be desired. Significantly underperforming the AR1 model. Nonetheless, our fine-tuned TimesFM is still the only model to achieve positive returns on every market.

TABLE V
COMPARISON OF NEUTRAL COST ACROSS MODELS/MARKETS

	Ours	Original TimesFM	Random	AR1
S&P500	0.60%	0.11%	-0.008%	0.34%
TOPIX500	0.14%	-0.24%	0.02%	-0.18%
Currencies	0.08%	-0.017%	-0.008%	0.27%
Crypto Daily	0.44%	-0.07%	0.010%	0.88%

VI. DISCUSSION

In section V, we have rigorously shown the viability of fine-tuning a foundation time series model (TimesFM) for the task of price prediction on financial markets. Nonetheless, our results raise several questions and motivations for future work.

In preparation of the fine-tuning data, we used a mix of data from various markets and granularities. However, majority of training data was dominated by hourly cryptocurrency and stock data, which may result in biases during training towards a specific granularity or market. One could potentially upsample the underrepresented granularity or market data to balance the dataset, as was done in TimesFM, but keeping in mind the repitition of data that might deteriorate model performance.

In TimesFM, it was also shown that including synthetic data in training, specifically time series given by simple mathematical functions, improves model performance even when evaluated on real-world time seris information. In other modalities, synthetic data has also shown to benefit model performance [34], [35] and the authors question to what extent synthetic data can benefit a time series model for financial price prediction here.

During the training process, several decisions were empirically made about the loss function and the masking scheme to tweak TimesFM for fine-tuning on financial data. Another potential loss function can be to compute $\log(MSE)$ instead of $MSE(\log)$ and our preliminary observations show that they give rather similar results. Out of the scope of this paper, but more recently supported by the TimesFM authors, is training with quantile loss where the model also outputs confidence scores and quantiles during inference time.

While we chose to fine-tune with continual pre-training, this is one of the slowest methods of fine-tuning, which was only feasible in this situation due to the rather limited size of our dataset. Such a fine-tuning method also increases the magnitude of the change in the model weights compared to before fine-tuning. Other alternatives such as freezing model weights, LoRA [36] can help the model make smaller updates during fine-tuning while still achieving desired performance.

In our evaluation experiments, we have also seen that the original TimesFM significantly underperforms even the most basic AR1 model, giving statistically insignificant and even negative returns on most of the markets. While this shows the benefit of fine-tuning TimesFM, we question where the performance bottleneck in the original TimesFM lies. The authors hypothesize the irregularities of price data compared to the regular time series data that TimesFM is trained on to be a primary factor causing TimesFM to be unable to capture the underlying market dynamics. Additional experiments of original TimesFM on a wider range of data complexities, granularities, trading periods can help to elucidate the differences. A helpful comparison would be the difference between the original and fine-tuned weights of TimesFM.

We observed that accuracy in price prediction task improves, what happens to the performance on generic time series forecasting? Specifically within language models, fine-tuning can deteriorate generalization performance by destroying pre-trained features [37]. Specifically for a fine-tuning dataset like ours, containing price data with little to no correlation with standard time series data, fine-tuning has the potential of worsening performance on general benchmarks. A next step to this would be to evaluate the model through MAE (mean average error) scores on benchmarks like Darts [29], Monash [30] and Informer [31] as was used in TimesFM.

While fine-tuning improves TimesFM over its baseline, we are unable to ascertain consistently better performance over just a simple AR1 model. How should we improve the fine-tuning to surpass AR1? How would its performance compare to other autoregressive models? Possible directions include crafting a better fine-tuning dataset with balance over different granularities [14], or adjusting the loss function to perform probabilities forecasting through quantile loss.

This also raises the question of what exactly is the model learning? Comparisons in Tables V and IV show that it is not entirely just based off a single autoregressive term, else our metrics would have strong similarity to AR1 across all markets. Computing the correlation in the predicted prices between TimesFM and AR1, as well as with other autoregressive models, or probing the internal activations with linear probing techniques [38], can help explore what kind of momentum (or some other) strategies a large time series model is learning.

VII. Conclusion

In this paper, we have fine-tuned a time series foundation model [14] for usage on financial data. By evaluating the loss and accuracy of the model, we see that fine-tuning allows to achieve significantly superior results using the large capacity of TimesFM, outperforming traditional models.

We tested this model by constructing a trading strategy that places buy/sell trades according to the predictions of the model. Through thorough evaluation, we found that a market neutral strategy with a long horizon gives consistently better performance over traditional models, with a sharpe ratio up to 1.68 when traded on S&P500.

We publish our code and model weights for reproducibility of results, and hope it inspires future research in this direction.

References

[1] E. McKenzie, "General exponential smoothing and the equivalent arma process," *Journal of Forecasting*, vol. 3, no. 3, pp. 333–344, 1984.

[2] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 1970.

[3] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-beats: Neural basis expansion analysis for interpretable time series forecasting," *International Conference on Learning Representations (ICLR)*, 2020.

[4] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A time series is worth 64 words: Long-term forecasting with transformers," *International Conference on Learning Representations (ICLR)*, 2022.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 5999–6009.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *Association for Computational Linguistics*, 2018.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[8] N. Gruver, M. Finzi, S. Qiu, and A. G. Wilson, "Large language models are zero-shot time series forecasters," *Neural Information Processing Systems*, 2023.

[9] M. Jin, S. Wang, L. Ma, Z. Chu, J. Y. Zhang, X. Shi, P.-Y. Chen, Y. Liang, Y.-F. Li, S. Pan, and Q. Wen, "Time-llm: Time series forecasting by reprogramming large language models," *International Conference of Learning Representations*, 2024. [Online]. Available: https://arxiv.org/abs/2310.01728

[10] X. Zhang, R. R. Chowdhury, R. K. Gupta, and J. Shang, "Large language models for time series: A survey," *International Joint Conference on Artificial Intelligence*, 2024. [Online]. Available: https://arxiv.org/abs/2402.01801

[11] J. Requeima, J. Bronskill, D. Choi, R. E. Turner, and D. Duvenaud, "Llm processes: Numerical predictive distributions conditioned on natural language," 2024. [Online]. Available: https://arxiv.org/abs/2405.12856

[12] C. Sun, H. Li, Y. Li, and S. Hong, "Test: Text prototype aligned embedding to activate llm's ability for time series," *International Conference of Learning Representations*, 2024. [Online]. Available: https://arxiv.org/abs/2308.08241

[13] M. Tan, M. A. Merrill, V. Gupta, T. Althoff, and T. Hartvigsen, "Are language models actually useful for time series forecasting?" 2024. [Online]. Available: https://arxiv.org/abs/2406.16964

[14] A. Das, W. Kong, R. Sen, and Y. Zhou, "A decoder-only foundation model for time-series forecasting," *International Conference of Machine Learning*, 2024. [Online]. Available: https://arxiv.org/abs/2310.10688

[15] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, 1986. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304407686900631

[16] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, 1960.

[17] L. Catello, L. Ruggiero, L. Schiavone, and M. Valentino, "Hidden markov models for stock market prediction," 2023. [Online]. Available: https://arxiv.org/abs/2310.03775

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Readings in cognitive science*, 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:62245742

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[20] M. Liu, J. Huo, Y. Wu, and J. Wu, "Stock market trend analysis using hidden markov model and long short term memory," 2021. [Online]. Available: https://arxiv.org/abs/2104.09700

[21] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[22] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[23] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mixtral of experts," 2024. [Online]. Available: https://arxiv.org/abs/2401.04088

[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *Proceedings of the International Conference on Learning Representations*, 2021.

[25] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "Vivit: A video vision transformer," 2021. [Online]. Available: https://arxiv.org/abs/2103.15691

[26] W. Peebles and S. Xie, "Scalable diffusion models with transformers," 2023. [Online]. Available: https://arxiv.org/abs/2212.09748

[27] A. Garza, C. Challu, and M. Mergenthaler-Canseco, "Timegpt-1," 2024. [Online]. Available: https://arxiv.org/abs/2310.03589

[28] H. Yang, X.-Y. Liu, and C. D. Wang, "Fingpt: Open-source financial large language models," 2023. [Online]. Available: https://arxiv.org/abs/2306.06031

[29] J. Herzen, F. Lässig, S. G. Piazzetta, T. Neuer, L. Tafti, G. Raille, T. V. Pottelbergh, M. Pasieka, A. Skrodzki, N. Huguenin, M. Dumonal, J. Kościsz, D. Bader, F. Gusset, M. Benheddi, C. Williamson, M. Kosinski, M. Petrik, and G. Grosch, "Darts: User-friendly modern machine learning for time series," *Journal of Machine Learning Research*, 2022. [Online]. Available: https://arxiv.org/abs/2110.03224

[30] R. Godahewa, C. Bergmeir, G. I. Webb, R. J. Hyndman, and P. Montero-Manso, "Monash time series forecasting archive," 2021. [Online]. Available: https://arxiv.org/abs/2105.06643

[31] 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," *Proceedings of the AAAI conference on artificial intelligence*, 2021.

[32] J. Brabec, T. Komárek, V. Franc, and L. Machlica, "On model evaluation under non-constant class imbalance," 2020. [Online]. Available: https://arxiv.org/abs/2001.05571

[33] J. Opitz and S. Burst, "Macro f1 and macro f1," 2021. [Online]. Available: https://arxiv.org/abs/1911.03347

[34] S. Azizi, S. Kornblith, C. Saharia, M. Norouzi, and D. J. Fleet, "Synthetic data from diffusion models improves imagenet classification," 2023. [Online]. Available: https://arxiv.org/abs/2304.08466

[35] Y.-w. Kim, S. Mishra, S. Jin, R. Panda, H. Kuehne, L. Karlinsky, V. Saligrama, K. Saenko, A. Oliva, and R. Feris, "How transferable are video representations based on synthetic data?" in *Advances in Neural Information Processing Systems*, S. Koyejo,

S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 35 710–35 723. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/e8507db80464ced5658d16b49bd458b9-Paper-Datasets_and_Benchmarks.pdf

[36] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[37] A. Kumar, A. Raghunathan, R. M. Jones, T. Ma, and P. Liang, "Fine-tuning can distort pretrained features and underperform out-of-distribution," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=UYneFzXSJWh

[38] G. Alain and Y. Bengio, "Understanding intermediate layers using linear classifier probes," 2018. [Online]. Available: https://arxiv.org/abs/1610.01644