Distributional Forecasting for Exchange Rates

by Arne Tillmann, Nils Wüstefeld

Submitted in fulfillment of the requirements
for the course Statistical Practical

September 15, 2024

Study Program: Applied Statistics M.Sc.
Faculty: Faculty of Business and Economics
Chair: Chair of Statistics

1 Abstract

Forecasting plays a crucial role in business operations and various other sectors. Organizations strive to generate the most accurate forecasts possible to efficiently allocate their resources. The BASF Group, a global leader in the chemical industry, is engaged in daily transactions within the foreign exchange market. To mitigate financial risk and potentially leverage market fluctuations, BASF is particularly focused on predicting foreign exchange volatility over short time frames, such as three days. Although conventional wisdom suggests that ultra-short-term market volatility is largely driven by random noise, anecdotal evidence from the trading and currency unit within BASF indicates otherwise. This study assesses several models for distributional forecasting, including statistical methods, deep learning techniques, transformers, and large language models (LLMs). Our findings show only a slight improvement in performance compared to a naive model, though there is significant potential for further improvement.

2 Introduction

Forecasting is a critical skill across various domains. The ability to accurately predict future events based on current knowledge significantly impacts almost every aspect of operations. By reducing uncertainty with accurate forecasts, available resources can be allocated more efficiently. Therefore, organizations are highly interested in making the most accurate forecasts possible. In finance, robust forecasting models are particularly necessary to mitigate exposure to market volatility (Poon and Granger 2003). Here, the main interest lies not in point forecasts but rather in identifying trends and target ranges (Menkhoff 2010).

For this purpose, a broad range of proven statistical tools for distributional forecasting is available. In particular, distributional forecasting allows for modeling trends alongside the probability of their occurrence. Distributional forecasting models therefore provide users with additional valuable information about the risks involved in using the forecast for decision-making (Gaglianono and Lima 2012). Among these, classic statistical models such as ARIMA and ETS have traditionally served as the standard for forecasting. Recently, there has been a shift toward deep learning models (Hyndman and Athanasopoulos 2018). The broader availability of computing power and data has fostered the emergence of machine learning and deep learning approaches in forecasting (Ansari et al. 2024). This avenue may provide a promising new approach to distributional forecasting.

In this work, we present and evaluate a broad range of models including statistical models and deep learning approaches for distributional exchange rate forecasting. We illustrate the application of these models by evaluating the US-Dollar/Euro exchange rate from [Month] 2024 to [Month] 2024. Our results indicate that the Auto-ETS model performed best in the presented setup for our use-case. However, the resulting prediction intervals are not sufficiently accurate, and further improvement and evaluation are necessary.

The remainder of this paper is structured as follows: In section 3, we explain the use case and context of this work. In section 4 a brief overview of related works and the theoretical background of distributional time series forecasting and exchange rate forecasting is provided. section 5 reserved for the data and its processing. Following that, the employed models are described in section 6. In section 7, the results are illustrated and section 8is reserved for the discussion of the results. The last section, section 9 contains the conclusion and final remarks of this study.

3 Use Case and Motivation

The BASF Group is one of the world's largest chemical companies, operating 239 production sites across 91 countries. Its primary focus includes chemicals, surface technologies, materials, nutrition and care, agriculture, and industrial solutions. BASF serves approximately 82,000 customers and sources goods and services from over 70,000 Tier 1 suppliers[1] (BASF 2022). As a globally operating company with an extensive supply network, maintaining adequate liquidity for the high volume of daily financial transactions is crucial.

The importance of liquidity is not only in the amount but also in the currency in which it is held. As a German company headquartered in Ludwigshafen, BASF is required to manage its accounts and books in euros. However, transactions with international business partners often require currencies other than the euro. For instance, an American supplier would typically prefer payment in US dollars. Additionally, major resources such as carbohydrates and derived products are exclusively traded in US dollars. In summary, BASF maintains its books in euros while transacting in various currencies. As a consequence, BASF is dependent on the foreign exchange market (forex), to obtain the right amount of currency in the right denomination to ensure cash flow.

A typical transaction is conducted as follows: The value of the sale or invoice is calculated and recorded using the current euro exchange rate. To

[1]Tier 1 suppliers are the final link in the supply chain before the manufacturer, producing refined and highly processed pre-components.

fulfill the transaction, a foreign currency is obtained on the forex market. However, this practice carries a certain degree of risk, as the forex market is subject to volatility. As a consequence, BASF may encounter less favorable exchange rates than anticipated and recorded. Volatility can pose a significant risk, especially for currencies with lower trading volumes. In practice, BASF mitigates this risk by hedging 80% of the transaction volume through spot transactions. Nevertheless, volatility can also work to BASF's advantage. By postponing the closure of remaining 20% of its foreign exchange positions, BASF may secure a more favorable exchange rate, potentially resulting in a modest profit relative to the accounted value. By employing this short-term volatility speculation strategy, BASF's currency department has consistently outperformed the market and realized profits. This contrasts with the classic Messe-Rogoff-Puzzle presented in Meese and Rogoff (1983). The puzzle suggests that, to a certain extent, short-term exchange rates are dominated by random-walk behavior.

In contrast, the analysis of historical transactions of BASF suggests that the profitability of these trades cannot be attributed to chance alone. This contradiction could imply a noticeable gap in the financial market literature on the short-term. Based on this evidence, we hypothesize that short-term forex market volatility follows stationary distributional behavior, rather than being dominated by random walks.

4 Related Work and Background

4.1 Distributional Time Series Forecasting

Distributional forecasting, also known as probabilistic forecasting, serves as a method to predict the future while quantifying the uncertainty of the prediction (Gneiting and Katzfuss 2014). From a different point of view, one can argue that distributional time series forecasting is a method to predict the joint distribution of the next time steps (Gneiting and Katzfuss 2014; Ansari et al. 2024). This task is challenging, as time series data consists of three components: a trend-cycle component, a seasonal component, and a remainder component (Hyndman and Athanasopoulos 2018). Identifying these patterns is not trivial, as appropriate forecasts require recognizing a wide range of linear and non-linear dependencies.

4.2 Forex

The foreign exchange market (forex) is the largest financial market in the world, with a daily volume of over-the-counter financial products totaling $918.4 billion across all currency pairs in 2023. The EUR/USD exchange accounts for a volume of $16.6 billion in April 2023. What makes the forex market unique is its decentralized structure across the four major financial hubs: London, New York, Tokyo and Sydney. Due to their geographic locations, at least one exchange is in trading session, allowing trades 24 hours a day, except on weekends. The high volume and the high availability of the forex market make it difficult to predict movement accurately due to the high variation and irregular seasons of non-linearity (Ahmed et al. 2020). However, there is debate over whether the forex market has underlying distributional patterns. As mentioned above, the puzzle suggests that, to a certain extent, short-term exchange rates are dominated by random-walk behavior.

In contrast, mean reversion, popularized by Poterba and Summers (1988) and Fama and French (1988), assumes that exchange rates tend to revert to a long-term equilibrium. Furthermore, Ca' Zorzi, Muck, and Rubaszek (2016) showed that real exchange rates exhibit mean-reverting behavior. In addition to endogenous market dynamics, exogenous shocks significantly impact the foreign exchange market. Central banks and policymakers exert direct influence through mechanisms such as setting interest rates and controlling liquidity. Moreover, macroeconomic releases and surveys shape market behavior by affecting investor expectations. These exogenous factors, often referred to as "economic surprises," play a critical role in driving fluctuations and shifting the scale of the exchange rates."

4.3 Stationary Processes on the Forex

The foundation of high-frequency trading is the stationary process (Li et al. 2019). A stationary stochastic process is a stochastic process, whose distribution does not change over time. Consequently, the mean and variance are not time-dependent and remain constant. Stationary processes can be divided into strongly stationary processes and weakly stationary processes.

A stochastic process $X(t)$ is called strongly stationary if, for any $a > 0$ and any finite times $t_1 < t_2 < \cdots < t_n$, it satisfies the following condition:

$$P(X(t_1), X(t_2), ..., X(t_n)) = X(t_1 + a), X(t_2 + a), ..., X(t_n + a)$$

where $X_(t)$ is a random vector at time t. In other words, a strongly stationary process preserves its entire distribution over time. In contrast, a weakly stationary processes, which only require the first two moments, the mean

and variance, to remain constant. A stochastic process $X(t)$ is called weakly stationary if the following conditions are satisfied:

$$E[X(t)] = E[X(t+c)];$$

$$E[X(s)X(t)] = E[X(s+c)X(t+c)];$$

where c is larger than 0. Hence, a stochastic process is a weakly stochastic process if the expectation of a random vector at time t is equal to the expectation of the vector at time $t+c$. Furthermore, the expectation of the product of two vectors is equal at time t and $t+c$. This definition of weak stationarity is more applicable to financial markets, as the conditions for strong stationarity are often too restrictive (Li et al. 2019).

4.4 Mean Reversion Strategy

Based on the assumption that part of the governing behavior of the forex is due to stationarity, we can formulate a simple trading strategy:

Algorithm 1: Mean Reversion Trading Strategy

Data: Price series $P(t)$ over time
Result: Trading signals: Buy, Sell, or Hold
for *each time step* t **do**
  Compute the quantiles of the price series up to time t ;
  Let q_{20} be the 20% quantile of the price series;
  Let q_{10} be the 10% quantile of the price series;
  Let Q_{80} be the 80% quantile of the price series;
  Let Q_{90} be the 90% quantile of the price series;
  if $P(t) < q_{20}$ ***and*** $P(t) > q_{10}$ **then**
    Buy signal ;
  else
    if $P(t) > Q_{80}$ ***and*** $P(t) < Q_{90}$ **then**
      Sell signal ;
    else
      Hold ;
    end
  end
end

Excluding the lower 10% quantile for investing is important to avoid investments when the price level has fallen, and conversely, to hold when the

price level has risen, as fluctuations might not be explained by stationarity. The challenge for the model will be distinguishing between stationarity and random level shifts.

(Li et al. 2019)

5 Data

To train and evaluate the models, data from the EUR/USD forex market is used. The data consists of EUR/USD foreign exchange rates from 2022 to 2024 at 15-minute intervals in the Open High Low Close (OHLC) format. As we are only interested in the close rates, the other values are disregarded. An illustration of the close rates for a selected day is presented in figure 1.

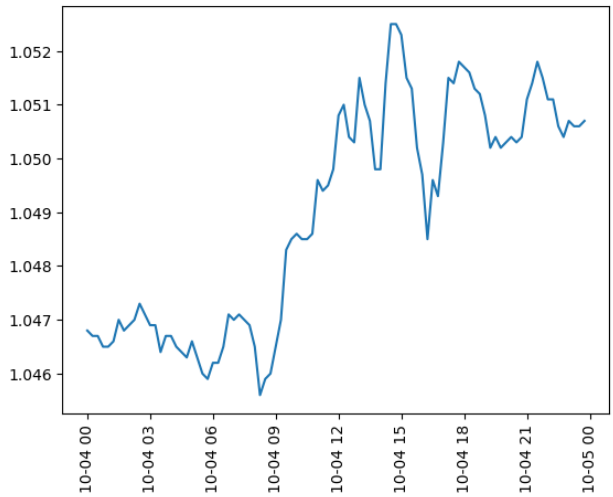

Figure 1: EUR/USD Close Rates for 10. of April 2023.
The figure shows the closing rates of EUR/USD forex for April 10, 2023. For this day, the exchange rate fluctuates by approximately 60 basis points, implying high volatility.

Since most models used in this work require fixed input dimensions, the data must be preprocessed to meet this requirement. A sequence size of 96 observations is chosen to capture all trading sessions, resulting in a 24-hour window. By removing incomplete days, we obtain a total of 109 full days. To increase the number of observations, a sliding window with a specified stride is employed. Sliding windows are a common approach to increase the number of observations without altering the data. We are left with 164 overlapping

sequences of each 96 observations. As mentioned above, the major financial markets are closed on weekends and resume trading on Monday. To avoid jumps in the sequence the sliding window uses block mapping to group data points. If the adjustment of the sliding window would incorporate data from a different block, the time difference of at least two data points would have a date-time difference of more than exactly 15 minutes, leading to a jump in the sequence. If this occurs, the proposed window is discarded, and the sliding algorithm starts at the beginning of the next block. An illustration of the preprocessing algorithm is provided here:

Algorithm 3: Sliding Window Algorithm for Data Processing.
This algorithm creates samples from the time series data by using a sliding window to extract overlapping samples of same length.

Input: **Data**: A dataset of EUR-USD exchange rates. Each row represents a close price of a 15 minute interval
Size: Length of the sliding window
Stride: Stride length of the sliding window
Output: **Windows**: A data frame with unique IDs for each sequence

```
Function SlidingWindow(Data, Size, Stride):
    start ← 0 ;
    currentID ← 0 ;
    while start + Size ≤ length(Data) do
        end ← start + Size ;
        window ← Data.iloc[start:end] ;
        if difference of any data points > 15 minutes then
            start ← start + Stride ;
            Continue ;
        end
        window[ID] ← currentID ;
        Windows.append(window) ;
        currentID ← currentID + 1 ;
        start ← start + Stride ;
    end
    return Windows ;
```

Furthermore, the code for this paper, including the original dataset and all results, is available in the GitHub repository associated with this work.

For training and testing, the sequences are grouped into instances of four days. Three days are for training, while the last day is reserved for evalu-

ation. This approach is motivated by the mean-reversion assumption and our anecdotal evidence. We assume stationary for the mean of the exchange rates for a period of up to five days. For the sake of simplicity, we do not account for economic surprises or external effects on volatility. While changes in local demand for euros or US dollars during each session may influence market volatility, this effect is neglected due to the sheer volume of orders, even in sessions with relatively low demand. Furthermore, changes in inflation or adjustments due to economic surprises or other external factors are neglected. Changes in inflation are accounted for by batch normalization. Eventually, the data is batched and loaded into the model via a data loader. The data is split into training and validation sets. For testing, data from the Yahoo Finance plugin (Roussi 2019) is obtained for June and July 2024.

6 Methods

This section discusses the methods used in this study. As outlined in the literature review, traditional forex analysis consists of two main approaches: fundamental analysis and technical analysis. However, the focus of this study is limited to methods within technical analysis. These methods can be categorized into statistical and deep learning approaches. The deep learning models can be further divided into transformer-based models, large language models, and non-transformer models.

For the purpose of this study, we first discuss classic statistical models, then our own implementation of a LSTM-VAE, followed by the model zoo of the autogluon library (Shchur et al. 2023).

6.1 GARCH

The Generalized AutoRegressive Conditional Heteroscedasticity (GARCH) model is a stochastic model of order (p, q) for the evaluation of conditional variances in autoregressive data, first proposed by Bollerslev (1987). The GARCH incorporates p, the number of lagged conditional variances (GARCH terms) and q, the number of lagged squared error terms (ARCH terms) in the model. By adjusting p and q, complex volatility patterns can be captured. It is a popular approach to predict financial volatility, such as in Chong, Ahmad, and Abdullah (1999) or more recently for the investigation of the impact of Covid-19 on financial markets in Khan et al. (2023).

The model can be defined by the following set of equations:

$$X_t = \sigma_t \epsilon_t \tag{1}$$

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i X_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2 \tag{2}$$

$$\text{where} \quad \epsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0,1) \quad \text{and} \quad \omega, \beta_j, \alpha_i \geq 0. \tag{3}$$

In this formulation the conditional variance σ^2 is recursively defined as linear combination of the lagged previous versions of itself. Eventually, we can decompose the parts of the model into the ARCH effects and the GARCH effects to enable the lagged conditional variances (Bollerslev 1987).

6.2 ARMA-GARCH

Besides the classic GARCH model, the ARMA-GARCH is a popular GARCH-based approach. The ARMA-GARCH extends the GARCH model by incorporating an AutoRegressive Moving Average (ARMA). This is useful for financial modeling since the moving average (MA) is one of the most popular technical indicators in finance (Li et al. 2019). In general, the ARMA-GARCH can be understood as a two-part model. First, the ARMA model calculates the conditional mean, then the residuals of the ARMA model are then used as input for the GARCH model. This approach allows for simultaneous calculation of the conditional mean and variance. The ARMA model can be described as follows:

$$X_t = c_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{j=1}^{q} \theta_i \epsilon_{t-i}$$

where c_t is a constant, ϵ are the residuals, φ_i are the auto-regressive components and θ_i are the moving average components. The AR part of this equation $\sum_{i=1}^{p} \varphi X_{t-i}$ creates predictions based on previous observations, while the MA part $\sum_{j=1}^{q} \theta_i$ models the dependence of an observation based on previous residuals (Makridakis and Hibon 1997). By incorporating the conditional average, calculated by the ARMA, the ARMA-GARCH can produce better forecasts for financial data than the GARCH model.

6.3 LSTM-VAE

For providing suitable forecasts of the forex market, a model architecture is required to make the predictions based on previously learned patterns of past

observations. However, length and influence of the patterns may vary over periods, from short trends to long-term seasonality. Especially, if the patterns are non-linear, this poses a significant challenge. To capture and learn these patterns, a model should have the ability to retain information from the past. Simultaneously, the model must be able to construct quantiles for the predictions to quantify the uncertainty. Based on these requirements and the understanding of our research question, we concluded that a LSTM-VAE may be a suitable model for distributional exchange rate forecasting. This model combines the time-dependent properties of the "long short-term memory" (LSTM) and stochastic properties of the "Variational Auto Encoder" (VAE) and combine both into a new model. The LSTM layers serve as the memory of the model, remembering information across periods of varying length while the VAE generates a distribution form the learned data.

6.3.1 LSTM

The Long Short-Term Memory (LSTM) is a form of gated recurrent neural network (gated RNN) originally designed by Hochreiter and Schmidhuber (1997). Gated RNNs are based on the idea, that each RNN cell learns which information can be forgotten or retained, thereby mitigating the vanishing gradient problem. For example, during training of deep neuronal networks with gradient based back propagation and non-ReLU activation, the gradients might become too small or zero after passing through too many activation functions. As a consequence, the layers of the network cannot adjust the weights properly, resulting in improper learning and eventually poor performance. RNNs, especially LSTMs can mitigate gradient vanishing by letting the gradient pass by the layer, resulting in better and more stable performance (Gers, Schmidhuber, and Cummins 2000). In past studies RNNs have performed well when predicting time series (Ahmed et al. 2020).

The LSTM differs fundamentally from other gated RNNs through the use of memory cells and its more complex structure and multiple gates (Gers, Schmidhuber, and Cummins 2000). In its most basic form, a LSTM consists of a forget gate, an input gate, an output gate, and the memory block consisting of one or more memory cells each with a current cell state. The cell state is determined by the current state of the cell and the cell state update. Moreover, the current state of the cell serves as a linear self-loop controlled by a forget gate unit $\{_i^{(i)}$ which maps the weights between 0 and 1 via a sigmoid function:

$$f_t = \sigma\left(W_f \circ [h_{t-1,x_t} + b_f]\right)$$

where $W_f \circ [h_{t-1,x_t} + b_i]$ is the Hadamard product of the weight matrix W_f, the input vector x_t and the hidden state of the previous time step h_{t-1}. The

b_f represents the bias vector. In other words, the cell state serves as the long-term memory of the cell and the forget gate regulates, how much information is retained from the previous state.

Next, the input gate i_t regulates the flow of new information influencing the state update. Like the forget gate, a sigmoid function maps the weight matrix for the input:

$$i_t = \sigma\left(W_f \circ [h_{t-1}, x_t] + b_i\right)$$

where the sigmoid function σ maps the sum of the Hadamard product of the weight matrix of the input, the data vector and the last hidden state and the bias. The input gate determines the weight for the new proposed state of the cell. The proposed cell state state $\tilde{C}_t$ is calculated as follows :

$$\tilde{C}_t = tanh(W_C \circ [h_{t-1}, x_t] + b_C)$$

where a hyperbolic tangent function maps the Hadamard product of the weights and the input vector. By combing how much information of the previous state is forgotten and how much new information is retrieved, the cell can be updated. The current cell state can be obtained by the sum of the weighted previous state and the weighted proposed state:

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$

Eventually what's left is, what information should leave the cell via the output gate. Similar to the previous gates, the output gate regulates how much information leaves the cell:

$$o_t = \sigma(W_o \circ [h_{t+1}, x_t] + b_o)$$

whereas a sigmoid function maps the weighted hidden state and input vector plus the output bias. Finally, the current hidden state can be updated by the Hadamard product of the output function and the current state C_t, transformed by a hyperbolic tangent function:

$$h_t = o_t \circ tanh(C_t)$$

In summary, a LSTM cell is a combination of input- and output filters combined with a state, to decide how much information is kept from previous time steps. Commonly, multiple LSTM cells are combined into stacked cells to identify complex localized patters or hierarchical dependencies. The performance can be further enhanced by using LSTM cells with bi-directional information flow (Wang et al. 2023).

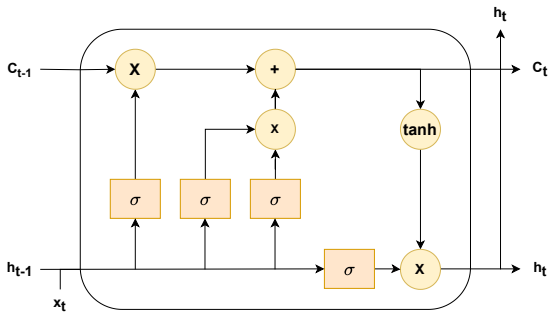

Figure 2: LSTM Memory Cell.
The LSTM cell receives as input the previous state C_{t-1}, the output of the previous state h_{t+1} and the input vector of the current state x_i. The gates of the LSTM are illustrated by the orange σ boxes. The operations are illustrated by the yellow circles, whereas "X" is the Hadamard operations and "+" represents addition.

LSTM-based models have recently emerged in the domain of financial prediction (Islam et al. 2020; Ahmed et al. 2020). The LSTM has shown that it can store non-linear dynamics better than its predecessors (Wang et al. 2023). As such, various models based on LSTMs have been applied to predicting financial markets. Ahmed et al. (2020) compared a Multi-LSTM, OHLC-LSTM and ForexLossFunction-LSTM (FLF-LSTM) model. The FLF-LSTM demonstrated a significant error reduction compared to both other tested models. Wang et al. (2023) proposed a DAFA-BiLSTM network. By combining a vector autoregressive layer to a feature augmented bidirectional LSTM-network, the model achieved high performance across multiple benchmark data sets. However, these models almost exclusively focus on point forecasting, which only has limited application in financial forecasting. Later in this section, LSTM-based models for distributional forecasting are introduced.

6.3.2 VAE

The second sub-model of the LSTM-VAE is the "Variational Auto Encoder" (VAE), a generative neuronal network (GNN) propose by Kingma (2013). The major motivation of the VAE is to learn the joint distribution over all variables and generate data from this learned distribution (Kingma, Welling, et al. 2019). In other words: The VAE is an approach to learn a latent representation from the original data and then sample from this latent distribution.

The Model can be understood as a combination of two independently parameterized models, which support each other. Namely the encoder/recognition model and the decoder/generative model (Kingma 2013).

Let us assume the data set $X = \{x^{(i)}\}_{i=1}^{N}$ to consist of N i.i.d. samples of a variable X, generated by an unobserved process involving the unobserved variable z. The encoder network approximates the underlying intractable true posterior based on its learned parameters ϕ:

$$q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(z|x), \Sigma_\phi(z|x))$$

whereas the approximated posterior is assumed to be normal distributed with mean μ and covariance Σ. The decoder model samples a vector z from a multinormal distribution given by the parameters from the encoder. In the next step, the decoder generates data by maximizing the likelihood given its set of parameters θ and the sampled z:

$$p_\theta(x|z) = \mathcal{N}(\mu_\theta(x|z), \Sigma_\theta(x|z))$$

The combination of both models allows us to calculate the evidence based lower bound (ELBO) of the data likelihood $p_\phi(x)$. Since it is difficult to obtain the (log) data likelihood directly, we can calculate the lower bound. The data likelihood is defined as:

$$log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[log p_\theta(x^{(i)})\right]$$

where $\mathbf{E}z \sim q\phi(z|x^{(i)})[\cdot]$ is the expectation from the distribution $q_\phi(z|x^{(i)})$ and $q_\phi(z|x^{(i)})$ is an approximation of the posterior distribution of latent variables z given the data point $x^{(i)}$.

By using Bayes' Rule and subtracting and adding the Kullbach-Leibler divergence, we obtain the following formula for the logarithmic data likelihood:

$$\begin{aligned}\log p_\theta(x^{(i)}) = \mathbb{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}\left(q_\phi(z|x^{(i)}) \parallel p(z)\right) \\ + D_{KL}\left(q_\phi(z|x^{(i)}) \parallel p(z)\right)\end{aligned}$$

whereas the terms can be summarized into the ELBO L and a Kulbach-Leibler term > 0:

$$\log p_\theta(x^{(i)}) = L(x^{(i)}, \theta, \phi) + D_{KL}\left(q_\phi(z|x^{(i)}) \parallel p(z)\right)$$

Finally, the variational lower bound is obtained:

$$\log p_\theta(x^{(i)}) \leq L(x^{(i)}, \theta, \phi)$$

During training of the VAE, the ELBO is maximized to obtain the optimal parameters for the encoder and decoder:

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} L(x^{(i)}, \theta, \phi)$$

In other words, we try to optimize the construction of the distribution of the input data while simultaneously trying to obtain a posterior distribution close to the sampled prior $z \in \mathcal{N}(\mu, \sigma^2)$. This is done by optimizing θ and ϕ simultaneously. In this case, the ELBO loss proves to be an elegant solution to circumvent to problem of the non-differentiable data likelihood. However, the ELBO has a noticeable drawback. Since it depends on the stochastic term z, the loss function is not differentiable due to the randomness of the vector. This poses a critical problem, since the VAE, like many other deep learning models, uses a gradient-based approach for its training. Hence, the gradients are required to back-propagate trough the model to update the weights of the layers. If z is not differentiable, the backpropagation algorithm gets stuck, and the model is unable to learn. To circumvent the problem, the reparameterization trick is applied to separate the stochastic part of z and create a deterministic path for the gradient flow (Kingma 2013). Since z is sampled from a Gaussian distribution, we can use its properties and recast the distribution of z, without changing its meaning. The reparameterization is expressed in the following equation:

$$z = \mathcal{N}(\mu, \sigma^2) \rightarrow z = \mu + \sigma \circ \epsilon, \epsilon \in \mathcal{N}(0, 1)$$

where σ^2 is transformed into $\sigma \circ \epsilon$, where the stochastic element ϵ is separated from our learnable parameters μ and σ. By applying this trick, we now consider $\epsilon \sim \mathcal{N}(0, 1)$ as a sample from a fixed source of noise allowing us to calculate the gradient of the parameterized Gaussian:

$$\bigtriangledown_\theta \mathbb{E}_{x \sim p_\theta(x)}[f(x)]$$

Since we consider ϵ as a source of noise from a fixed distribution, we can rewrite epsilon as follows:

$$\epsilon = \frac{z - \mu}{\sigma}$$

allowing us to express the standard normal random variable ϵ in terms of the original random variable z and the parameters μ and σ. We can now rewrite our gradient as:

$$\bigtriangledown_\theta \mathbb{E} z \sim q\theta(z|x)[f(z)] = \mathbb{E}\epsilon \sim \mathcal{N}(0, 1)[\bigtriangledown \theta f(\mu + \sigma \circ \epsilon)]$$

This transformation allows us to move the gradient operator inside the expectation. The key insight here is that we are no longer trying to compute

the gradient with respect to a random variable (z), but instead with respect to the deterministic parameters μ and σ of the distribution. In summary, the reparameterization trick effectively transforms the problem from one of estimating a gradient through a stochastic node to one of estimating a gradient through a deterministic function of the parameters and a fixed noise distribution. This allows the use of backpropagation to update the model parameters.

6.3.3 LSTM-VAE

Finally, by combining the LSTM with the VAE model, the LSTM-VAE is obtained. The LSTM-VAE is able to capture long and short-term properties of sequential data while simultaneously constructing a latent representation of the underlying distribution, hence satisfying the requirements previously stated. LSTM-VAE models and sub-variants have successfully been applied to various problems, in particular, anomaly detection and unsupervised classification. For example, in Lin et al. (2020) and in Niu, Yu, and Wu (2020), a LSTM-VAE model is employed to find anomalies in sensor data.

For this paper, a LSTM-VAE with stacked LSTM layers in the encoder and decoder is employed. The architecture can be described as followed: The encoder model consists of two LSTM layers followed by a dense layer and parallel dense layers for each the mean and the logarithmic variance. These layers are followed by a dense layer for the z. During training, the input for the z layer is calculated by means of the reparameterization trick, while in validation and testing, the input is the output of the mean layer. In the decoder, the output of the z layer is forwarded to two LSTM layers followed by a dense layer, which generates a sequence. The described architecture is illustrated in figure 3.

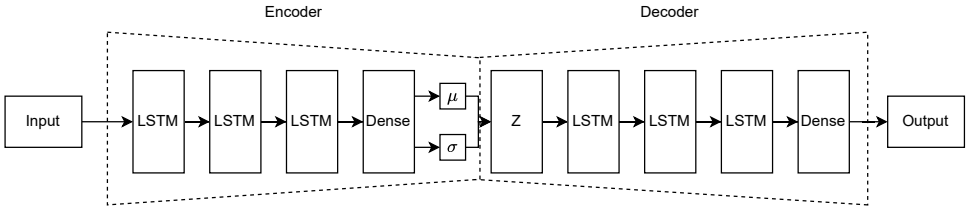

Figure 3: Architecture of the LSTM-VAE.
The Input is encoded via four LSTM-Layers and the parameters μ and σ for the multivariate normal distribution are constructed. In the decoder, for these parameters, the sample z is used to reconstruct the input via another four LSTM-layers and finally a dense layer.

6.4 Seasonal Naive

The Seasonal Naive model serves as the benchmark model for this experiment. The model estimates the next data point to be equal to the last observation of the same season:

$$Y_{T+h|T} = Y_{T+h-m(k+1)}$$

whereas the season m can be a week, month, year or other time frame and k is the number of complete terms in the forecast prior to $T+h$ (Hyndman and Athanasopoulos 2018). The simplicity of the model and its lack of parameters makes it a popular choice as benchmark model (Svetunkov and Petropoulos 2018).

6.5 Auto-ARIMA

The next model tested is the Auto-ARIMA model. Auto-ARIMA is an automatically tuned AutoRegressive Integrated Moving Average (ARIMA) model based on the implementation by Hyndman and Khandakar (2008). We consider a seasonal ARIMA$(p,d,q)(P,D,Q)_m$ process, characterized by non-seasonal components p, d, and q, corresponding to the autoregressive order, degree of differencing, and moving average order, respectively. The seasonal components are denoted by P (seasonal autoregressive order), D (seasonal differencing degree), and Q (seasonal moving average order). The model can be described as follows:

$$\Phi(B^m)\phi(B)(1-B^m)^D(1-B)^d y_t = c + \Theta(B^m)\theta(B)\epsilon_t$$

where $\Phi(z)$ and $\Theta(z)$ are polynomials of order P and Q, ϕ and θ are polynomials as well of order p and q. The backshift operator B describes the lag and is used in the differenced data $(1 - B^m)^D(1 - B)^d y_t$. The Auto-ARIMA picks the best order of the parameters p, q, P, Q, D, d step wise based on the AIC and returns the best fitting model.

6.6 Auto ETS

Similarly to the Auto-ARIMA, Auto-ETS is an automatically tuned forecasting algorithm based on the implementation by Hyndman and Khandakar (2008). In contrast to Auto-ARIMA, Auto-ETS is a state-space model with exponential smoothing. The algorithm picks the best fitting ETS-Model and tunes the parameters by evaluating the AIC for each. Here, ETS refers to three components: error, trend, and seasonality. These components can either be multiplicative, damped multiplicative, additive, damped additive or not included. The general model can be described by the following state space equations:

$$y_t = w(x_{t-1}) + r(x_{t-1})\epsilon_t$$
$$x_t = f(x_{t-1}) + g(x_{t-1})\epsilon_t$$

whereas x_t is a state vector, ϵ_t is a Gaussian white noise process, and the mean $\mu = w(x_{t-1})$. For the model with additive errors, $r(x_{t-1}) = 1$, so that $y_t = \mu_t + \epsilon_t$. For the model with multiplicative errors, $r(x_{t-1}) = \mu_t$, so that $y_t = \mu_t(1 + \epsilon_t)$. Thus, $\epsilon_t = (y_t - \mu_t)/\mu_t$. The specific form of the functions $w(x_{t-1})$, $f(x_{t-1})$, and $g(x_{t-1})$ depends on the chosen ETS model. Let's break down the components:

- Error (E): Can be additive (A) or multiplicative (M).
- Trend (T): Can be none (N), additive (A), additive damped (Ad), multiplicative (M), or multiplicative damped (Md).
- Seasonality (S): Can be none (N), additive (A), or multiplicative (M).

For example, an ETS(A,A,N) model (additive error, additive trend, no seasonality) would have the following form:

$$y_t = l_{t-1} + b_{t-1} + \epsilon_t$$
$$l_t = l_{t-1} + b_{t-1} + \alpha\epsilon_t$$
$$b_t = b_{t-1} + \beta\epsilon_t$$

Where l_t is the level at time t, b_t is the trend at time t and α and β are smoothing parameters.

For a more complex model like ETS(M,Ad,M) (multiplicative error, additive damped trend, multiplicative seasonality), the equations would be:

$$y_t = (l_{t-1} + \phi b_{t-1}) s_{t-m}(1 + \epsilon_t)$$

$$l_t = (l_{t-1} + \phi b_{t-1})(1 + \alpha \epsilon_t)$$

$$b_t = \phi b_{t-1} + \beta (l_{t-1} + \phi b_{t-1}) \epsilon_t$$

$$s_t = s_{t-m}(1 + \gamma \epsilon_t)$$

Where s_t is the seasonal component at time t, m is the number of seasons per year, ϕ is the damping parameter ($0 < \phi < 1$), α, β, and γ are smoothing parameters.

The Auto-ETS algorithm evaluates all possible combinations of E, T, and S components, fitting each model and calculating its AIC. The model with the lowest AIC is selected as the best fit. This approach allows for automatic model selection and parameter tuning, making it particularly useful for time series forecasting tasks where the underlying patterns may not be known a priori (Hyndman and Athanasopoulos 2018). The flexibility of ETS models in handling different combinations of error, trend, and seasonal components makes them suitable for a wide range of time series data.

6.7 NPTS

Non-Parametric Time Series Forecaster. (Sundar Rangapuram et al. 2023) Like classical forecasting methods, such as exponential smoothing (ETS) and autoregressive integrated moving average (ARIMA), NPTS generates predictions for each time series individually. The time series in the dataset can have different lengths. NPTS is particularly adept at handling time series of varying lengths within a dataset allowing for different sized training and prediction ranges (Amazon Web Services, Inc. 2024). One of NPTS key advantages is its handling of sparse time series. Amazon Forecast NPTS forecasters have the following variants: NPTS, seasonal NPTS, climatological forecaster, and seasonal climatological forecaster The seasonal NPTS variant samples only from observations in past seasons, while the climatological forecaster samples all past observations with uniform probability (Amazon Web Services, Inc. 2024). The weighting scheme can be expressed mathematically as follows:

$$\begin{cases} exp(-\sum_{i=1}^{D} \alpha |f_i(t) - f_i(T)|) \text{unweighted} \\ exp(-\sum_{i=1}^{D} \alpha |f_i(t) - f_i(T)|) \text{weighted} \end{cases}$$

where $q_T(t)$ represents the sampling probability, $f_i(t)$ is the i-th feature at time t, α and α_i are weighting parameters, and D is the number of features. NPTS's principal advantage is its non-parametric handling time series, allowing practitioners to choose the most appropriate model based on the characteristics of their dataset.

6.8 Dynamic Optimized Theta

Dynamic Optimized Theta (DOTM) is an extension to the classic theta model, proposed by (Fiorucci et al. 2016). Instead of assuming the parameters to be stationary, the model dynamically tunes the parameters for each time-step to find the best possible combination of parameters. Like the original model, DOTM relays on Theta-lines. The state space model for the DOTM can be described as followed:

$$y_t = \mu_t + \epsilon_t$$

$$\mu_t = l_{t-1} + (1 - \frac{1}{\Theta})[(1-\alpha)^{t-1}A_{t-1} + (\frac{1-(1-\alpha)^t}{\alpha})B_{t-1}]$$

where y_t is the observed value at time t, μ_t is the local level at time t, ϵ_t is the error term, Θ is the theta coefficient and α is a smoothing parameter. The level parameter is recursively updated by:

$$l_t = \alpha y_t + (1-\alpha)l_{t-1}$$

The minimum square coefficients for each time step A and B are calculated:

$$A_t = \overline{y_t} - \frac{t+1}{2}B_t$$

$$B_t = \frac{1}{t+1}[(t-2)B_{t-1} + \frac{6}{t}(y_t - \overline{y}_{t-1})]$$

$$\overline{y_t} = \frac{1}{t}[(t-1)\overline{y}_{t-1} + y_t]$$

where $\overline{y_t}$ represents the running mean of the series. The key advantage is the automatic choice of optimal parameters. Like other state-space models, the DOTM can be described by a system of equations:

$$Y_t = wZ_t(\theta_1) + (1-w)Z_t(\theta_2)$$

whereas Y_t is the prediction for time t, Z_t represents the theta line depending on the theta coefficient θ and w is a weighting factor. In other words, our prediction is a weighted combination of two theta lines (Vassilis Assimakopoulos and Nikolopoulos 2000).

6.9 DeepAR

DeepAR is an autoregressive recurrent neural network model for probabilistic forecasting, introduced Salinas et al. (2020). While it shares similarities with the LSTM-VAE in its use of LSTM layers and an encoder-decoder architecture, DeepAR is able to directly estimate the parameters of the conditional probability distribution $P(z_t|z_{1:t-1}, x_{1:t})$, where z_t represents the target at time t, and x_t denotes the vector of time-dependent covariates. Unlike the LSTM-VAE, which relies on sampling from a latent space, DeepAR directly learns the parameters directly. This direct estimation of distribution parameters enhances the interpretability of the model's outputs and its performance (Salinas et al. 2020).

A key component of DeepAR is its utilization of a context vector, which serves as a compact representation of the time series' history. This vector is updated recursively through the network's hidden state, enabling the model to capture and leverage long-term dependencies in the data. The model's likelihood function for a single time series is expressed as:

$$L(\theta) = \prod_{t=1}^{T} p(z_t|z_{1:t-1}, x_{1:t}; \theta)$$

where θ represents the parameters of the neural network. During training, the negative log likelihood across all-time series in the data is minimized.

DeepAR's architecture facilitates learning of global patterns across multiple related time series, allowing for effective knowledge transfer to new or sparse series (Salinas et al. 2020) This capability, combined with its direct parameter estimation and use of the context vector, makes DeepAR particularly adept at handling complex forecasting scenarios, including those involving multiple related time series or predictions for new entities with limited historical data. DeepAR has been successfully applied to various problems such as the prediction of landslides in Dong et al. (2021) or biochemical applications such as in Schaduangrat et al. (2023).

6.10 TemporalFusionTransformer

The Temporal Fusion Transformer (TFT) is transformer model proposed by Lin et al. (2020). It combines a transformer architecture with LSTM layers, allowing for efficient multi-horizon forecasting. Another key advantage is its interpretability in contrast to other deep learning models (Lin et al. 2020) The architecture model consists of LSTM layers followed by self-attention layers to capture long term dependencies. This part of the model captures and

encodes the observations. This is followed by a feature extraction layer and gating mechanism to control information flow. The authors demonstrated significant performance improvements over existing benchmarks on various real-world datasets. In summary, the TFT model combines the strengths of transformers (for long-range dependencies) with those of LSTMs (for local processing), making it particularly well-suited for complex multi-horizon forecasting tasks.

6.11 PatchTST

PatchTST is a popular transformer-based model for time series forecasting designed by Nie et al. (2022). It is an improvement to previous transformer models for time series prediction such as LogTrans, Informer and FEDformer, by incorporating patch-based input tokenization. While previous approaches encode each time step as an own token, patching tokenizes over multiple time steps. These patches can overlap and form a sequence of individual patches (Nie et al. 2022). Thus, PatchTST can capture local, comprehensive semantic information by aggregating time steps into local subsets. This approach has the advantage, that relationships between time steps are considered. As a result, trends and seasonality can be better captured. The patches are mapped to the transformer's latent space. The architecture of the model is described in figure 4.

6.12 Chronos [base]

Chronos a state-of-the-art large language model (LLM) for time series forecasting proposed by Ansari et al. (2024). In contrast to the other models discussed in this section, Chronos can make predictions on data not seen before, making it a so called zero-shot model. To achieved this, the model architecture is based on the T5-family (Text-to-Text Transfer Transformer) proposed in Raffel et al. (2020). The T5 model concept is popular for language processing to solve text-to-text tasks such as translation or text classification. In general, the model encodes natural language, tokenizes the input stream and predicts the next token. Chronos follows the same concept, however instead of natural language, predicts the tokens for the next time series step.

On the high level, Chronos can be described by the following steps. First, time series data is scaled and quantized to generate a tokenized representation of the data. Secondly, the model is trained to create prediction probabilities for the next token. Third, the model samples prediction tokens from

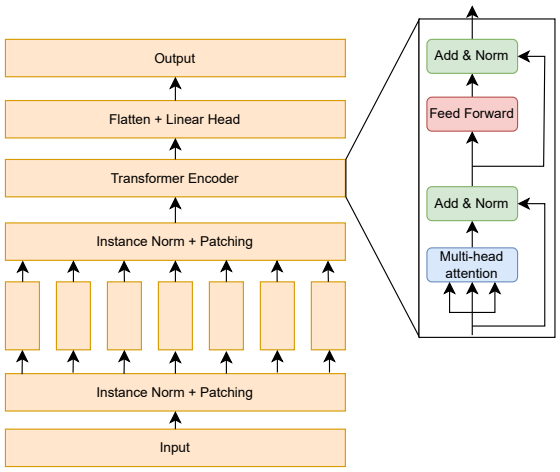

Figure 4: Architecture of PatchTST
The input in form of a univariate time series is normalized and divided into localized subsets, known as patches.

the probability distribution and the context tokens of the input time series. (Ansari et al. 2024) What makes Chronos a zero-shot model is the fact that the second step, the training is already done before using the model. Chronos pre-trained on a large collection on time series data, allowing the model to learn general purpose representations of the time-series data via transfer learning. As we discussed before, time series data composes of three components: a trend-cycle component, a seasonal component, and a remainder component. Consequently, Chronos can a general representation of these components. In short, Chronos is a pre-trained general purpose transfer learning model, tuned for time series data components. For this study, the base version of Chronos with 200M parameters is used.

6.13 Measurement Performance

For the measurement of the accuracy of the models, different methods for evaluation are employed.

6.13.1 Scaled Quantile Loss

To examine the accuracy of the predicted intervals, the Scaled Quantile Loss (SQL) is employed. The Scaled Quantile Loss, also known as Scaled Pinball

Loss, is a normalized quantile-based method to compare results from different time series, even of different scales (Makridakis, Spiliotis, and Vassilios Assimakopoulos 2022). The SQL is defined as follows:

$$\text{SPL}_i(u) = \frac{1}{h} \frac{(y_t - q_t(u))u\mathbf{1}\{q_t(u) \leq y_t\} + (q_t - y_t)(1-u)\mathbf{1}\{q_t(u) > y_t\}}{\frac{1}{n-1}\sum_{t=2}^{n} |y_t - y_{t-1}}$$

Where i is the index of the time series, u is the quantile level (e.g., 0.5 for median, 0.025 and 0.975 for 95% prediction interval). Moreover h is the forecast horizon, y_t is the actual value at time t, $q_t(u)$ is the predicted u-quantile at time t and $\mathbf{1}\cdot$ is the indicator function. The lower the loss, the more accurate the quantile forecast. In our case, a value of 1 suggests that the forecast performs similarly to a seasonal naive forecast that always predicts no change. Consequently, values below 1 indicate improvement over the naive forecast, whereas values above 1 suggest the forecast is worse than the benchmark model. We measure the accuracy of the probabilistic (quantile) forecasts by reporting the mean scaled quantile loss averaged over 9 quantile levels q. (Shchur et al. 2023).

6.13.2 UMAP

To evaluate the LSTM-VAEs ability to accurately learn the distribution of the data, Unifold Manifold Approximation and Projection (UMAP) is employed. If UMAP can distinguish the original data from the generated data, the distribution might not be learned sufficient. UMAP is a popular graph based manifold learning technique for dimension reduction developed by McInnes, Healy, and Melville (2018). In contrast to other graph-based dimension reduction techniques, UMAP can accurately preserve almost the whole local structures while better incorporating global structures, that its competitors tSNE, fit-SNE and LargeVis (McInnes, Healy, and Melville 2018). In order to better understand UMAP, it can be divided into three steps:

- 1. estimating the manifold of the high dimensional data and constructing a weighed graph of the data
- 2. constructing a graph based on the known low dimensional manifold
- 3. Minimize the difference between the high-dimensional graph and the low dimensional graph by fitting the latter.

For these steps three assumptions are required:

- 1. The data is uniformly distributed across a Riemannian manifold.
- 2. The Riemannian metric is approximate locally constant
- 3. The manifold is locally connected

From these three assumptions it is possible to construct a fuzzy topological structure of the data points in the high dimensional original space By creating a cover around each point, the assumptions ensures that each point has at least one overlapping cover. This is possible since the locally constant Riemannian metric ensures the shortest length between two points on the manifold is constant. From sets of these overlapping covers, abstract simplicial complexes can be constructed.

Following the Nerve theorem, a property of the abstract simplicial complex is that any subset of nonempty sets with nontrivial common intersections is part of the simplicial complex. This allows the recovery of the important topology of the original space by constructing a graph from abstract simplicial complexes. By creating customized distances for each nearest neighbor, the distribution of data on the Riemannian manifold is not inherited from the ambient manifold. This has the advantage that the data in the original space dose not has to follow a specific distribution. In order to construct the high dimensional graph, UMAP employs an exponential probability distribution:

$$v_{i|j} = exp[(-d(x_i, x_j) - \rho_i/\sigma_i]$$

whereas $d(x_i, x_j)$ are the distance between the points x_i and x_j and ρ_i is the distance from x_i to the nearest neighbor. If the weights of the graph between x_i and x_j and the weights of the graph between x_j and x_i are not equal, the probabilistic t-conorm operation is employed:

$$v_{ij} = (v_{j|i} + v_{i|j}) - v_{j|i}v_{i|j}$$

with v_{ij}, the probability of the union of the independent sets x_i and x_j. Recalling the previously discussed intersections for the construction of simplicial complexes, v_{ij} can be interpreted as the likelihood of a common membership of a local fuzzy simplex set. Now that we have the method for constructing the high dimensional representation, the low dimensional representation can be constructed. For this UMAP employs a cross entropy of the two fuzzy sets v and w:

$$C(v, w) \widehat{=} \sum_{i \neq j} v_{ij} log\left(\frac{v_{ij}}{w_{ij}}\right) + (1 - v_{ij}) log\left(\frac{1 - v_{ij}}{1 - w_{ij}}\right)$$

whereas v_{ij} represents the high dimensional similarity and w_{ij} the low dimensional similarity. Here, v_{ij} is fixed and this cost function is minimized by modifying w_{ij}. The low dimensional similarly w_{ij} is defined as:

$$w_{ij} = (1 + a\|y_i - y_j\|_2^{2b})^{-1}$$

here the variables a and b are the hyperparameters, set by the user. As a result, UMAP creates a low dimensional graph, as similar as possible to the high dimensional graph.

7 Results

Model	Test Score	Validation Score
AutoETS	0.871127	1.981468
Dynamic Optimized Theta	0.892631	2.073290
Temporal Fusion Transformer	1.028134	2.228520
DeepAR	1.083137	2.464062
NPTS	1.086930	2.429495
WeightedEnsemble	1.109582	1.950413
Seasonal Naive	1.122997	1.950625
Auto ARIMA	1.351005	2.234008
PatchTST	1.354599	3.264152

Table 1: Results
The table presents the results of the models, sorted by test score. Both, the Test Score and the Validation Score are calculated via the SQL. The AutoETS Model performed best, followed by the Dynamic Optimized Theta and the Temporal Fusion transformer.

7.1 Results GARCH and ARMA-GARCH

We first evaluated the performance of GARCH and ARMA-GARCH models. The hyperparameter sets for both models were determined through grid search, leading to the selection of GARCH(2,1) and ARMA-GARCH(2,2) as the best-fitting parameter combinations. A summary of the ARMA-GARCH model is presented in table 1. While the adjusted R-squared of 0.937 for the ARMA-GARCH suggests a decent fit, the validation of both models indicated their unsuitability for short-term forex prediction. The autocorrelation

function (ACF) plots for both models, illustrated in figure 5, reveal significant autocorrelation of the residuals, thereby violating the independent and identically distributed (i.i.d.) assumption of the error terms. Although both models exhibit correlated residuals, the ARMA-GARCH model demonstrates less correlation.

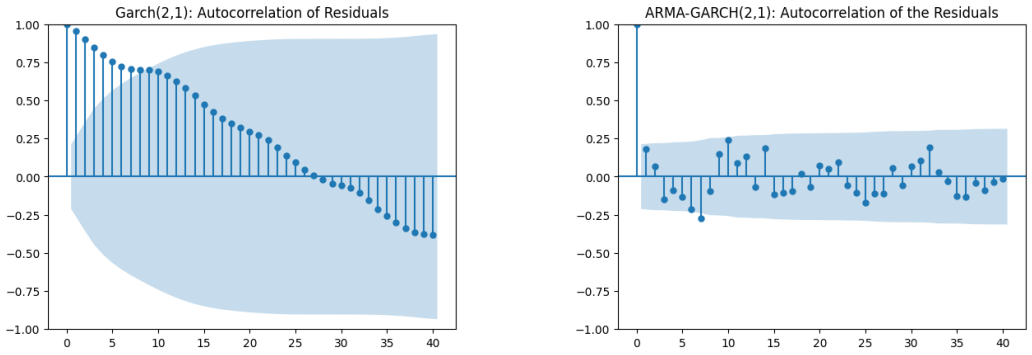

Figure 5: ACF Plots of the GARCH and ARMA-GARCH.
The residuals of the GARCH on the left are highly correlated. The residuals of the ARMA-GARCH are less correlated, but still follow trends. Overall, the residuals of both models are correlated, hence violate the i.i.d. assumption residuals.

Since the GARCH and ARMA-GARCH models have correlated standardized residuals, both appear to have failed to capture the entire auto correlation in the data. Additionally, the Quantile-Quantile plots in figure 6 indicate that the residuals are not normally distributed. This observation is further supported by the results of the Jarque-Bera test, which yielded p-values of 0.0151 for GARCH and 0.0271 for ARMA-GARCH, both below the conventional significance level of 0.05. In general, the validation indicates that both models are not able to fully capture the autocorrelation in the data.

As mentioned above, the ARMA-GARCH performed better than the vanilla GARCH but both models did not perform well enough. Nevertheless, we still can gain insights from these results. The parameter of the mean model is 1.0090, which implies that the mean is not completely stationary at each time step of the observed period. While a divergence of 0.009 may appear small, it may have a noticeable impact on the mean, forming a positive trend for each time step. This observation could violate or assumption of mean-reverting weak stational processes in the data.

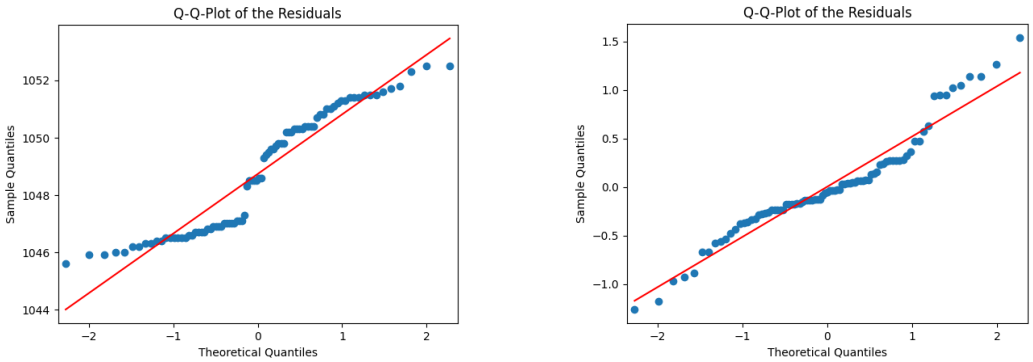

Figure 6: Quantile-Quantile Plots of the GARCH and ARMA-GARCH

Dep. Variable:	Close			
Mean Model:	AR			
Vol Model:	GARCH			
Method:	Maximum Likelihood			
Statistic	**Value**	**Std. Err.**	**t-value**	**P <|t |**
R-squared:	0.938			
Adj. R-squared:	0.937			
Log-Likelihood:	-54.9187			
AIC:	121.837			
BIC:	136.493			
No. Observations:	85			
Df Residuals:	83			
Df Model:	2			
Mean Model Parameters				
Const	-9.3388	0.271	-34.475	1.908e-260
Close[1]	1.0090	0.0003267	3088.218	0.000
Volatility Model Parameters				
omega	0.0129	0.008909	1.449	0.147
alpha[1]	0.0077	0.123	0.063	0.950
alpha[2]	0.1720	0.164	1.047	0.295
beta[1]	0.7838	0.06611	11.857	1.983e-32

Table 2: AR-GARCH Model Summary

7.2 Results LSTM-VAE

Next, the LSTM-VAE model is evaluated. Its properties allow us to quickly gain insights into the feasibility of constructing distributions from sequential forex data, in addition to forecasting. The training curves are presented in figure 7. Our results indicate, that the best ELBO loss we obtained is 45.1323

in Epoch 299. Since this loss is not directly interpreted, we can use UMAP to give us further insights on the quality of the reconstructed data. The main idea of this approach is, if UMAP cannot distinguish the reconstructed data from the original data, both the construction of the latent distribution and the reconstruction of the sample in the LSTM-VAE are sufficient.

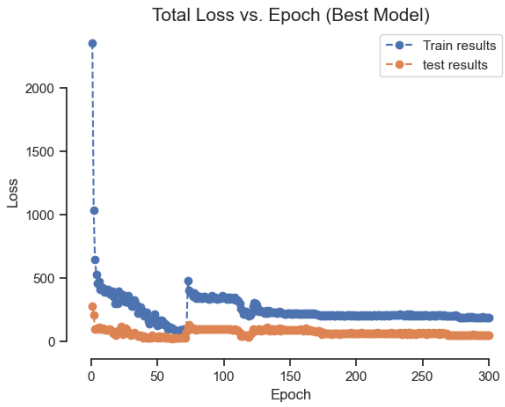

Figure 7: Loss Curves of the best LSTM-VAE Model
The training loss decreases rapidly and behaves stable. Counter-intuitively, the test-results are below the train results. However, this is due the initiation of the data loader and the batch size.

As shown in figure 8, the original data and the output from the LSTM-VAE are distinguishable. While there are overlapping points, most of the original data is separated from the reconstructed data. Since the Kulbach-Leibler divergence part of the ELBO loss function can be interpreted as the reconstruction loss of decoder, we can use this metric to gain further insights of the quality of approximation. The final Kulbach-Leibler divergence of 0.003 indicates that the LSTM-VAE model can accurately reconstruct the learned distribution, but it did not learn the original data distribution well enough to accurately reconstruct the data. While there are architectural improvements such as dropout layers or multi-attention layers available, we consider inference with the LSTM-VAE model not promising under the current circumstances.

7.3 Results Autogluon Models

The results of the autogluon models are described in this section. Besides the models described in section 6, a weighted ensemble of all models presented in table 1 is calculated. Based on the validation score, AutoETS performed

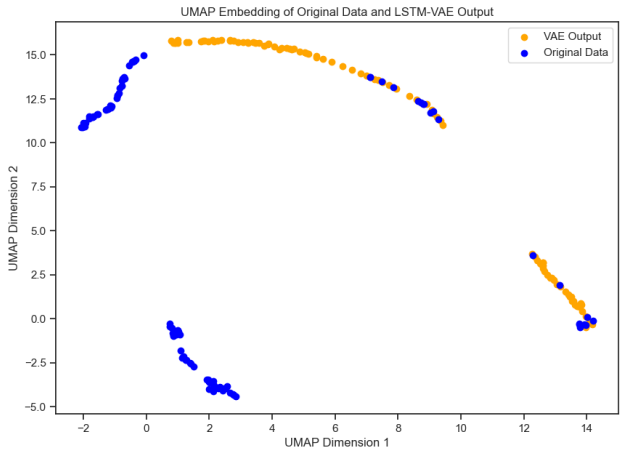

Figure 8: UMAP Plot Original vs. Output
The UMAP embedding of the original sequence and the model reconstruction is plotted. The original data observations are blue, whereas the reconstructed are orange. The figure indicates that the LSTM-VAE did not achieve a sufficient reconstruction of the original distribution. While some points overlap, original and reconstructed data are visually distinguishable from each other.

best, followed by Dynamic Optimized Theta and the Temporal Fusion Transformer model. In contrast, the Weighted Ensemble, which performed best in validation, performed seventh best overall.

In total, all the test scores of all models are relatively close to 1, which implies that the performance of the models is similar to a naive model. Only Auto-ETS (0.8711) and Dynamic Optimized Theta (0.8929) managed to beat the naive model but yield only small improvement. The evaluation of the deep learning models revealed significant overfitting, indicating their inability to generalize effectively to unseen data. DeepAR, TFT, and PatchTST predominantly attempted to predict by mimicking the chart movements from the previous day. This focus on reproducing short-term patterns suggests that the models were overly focused on the short term and failed to capture longer-term dependencies, implying limited consideration of historical data. As mentioned earlier, we assume up to 5 days of mean-reverting stationary behavior, with distributional properties that should hold within this period. The performance of the models, however, suggests that they were unable to capture this mean-reverting behavior. Their predictions resembled a naïve approach, using daily time steps, rather than identifying more complex patterns, such as mean reversion.

In addition to these technical limitations, we cannot rule out that mean-

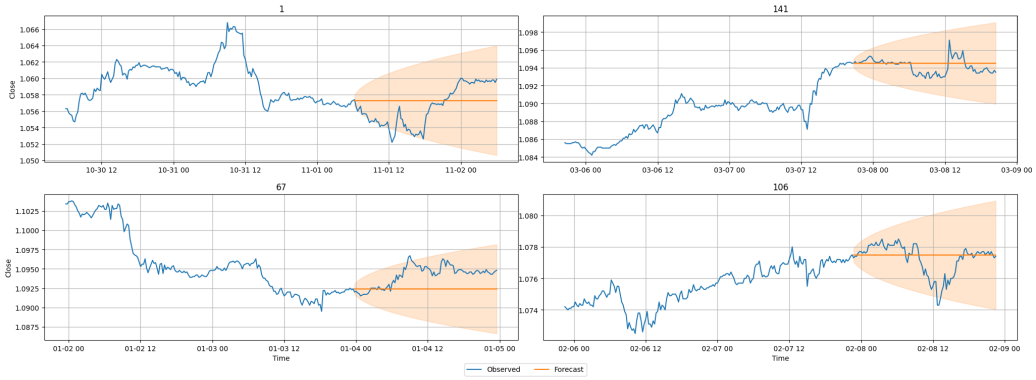

Figure 9: AutoETS Plot
The plots of the AutoETS model show the 90% prediction interval over the real data.

reverting behavior does not exist within the ultra-short-term observations. The ARMA-GARCH model also indicated a lack of mean reversion, but these results should be interpreted with caution due to violations of the model's assumptions. Moreover, this does not explain the performance of AutoETS and Dynamic Optimized Theta. Lastly, we tested the Chronos model. The model failed to produce meaningful forecasts. This was expected since it is a pre-trained single shot model and as such it is unlikely to accurately predict very complex time series. The plots for the deep learning models and the Chronos are provided in the repository of this study. In summary, only AutoETS and Dynamic Optimized Theta performed better than a naive model, while the deep learning models failed to generalize.

8 Discussion

The results of this study indicate that the tested models offer substantial room for improvement in short-term forex market forecasting. While the behavior of the deep learning models suggests random-walk characteristics, the performance of AutoETS and Dynamic Optimized Theta implies the presence of learnable distributional patterns. This aligns with the provided anecdotal evidence but necessitates further investigation given the overall low performance of the models.

A key factor likely contributing to the weak performance of the tested models is the impact of economic surprises. These announcements significantly influence exchange rates and often cause breaks in the assumed sta-

tionarity of the time series. Despite scaling efforts, our models seem unable to adequately handle these structural breaks. For this reason, several avenues for improvement could be explored.

First, transforming the data could lead to better results (Hyndman and Athanasopoulos 2018). In our study, we considered raw closing rates but transforming them into volatility rates could be beneficial. Additionally, multivariate approaches could yield superior results by capturing inter-market dynamics. Furthermore, the overfitting observed in the deep learning models could result from suboptimal parameter choices. While the overfitting aligns with anecdotal expectations from experts in the field, other parameter sets may result in better model fitting.

More sophisticated hyperparameter tuning strategies and the incorporation of different dropout rates could mitigate overfitting and significantly improve performance. Lastly, conducting qualitative interviews with traders could provide insights into their trading strategies, potentially informing model design and improving feature selection. Although the model classes tested in this study are appropriate for time series forecasting, they appear highly sensitive to hyperparameter settings and data preprocessing techniques.

To address these technical issues, the models could be refined and reevaluated to eliminate technical weaknesses and gain deeper insights into the feasibility of distributional forex modeling in the short term. Although the results are in line with our initial expectations, the evidence suggests that there are distributional properties that can be exploited for trading.

9 Conclusion

This study examined diverse statistical and deep learning models to assess their capacity for making distributional forecasts in the foreign exchange (forex) market over ultra-short-term horizons. Our investigation was motivated by anecdotal evidence from BASF, suggesting that volatility in periods up to five days is not solely driven by random walks, but also by recurring distributional patterns.

Our primary objective was to model the intuition of BASF's traders by developing distributional inference models. We tested various statistical and deep learning architectures, with the Auto-ETS model emerging as the best performer among all tested models. However, it is noteworthy that only two models outperformed the naive model, and even these yielded only marginal improvements.

The results of this study indicate that further research is necessary to

fully evaluate the efficacy of these model classes in ultra-short-term forex forecasting. Several avenues for potential improvement have been identified such as incorporating economic surprises. While our findings align with initial expectations and suggest the presence of exploitable distributional properties in forex trading, the puzzle of market behavior in ultra-short-term horizons remains largely unsolved. The marginal improvements over naive models underscore the complexity of this forecasting task and the need for continued research in this area.

10 Acknowledgements

We would like to thank Christoph Weisser and Josef Lüth for their continuous support and their valuable feedback.

References

Ahmed, Salman et al. (2020). "FLF-LSTM: A novel prediction system using Forex Loss Function". In: *Applied Soft Computing* 97, p. 106780. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2020.106780. URL: https://www.sciencedirect.com/science/article/pii/S1568494620307183.

Amazon Web Services, Inc. (2024). *Non-parametric Time Series (NPTS) Recipes for Amazon Forecast*. Accessed: 2024-09-10. URL: https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-npts.html.

Ansari, Abdul Fatir et al. (2024). *Chronos: Learning the Language of Time Series*. arXiv: 2403.07815 [cs.LG]. URL: https://arxiv.org/abs/2403.07815.

Assimakopoulos, Vassilis and Konstantinos Nikolopoulos (2000). "The theta model: a decomposition approach to forecasting". In: *International journal of forecasting* 16.4, pp. 521–530.

BASF (2022). *The BASF Group*. Accessed: [01.08.2024]. BASF. URL: https://www.basf.com/cn/en/media/GC-report/GC-report-2022/the-basf-group.

Bollerslev, Tim (1987). "A conditionally heteroskedastic time series model for speculative prices and rates of return". In: *The review of economics and statistics*, pp. 542–547.

Ca' Zorzi, Michele, Jakub Muck, and Michal Rubaszek (Jan. 2016). "Real Exchange Rate Forecasting and PPP: This Time the Random Walk Loses". In: *Open Economies Review* 27.3, pp. 585–609. ISSN: 1573-708X. DOI: 10.1007/s11079-015-9386-4. URL: http://dx.doi.org/10.1007/s11079-015-9386-4.

Chong, Choo Wei, Muhammad Idrees Ahmad, and Mat Yusoff Abdullah (1999). "Performance of GARCH models in forecasting stock market volatility". In: *Journal of forecasting* 18.5, pp. 333–343.

Dong, Mei et al. (2021). "Deformation Prediction of Unstable Slopes Based on Real-Time Monitoring and DeepAR Model". In: *Sensors* 21.1. ISSN: 1424-8220. DOI: 10.3390/s21010014. URL: https://www.mdpi.com/1424-8220/21/1/14.

Fama, Eugene F and Kenneth R French (1988). "Permanent and temporary components of stock prices". In: *Journal of political Economy* 96.2, pp. 246–273.

Fiorucci, Jose A et al. (2016). "Models for optimising the theta method and their relationship to state space models". In: *International journal of forecasting* 32.4, pp. 1151–1161.

Gaglianono, Wagener Piazza and Luiz Renato Lima (2012). "Constructing Density Forecasts from Quantile Regressions". In: *Journal of Money, Credit and Banking* 44.8, pp. 1589–1607.

Gers, Felix A, Jürgen Schmidhuber, and Fred Cummins (2000). "Learning to forget: Continual prediction with LSTM". In: *Neural computation* 12.10, pp. 2451–2471.

Gneiting, Tilmann and Matthias Katzfuss (2014). "Probabilistic forecasting". In: *Annual Review of Statistics and Its Application* 1.1, pp. 125–151.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts.

Hyndman, Rob J and Yeasmin Khandakar (2008). "Automatic time series forecasting: the forecast package for R". In: *Journal of statistical software* 27, pp. 1–22.

Islam, Md. Saiful et al. (2020). "A Review on Recent Advancements in FOREX Currency Prediction". In: *Algorithms* 13.8. ISSN: 1999-4893. DOI: 10.3390/a13080186. URL: https://www.mdpi.com/1999-4893/13/8/186.

Khan, Maaz et al. (2023). "COVID-19 pandemic & financial market volatility; evidence from GARCH models". In: *Journal of Risk and Financial Management* 16.1, p. 50.

Kingma, Diederik P (2013). "Auto-Encoding Variational Bayes". In: *arXiv preprint arXiv:1312.6114*.

Kingma, Diederik P, Max Welling, et al. (2019). "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4, pp. 307–392.

Li, Long et al. (2019). "A method to get a more stationary process and its application in finance with high-frequency data of Chinese index futures". In: *Physica A: Statistical Mechanics and its Applications* 525, pp. 1405–1417. ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2019.04.085. URL: https://www.sciencedirect.com/science/article/pii/S0378437119304509.

Lin, Shuyu et al. (2020). "Anomaly detection for time series using vae-lstm hybrid model". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Ieee, pp. 4322–4326.

Makridakis, Spyros and Michele Hibon (1997). "ARMA Models and the Box–Jenkins Methodology". In: *Journal of Forecasting* 16.3, pp. 147–

163. DOI: https://doi.org/10.1002/(SICI)1099-131X(199705)16:3<147::AID-FOR652>3.0.CO;2-X. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291099-131X%28199705%2916%3A3%3C147%3A%3AAID-FOR652%3E3.0.CO%3B2-X. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291099-131X%28199705%2916%3A3%3C147%3A%3AAID-FOR652%3E3.0.CO%3B2-X.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2022). "M5 accuracy competition: Results, findings, and conclusions". In: *International Journal of Forecasting* 38.4, pp. 1346–1364.

McInnes, Leland, John Healy, and James Melville (2018). "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426*.

Meese, Richard A and Kenneth Rogoff (1983). "Empirical exchange rate models of the seventies: Do they fit out of sample?" In: *Journal of international economics* 14.1-2, pp. 3–24.

Menkhoff, Lukas (2010). "The use of technical analysis by fund managers: International evidence". In: *Journal of Banking & Finance* 34.11, pp. 2573–2586.

Nie, Yuqi et al. (2022). "A time series is worth 64 words: Long-term forecasting with transformers". In: *arXiv preprint arXiv:2211.14730*.

Niu, Zijian, Ke Yu, and Xiaofei Wu (2020). "LSTM-Based VAE-GAN for Time-Series Anomaly Detection". In: *Sensors* 20.13. ISSN: 1424-8220. DOI: 10.3390/s20133738. URL: https://www.mdpi.com/1424-8220/20/13/3738.

Poon, Ser-Huang and Clive W J Granger (2003). "Forecasting volatility in financial markets: A review". In: *Journal of economic literature* 41.2, pp. 478–539.

Poterba, James M and Lawrence H Summers (1988). "Mean reversion in stock prices: Evidence and implications". In: *Journal of financial economics* 22.1, pp. 27–59.

Raffel, Colin et al. (2020). "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140, pp. 1–67.

Roussi, Rani (2019). *yfinance: Yahoo! Finance market data downloader*. Accessed: 2024-09-05. URL: https://github.com/ranaroussi/yfinance.

Salinas, David et al. (2020). "DeepAR: Probabilistic forecasting with autoregressive recurrent networks". In: *International Journal of Forecasting* 36.3, pp. 1181–1191. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2019.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0169207019301888.

Schaduangrat, Nalini et al. (2023). "DeepAR: a novel deep learning-based hybrid framework for the interpretable prediction of androgen receptor antagonists". In: *Journal of Cheminformatics* 15.1, p. 50.

Shchur, Oleksandr et al. (2023). *AutoGluon-TimeSeries: AutoML for Probabilistic Time Series Forecasting*. arXiv: 2308.05566 [cs.LG]. URL: https://arxiv.org/abs/2308.05566.

Sundar Rangapuram, Syama et al. (2023). "Deep Non-Parametric Time Series Forecaster". In: *arXiv e-prints*, arXiv–2312.

Svetunkov, Ivan and Fotios Petropoulos (2018). "Old dog, new tricks: a modelling view of simple moving averages". In: *International Journal of Production Research* 56.18, pp. 6034–6047. DOI: 10.1080/00207543.2017.1380326.

Wang, Heshan et al. (2023). "DAFA-BiLSTM: Deep Autoregression Feature Augmented Bidirectional LSTM network for time series prediction". In: *Neural Networks* 157, pp. 240–256. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2022.10.009. URL: https://www.sciencedirect.com/science/article/pii/S0893608022003938.