forecasting

MDPI

Article

Data-Centric Benchmarking of Neural Network Architectures for the Univariate Time Series Forecasting Task

Philipp Schlieper [1,*], Mischa Dombrowski [1], An Nguyen [1], Dario Zanca [1] and Bjoern Eskofier [1,2]

[1] Department Artificial Intelligence in Biomedical Engineering, Friedrich-Alexander-University, 91052 Erlangen, Germany; mischa.dombrowski@fau.de (M.D.); an.nguyen@fau.de (A.N.); dario.zanca@fau.de (D.Z.); bjoern.eskofier@fau.de (B.E.)

[2] Institute of AI for Health, Helmholtz Center Munich German Research Center for Environmental Health, 85764 Neuherberg, Germany

* Correspondence: philipp.schlieper@fau.de

Abstract: Time series forecasting has witnessed a rapid proliferation of novel neural network approaches in recent times. However, performances in terms of benchmarking results are generally not consistent, and it is complicated to determine in which cases one approach fits better than another. Therefore, we propose adopting a data-centric perspective for benchmarking neural network architectures on time series forecasting by generating ad hoc synthetic datasets. In particular, we combine sinusoidal functions to synthesize univariate time series data for multi-input-multi-output prediction tasks. We compare the most popular architectures for time series, namely long short-term memory (LSTM) networks, convolutional neural networks (CNNs), and transformers, and directly connect their performance with different controlled data characteristics, such as the sequence length, noise and frequency, and delay length. Our findings suggest that transformers are the best architecture for dealing with different delay lengths. In contrast, for different noise and frequency levels and different sequence lengths, LSTM is the best-performing architecture by a significant amount. Based on our insights, we derive recommendations which allow machine learning (ML) practitioners to decide which architecture to apply, given the dataset's characteristics.

Keywords: deep learning; time series; neural networks; model selection; data synthesis; univariate forecasting

Citation: Schlieper, P.; Dombrowski, M.; Nguyen, A.; Zanca, D.; Eskofier, B. Data-Centric Benchmarking of Neural Network Architectures for the Univariate Time Series Forecasting Task. *Forecasting* **2024**, *6*, 718–747. https://doi.org/10.3390/forecast6030037

Academic Editor: Daniele Apiletti

Received: 22 July 2024
Revised: 15 August 2024
Accepted: 22 August 2024
Published: 26 August 2024

1. Introduction

Aside from computer vision and natural language processing, time series analysis is one of the core domains for applications of neural network architectures. In particular, forecasting univariate time series is a predominant task in various fields like finance and economics, healthcare, environmental analysis, and predictive maintenance [1].

To predict future states of data which have been recorded over time, scientists have implemented many derivations of common models like feedforward networks, recurrent networks, and convolutional networks [2]. However, new approaches are presented at a high pace in the time series analysis literature. As a result, even new neural network paradigms, such as the transformer, which originally emerged in the natural language processing domain [3], are introduced and applied to time series [4]. Usually, the latest model variants will produce improved state-of-the-art performances which are benchmarked on synthetic and open-source datasets. Given the fast developments in the community, machine learning practitioners can have difficulties choosing the suitable model or architecture for their prediction task in use cases with individual real-world datasets.

What is more, when freshly published model variants are applied to real-world use cases, respective performances may vary from the reported results in the literature. Thus, we hypothesize that it is difficult for machine learning users to judge published performances regarding their impact on a user's individual data. One explanation

might be that new approaches are usually tested using more or less subjectively chosen datasets. Despite there being some familiar sources for benchmarking datasets, such as the UCI Machine Learning Repository (accessible at http://archive.ics.uci.edu/, accessed on 5 September 2023), in the least amount of cases, a real-world dataset shares its set of characteristics with one of the benchmarking sets which was used for model evaluation. A dataset from a real-world use case might present different long-term dependencies, have a unique set of frequency and noise features, or might present large sequences. This is why, for example, a freshly proposed version of the transformer architecture can be declared as producing new state-of-the-art performance. Still, when applied to real-world data, the results are diminished, or other models perform even better.

In this work, we switched from a model-centric perspective to a data-centric one to compare different architectural designs of time series models. We use a synthetically generated dataset to take complete control over the above-mentioned characteristics. By synthesizing data as a combination of sinusoidal functions, we can define different levels of the delay length parameter, which controls long- and short-term dependencies. We further control the noise and frequency levels and the input length of the time series sequences. These synthetic data exhibit a more simplistic structure than many real-world datasets. However, this allows us to connect model performances with individual data characteristics. In our experimental set-up, we evaluated the main neural network paradigms for time series analysis (recurrence, convolution, and attention) on datasets where only one of the characteristics varied. Specifically, we restricted the task to univariate time series prediction to improve the results' interpretability and not constrain generality.

The architectures we investigate are the long short-term memory (LSTM) network [5], a convolutional neural network (CNN) model [6], and the transformer model [3]. These three models are designed differently in terms of how they process the temporal information in their internal structures. LSTM utilizes the cell and hidden states inside a recurrent layer to handle time dependencies in the data. At the same time, the CNN applies one-dimensional convolutions along the time axis. As our third paradigm, the transformer employs multiple attention layers, where the whole temporal sequence is put in at once. We used the most basic versions of these models by closely following the model structures of their first presentations in the literature. While more advanced versions and adaptions of these architectures would improve model performance in our experimental task, such as residual connections in CNNs, our intention was to assess the performance of the inner temporal information processing structures of the models. In addition, we performed hyperparameter tuning for parameters like the learning rate and batch size.

For this study, our main emphasis is on neural networks, delving into the internal mechanisms of prevalent architectures and examining their processing of univariate time series data. Classic machine learning (ML) models require a generally lower investment of implementation and training time and can therefore be applied with fewer resources. In contrast, for neural networks, these costs are higher. These implications are even more important when we consider real-world use cases where a limited budget of computational resources and time is often a crucial factor.

The entire content of this article is illustrated in Figure 1. It summarizes our work's main aspects and describes our experimental procedure. How we designed our experiments allowed us to draw conclusions by directly assessing the performance of the architectures with respect to specific characteristics in the data. We bypassed any influence from cherry-picked datasets with our generated data and dismissed any effects of highly specified and tuned model structures. Our main goal is to provide insights into how the processing of temporal information inside LSTM, CNN, and transformer models impacts their performance in the face of data with different long- and short-term memories and frequency levels, as well as different sequence lengths for the scenario of univariate time series prediction.

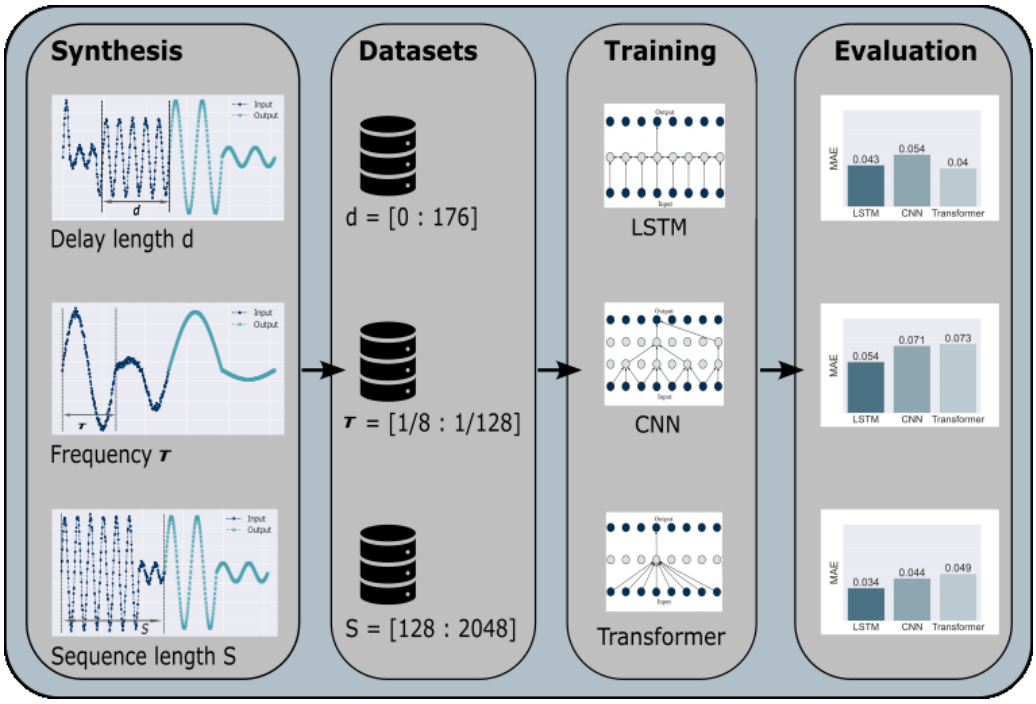

Figure 1. High-level view of the content of this article. We synthesized time series with certain delay lengths, frequencies, and sequence lengths to create datasets where only one of these characteristics varies. For each dataset, we trained a long short-term memory (LSTM) network, convolutional neural network (CNN), and transformer architecture to evaluate the performance concerning the given characteristic.

In this research paper, we introduce three major contributions to the field of time series forecasting. Firstly, we present a framework for synthetically generating time series data which allows for precise control over various data characteristics. This framework is designed to aid practitioners in creating customized datasets tailored to specific research needs. Secondly, we conduct an in-depth analysis of the learning phase and performance of the most common basic neural network architectures for univariate time series forecasting. This analysis provides valuable insights into the strengths and weaknesses of the architectures' different time series processing structures. Lastly, we establish a causal connection between the performances of these architectures and specific characteristics of the data. This connection helps with understanding how different data characteristics influence model effectiveness, offering guidance for selecting appropriate models based on the inherent properties of the time series data.

Our main goal is to develop a deeper intuition on how different neural network architectures—specifically recurrence-based, convolution-based, and attention-based models—process key characteristics in time series data, such as the delay length, frequency and noise, and sequence length. By systematically analyzing these architectures, we aim to uncover how each architecture type handles these distinct properties in the data, providing a clearer understanding of their internal mechanisms and performance. Additionally, we seek to detect any significant differences between these architectures in terms of their ability to manage and learn from these characteristics, thereby offering insights into their suitability for various time series forecasting tasks.

The remainder of this article is structured in the following way. Section 2 gives an overview of related works from the literature, Section 3 provides a description of the data synthesis and the experimental design, with the results and interpretations being included in Section 4, Section 5 discusses the limitations of our approach, and Section 6 summarizes the presented work.

2. Related Works

Numerous application papers offer extensive comparisons between different neural network architectures for specific time series datasets.

One related paper was published by Koprinska et al. [7]. Their goal was to compare the performance of CNNs with other paradigms in terms of electric load and solar power forecasting tasks. They used multilayer perceptron (MLP), LSTM, and persistent models as the baseline. Measured by their mean absolute error (MAE), the CNN and MLP models outperformed the baseline and the LSTM approach. LSTM performed even worse than the baseline. However, this paper includes little analysis of the dataset's characteristics and why MLP models might be better in some cases, and they did not compare it with the transformer approach. This shows that even though the performance of CNNs overall was better, not a single architecture outperformed the other, and even LSTM models were best in some cases. Nassar et al. [8] showed this ambivalence well. This paper analyzed various machine learning and deep learning approaches to weather-to-yield and weather-to-price prediction. Interestingly, LSTM performed better than CNNs in weather-to-yield prediction but worse in weather-to-price prediction. Moreover, both models performed considerably worse than hybrid models which combined all three model paradigms.

Wu et al. [9] compared the transformer model with LSTM in the task of univariate time series forecasting. They used a fixed-length sliding window approach to perform one-step-ahead prediction with the basic transformer architecture and compared it to a baseline LSTM model. The paper showed that the transformer approach can improve performance, suggesting it is suitable as an architecture for time series prediction. However, the results were only supported by a single case study.

These results offer some interesting insights but often say little about the learning capabilities of the architectures involved. Consequently, it remains challenging to understand which architecture will work best on a new dataset, unless this coincides with a new use case.

Furthermore, recent work has attempted to evaluate different neural network architectures for time series more systematically based on a broader selection of public datasets.

Wen et al. [4] published a survey which presented and compareed the latest versions of the transformer architecture for time series prediction tasks. They included in their evaluation the vanilla transformer [3], Autoformer [10], Informer [11], Reformer [12], and LogFormer models [13]. Additionally, they reported the results for a benchmark dataset with respect to robustness, model size, and impact of seasonal trend decomposition. Although Autoformer performed best in the robustness and model size analysis, for the seasonal trend decomposition analysis, the vanilla transformer model showed better results for smaller output lengths. Since the authors only evaluated one dataset, a general conclusion could not be drawn on which transformer variant was the best one. This further supports our approach of utilizing a controllable synthetic dataset rather than a collection of real-world datasets. Thus, we can directly relate architecture performances to specific characteristics in the data.

Agarwal et al. [14] compared CNNs, RNNs, and transformer-based architectures on publicly available datasets. Their goal was to choose these datasets based on a variability in characteristics similar to our approach and select datasets with different lengths, domains, scale variations, and time granularities. In contrast to our synthetic data generation, their real-world data characteristics could not be controlled. They concluded that the transformer model performed well across various problems and excelled on datasets where the number of training points was small. The RNN-based architectures performed well when there were enough training data and only a slight variance in the scale within the data. The CNN-based architectures performed well on time series with many sampling points and not so well if this number was small, although this number varied across different CNN-based architectures. Interestingly, all three paradigms provided the best architecture for at least one dataset. The transformer approach was best on the dataset which had many short time series, the CNN was best on the dataset with only a few long time series in their training

set, and the RNN was best when the dataset had no scale variations. They compared a total of 10 network-based forecasting models which are already advanced versions of the original LSTM, CNN, and transformer architectures. Thus, their reported performances can only be useful when the intent is to use one of these specific models. Since we are benchmarking basic versions of each of the three architectures, our performances can be associated with how these architectures process data.

Hewamalage et al. [15] focused on the recurrent neural network architectures and their application to time series forecasting. By establishing a benchmark and an empirical study, the authors provided guidelines and the best practices for LSTM and related recurrent models. In their results, they showed that this network family can model seasonality directly if the seasons are homogeneous in the dataset. By comparing them against an ETS and ARIMA baseline, the authors concluded that recurrent neural networks, and especially the LSTM variant, are competitive models for time series forecasting. This work is related to our approach to providing insights on model performances in different scenarios and deriving recommendations for their application based on those insights. While Hewamalage et al. [15] assessed recurrent network models, we are interested in comparing the three major network paradigms.

In the work "Benchmarking Attention-Based Interpretability of Deep Learning in Multivariate Time Series Predictions", the authors compared attention-based neural networks on synthetic multivariate time series data [16]. Therefore, they created different datasets with various interactions in a multivariate series. Each model was evaluated regarding their performance score, interpretability correctness, and sensitivity analysis. Their results show that the IMV-LSTM model presented the best performance and interpretability since it could learn both autocorrelations and crosscorrelations. While the perspective of their experiments was, similar to ours, data-centric in employing controlled synthetic data to test for performance differences, the target of the analysis was different. They were investigating interpretability mechanisms, while we are evaluating the processing structure of network architectures.

Other publications presented their takes on data-centric approaches for time series analysis as data engineering to provide more robust model training. Whang et al. [17] provided an overview of the available techniques for engineered data collection and improving its quality. Different approaches were presented which covered various techniques for improving the quality of training data, like data cleaning and sanitization. They concluded that the data-centric perspective would play a major role in future developments in the AI domain. Similarly, Hedge et al. [18] applied a data-centric approach to directly improve model performance and robustness in an anomaly detection scenario. By ensuring the quality of the training data, the model presented better performance than a model-centric baseline. While both papers relate to data-centric research, their directions differ from our take on benchmarking neural network architectures. Nonetheless, Whang et al. [17] acknowledged and described the importance of data-centric approaches for AI-related research.

With Li et al. [13] providing LogFormer to counter the memory bottleneck for time series forecasting, the authors also presented a technique to generate synthetic data based on sinusoidal signals. By altering the amplitudes of the signals, they created variation in the data. This approach is the foundation of our data synthesis method, as described in Section 3.2. For our purposes, we extended data generation to create signals which vary in different parameters and to obtain control over the varying characteristics.

Data-centric AI has become a more established technique across various ML domains. Mazumder et al. [19] introduced DataPerf, a framework for data-centric AI benchmarking which involves vision, speech, acquisition, debugging, and text-to-image prompting. The approach is designed to include future community-based benchmarks as well. By iteratively applying data-centric operations such as augmentation, cleaning, and representation selection, DataPerf creates new train and test sets to enhance the performance of ML models. The benchmark resembles an open-source platform intended to be extended with additional benchmarks in the future. A related approach was proposed by

Devarajan et al. [20] named DLIO, a benchmark aimed at scientific DL applications for improving input and output performance.

The aforementioned papers demonstrate that comparisons centered on models can benefit from a data-centric approach. Although model-centric approaches have been evaluated across a large battery of different datasets, it is difficult to conclude what kinds of data characteristics model-centric approaches are capable of learning and where it is possible to exploit the temporal processing capabilities of certain models. Unlike these works, we propose a data-centric evaluation, where dataset characteristics are tuned continuously to provide a performance spectrum which can serve as a guideline when applying these models to a specific time series problem.

3. Materials and Methods

This section provides a description of our methodology. First, we introduce the theoretical basics of the investigated neural network architectures. After that, the data synthesis process is described, which is used to obtain three datasets with varying characteristics. Next, we describe how we conducted the model training and testing. The following subsections contain explanations of our three experiments where we tested the delay lengths, frequency and noise levels, and sequence lengths.

3.1. Preliminaries

We investigated three learning paradigms of neural networks in our experiments. They all have a different method of modeling temporal dependencies, as shown in Figure 2. One follows the architecture of long short-term memory (LSTM) networks, one is based on convolutional neural networks (CNNs), and one is derived from the transformer architecture. The basics of these architectures are described below.

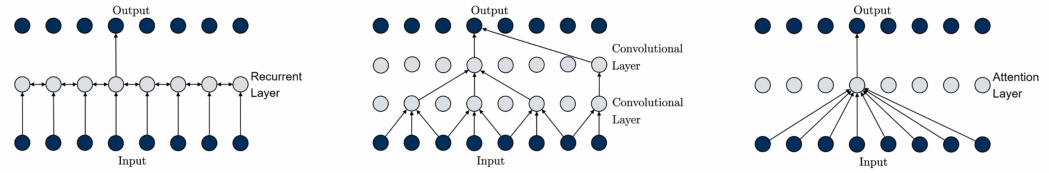

Figure 2. Conceptional comparison of different deep learning paradigms to model time dependencies across input samples: LSTM (**left**), CNNs (**middle**), and attention (**right**). Figure adjusted from [21].

3.1.1. Recurrence-Based Architectures

LSTM models differentiate themselves from feedforward networks by introducing recurrent connections to the model. These edges can create cyclic paths within the model used to learn the dependencies between single time steps. The main idea is that when looking at a single time step t, they use information from the input $x^{(t)}$ of the network and the hidden state from the previous time step $h^{(t-1)}$[5].

In its basic equations (Equations (1) and (2)), the network propagates hidden states into the future, which means that nodes corresponding to future values are conditioned on the past. Therefore, these hidden states intuitively represent the past for the model:

$$h^{(t)} = tanh(Wh^{(t-1)} + Ux^{(t)} + b) \quad (1)$$

$$y^{(t)} = f(Vh^{(t)} + c) \quad (2)$$

3.1.2. Convolution-Based Architectures

For many applications, such as time series prediction, the absolute position of a sample is not as important as its relative position. Convolutional layers make use of this property by applying the same convolution operation over the entire input sequence. This is realized by moving a kernel over the entire input sequence to compute an output. The kernel size defines how far the nodes can look to neighboring nodes. This size must be large enough

such that the output is conditioned on all the values that are input into the system, which makes this architecture translation invariant.

Temporal convolution (TC) is a particular technique in CNNs which makes special use of convolutional layers to process temporal information. Their unique property is that instead of learning simple multiplications to change data representations, they learn dilated convolutions [22]. In TC, instead of defining a kernel to look at the immediate neighbors of each node, the kernel only looks at every lth input, where l is the predefined dilation parameter. In mathematical terms, given a 1D input sequence $\mathbf{x} \in \mathbb{R}^{T/2}$ and a filter $m : \{-\frac{k-1}{2}, \ldots, \frac{k-1}{2}\} \to \mathbb{R}$ with an odd filter size k, the dilated convolution operation F_{conv} on element t can be written as

$$F_{conv}(t) = (x *_l m)(t) = \sum_{i=-\frac{k-1}{2}}^{\frac{k-1}{2}} m(i) \cdot \mathbf{x}_{t-l \cdot i} \tag{3}$$

which is similar to the formula from [23].

3.1.3. Transformer-Based Architectures

The transformer architecture was initially introduced in the paper "Attention is all you need" by Vaswani et al. [3]. By default, the transformer has two different types of inputs. The first one is the encoder input, which comes from the input domain of the data. The second type of input is supposed to lie in the output domain of the data. For time series forecasting, we do not have an output domain, and hence it suffices to use the encoder-only transformer depicted in Figure 3. The input embedding is a linear layer which increases the dimensionality of the input time series to the latent dimensionality d_{model} of the network. For this embedding, positional encoding is added to give information about the locality. The main building blocks are attention layers, a special type of layer which computes the output as a weighted sum over the input using some kind of distance metric. In [3], these are described as multi-head attention (MHA) layers, which means that the input is split into a query Q, key K, and value V. The output is then computed in the following way:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V. \tag{4}$$

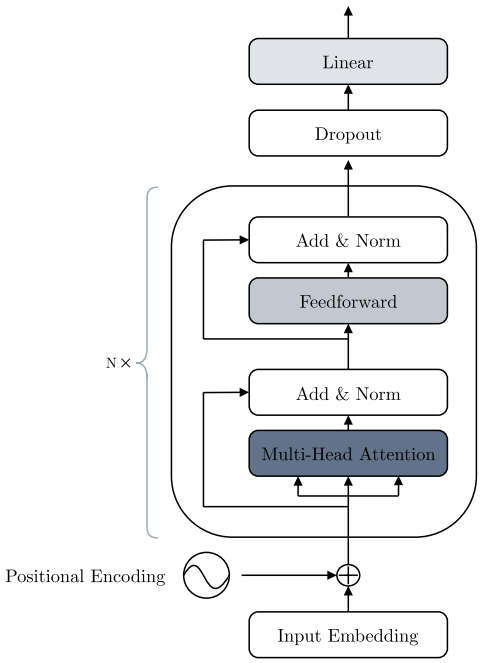

Figure 3. Encoder-only transformer architecture.

The attention is fed into a simple linear layer, and then added to the input of the MHA layer using a residual connection combined with layer normalization and finally another feedforward layer with a residual connection, hidden dimensionality d_{ff}, and layer normalization. This encoder layer is then repeated N times to produce the encoder memory which will be used in the decoder part of the network, as shown in Figure 3.

3.2. Data Synthesis

In this work, we explore whether the characteristics of a time series dataset influence the model's performance. In particular, we focus on four data characteristics: the length of a sequence, the noisiness of the data, the frequency of a signal, and the delay between useful information and the prediction time (differentiating between long- and short-term dependencies).

We used a modified version of the procedure described in [13] to develop our data synthesis approach. Meaningful information was manually inserted into the signal, which allowed for monitoring where this information is located. Additionally, we ensured that this segment was the only segment the model used for prediction. The following equations analytically describe our data generation process:

$$S(t) = \begin{cases} A_1 sin(2\pi t f) + \mu + N_x & t \in [0, t_0), \\ A_2 sin(2\pi t f) + \mu + N_x & t \in [t_0, t_1), \\ A_3 sin(2\pi t f) + \mu + N_x & t \in [t_1, T_s/2), \\ A_4 sin(\pi t f) + \mu & t \in [T_s/2, 3T_s/4), \\ A_5 sin(\pi t f) + \mu & t \in [3T_s/4, T_s), \end{cases} \tag{5}$$

$$\begin{gathered} \tau = T_s/8, \\ t_0 = t_1 - \tau \\ t_1 = \tfrac{T_s}{2} - d, \\ A_1, A_2, A_3 = \text{AmplitudeSampler}(), \\ A_4 = max(A_1, A_2), \\ A_5 = min(A_1, A_2), \\ \mu = \mathcal{U}[47, 97], \\ N_x \sim \mathcal{N}(0, \sigma^2) \end{gathered}$$

All parts use a sinusoidal as their base function. T_s is the sample sequence length and must be distinguished from a dataset's maximum sequence length T. The signal consists of five parts corresponding to the parts with different amplitudes A_1, A_2, A_3, A_4, and A_5. These amplitudes are sampled randomly from a predefined range, such as $[0, 60]$. Therefore, we can create segments from piece-wise sinusoidal functions which carry information or are irrelevant to the prediction. In what follows, we will refer to each segment by its amplitude. In our experiments, the input part of the sequence is where $x \in [0, T_s/2]$. The model is supposed to predict $x \in [T_s/2, T_s]$. The input part has twice the frequency of the output part to make predictions more difficult.

The analytical description immediately shows how we can generate samples with different characteristics. N_x is the noise added to each sampling point of the input part of the time series independently. We controlled the frequency by setting f and the delay by setting d. An additional hyperparameter τ defines the length of the A_2 segment. We fixed this value to be $T_s/8$ throughout this work.

The useful information for the model's prediction is contained in the amplitudes. A_1, A_2, and A_3 are randomly sampled. The sampling approach AmplitudeSampler depends on the experiment, as displayed in Tables 5–7. A_4 is the maximum of the first two amplitudes, while A_5 is the minimum of A_1 and A_2. The A_3 amplitude has the purpose of disturbing the model, incorporating a segment which carries no information. It is easy

to see that a model could calculate the entire output without knowing the value of A_3. Therefore, we refer to this part as a *delay d* in the signal, and we used it to characterize long- and short-term dependencies.

Figure 4 visualizes one realization of this generation process. The blue part is the time series input, while the turquoise part is the output. The vertical dashed lines indicate the different parts of the time series. The first amplitude has the value $A_1 = 60$ and is larger than the second amplitude $A_2 = 40$. Therefore, the first part of the output signal is supposed to have the amplitude $A_4 = max(A_1, A_2) = 60$, and the second part is supposed to have the amplitude $A_5 = min(A_1, A_2) = 40$. The model needs to learn this dependency, dealing with the fact that A_3 has no influence on the outcome of the signal, and use this learned information to make a good prediction. The delay part of the signal can be increased by making the A_3 part of the time series longer. In this example, the delay length is set to $d = 32$.

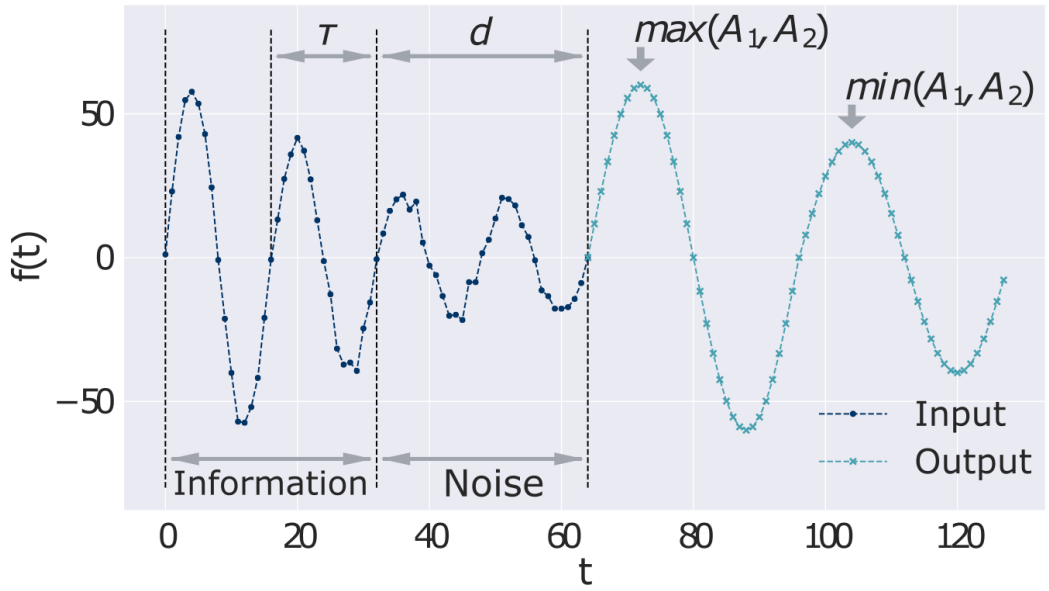

Figure 4. Sample from synthetic dataset with $A_1 = 60$, $A_2 = 40$, $A_3 = 20$, $d = 32$, $f = \frac{1}{16}$, and $\mu = 0$.

3.3. Experimental Design

In the following, we describe the training procedure applied in each of our experiments. Additionally, this section contains the experimental descriptions.

3.3.1. Training Procedure

For all three experiments, we created distinct datasets with predefined characteristics. Thus, for each experiment, we had separate train, validation, and test splits. For a single experiment, all three models were trained on the same train and validation splits. All experiments were performed on an NVIDIA Tesla V100 GPU. Furthermore, all of the training sessions ran for a maximum of 100 epochs. This limit was chosen to cap the training time and provide the same training conditions to every architecture while still achieving convergence of the loss curves. The code for our training procedure is published on GitHub (https://github.com/MischaD/Benchmarking-Univariate-Time-Series-Prediction, accessed on 5 September 2023).

We chose the upper boundaries of the hyperparameter settings so that all models had up to the same maximum amount of parameters, which was set to 450,000 in our experiments. To optimize the hyperparameters, we used the tree-structured Parzen estimator (TPE) due to its ability to outperform random search, as shown by Bergstra et al. [24]. For the transformer model, a hyperparameter search space was defined based on the original values from [3] as inspiration for the final search spaces. We chose the hyperparameters for the other architectures such that the maximum amount of parameters one architecture

could reach did not significantly differ from the others. We performed 25 independent optimizations for every model's configuration using the Adam optimizer in Optuna [25].

All training sessions first performed 10 startup trials. During these trials, the TPE started by randomly sampling from the hyperparameter optimization search space, with a few attempts which were used for exploration. For the third experiment (i.e., sequence length), we decreased the maximum possible model dimensionality of the CNN to make up for the larger amount of model parameters due to the larger maximum dilation. Table 1 shows the hyperparameters for all of the models, including their ranges and the types of sampling.

Table 1. Optuna hyperparameters and ranges which we optimized. Log-uniform means that we sampled the values uniformly in the logarithmic domain.

Hyperparameter	LSTM	CNN	Transformer	Range	Sampling Type
Model dimensionality d_{model}	No	Yes	No	$[1, 56]$, $[1, 48]$	Uniform integer
	Yes	No	No	$[1, 64]$	Uniform integer
	No	No	Yes	$\{8, 16, 32, 64\}$	Uniform
Heads	No	No	Yes	$\{2, 4, 8\}$	Uniform
Architecture depth N	No	Yes	No	$[1, 2]$	Uniform integer
	Yes	No	Yes	$[1, 5]$	Uniform integer
Feed-forward layer dimensionality d_{ff}	No	No	Yes	$[d_{model}, \ldots, 512]$	Log-uniform integer
Kernel size	No	Yes	No	$\{3, 5\}$	Uniform
Learning rate	No	Yes	No	$[3 \times 10^{-5}, 3 \times 10^{-4}]$	Log-uniform
	Yes	No	No	$[5 \times 10^{-5}, 5 \times 10^{-3}]$	Log-uniform
Optimization factor	No	No	Yes	$[0.1, 10]$	Log-uniform
Optimization warmup steps	No	No	Yes	$[200, 800]$	Uniform integer
Dropout	Yes	Yes	Yes	$[0.2, 0.5]$	Uniform float
Weight decay	Yes	Yes	Yes	$[1 \times 10^{-5}, 1 \times 10^{-2}]$	Uniform float
$\sum$	6	5	8		

Furthermore, we employed dropout and weight decay as regularization techniques to ensure that the deep learning models performed well on the training and test data. Dropout achieves this by randomly dropping features in hidden layers of the network to ensure that no features are used as the sole source of information for the output. We employed dropout for all models before the linear prediction layer.

After the 25 hyperparameter optimization trials, the best parameters are shown in Table 2 for the LSTM model, in Table 3 for the CNN model, and in Table 4 for the transformer architecture. Those settings were used for generating the results on the test dataset, as presented in Section 4.

Table 2. Best LSTM hyperparameter settings.

LSTM	N	d_{model}	Dropout	Learning Rate	Weight Decay
Experiment 1	4	38	0.36886	0.00170	0.00002
Experiment 2	5	47	0.35929	0.00172	0.00001
Experiment 3	4	47	0.25551	0.00331	0.00001

Table 3. Best CNN hyperparameter settings.

CNN	N	d_{model}	Dropout	Learning Rate	Weight Decay	Kernel Size
Experiment 1	2	45	0.31065	0.00016	0.00028	5
Experiment 2	2	53	0.24150	0.00011	0.00016	3
Experiment 3	1	47	0.28798	0.00029	0.00001	3

Table 4. Best Transformer hyperparameter settings.

Transformer	N	d_{model}	Dropout	Opt Factor	Warmup	Weigth Decay	h	d_{ff}
Experiment 1	4	64	0.36731	0.62271	430	0.00002	8	489
Experiment 2	3	32	0.33472	1.92401	797	0.00002	8	42
Experiment 3	5	64	0.26745	0.42063	453	0.00002	8	171

We saved a reference to the characteristics we used to create the data for each time series during data synthesis. This way, we could train the different models on a single dataset and look at the differences afterward by marginalizing the MAEs over the characteristics. For example, we only took the predictions from samples without delay ($d = 0$) and calculated the different statistics of these predictions as if they were a single dataset. In the end, we calculated the critical difference [26] to conclude which model performed the best. First, we split the test dataset into different subsets, depending on the amount of different characteristic values, separated by their respective characteristics. Finally, we checked whether or not one model significantly outperformed the other models.

3.3.2. Experiment 1: Delay Length

In the first experiment, we investigated the influence of the delay lengths on the model performance. Table 5 shows the different value ranges for all characteristics of the dataset. The frequency f, sequence length T_s, and noise σ were constant throughout this experiment. The delay values were equally sampled to guarantee their appearing in the dataset in the same amount. We purposefully left out the delay $d = 96$ in the train and validation datasets to further analyze the generalization capabilities of the models during testing.

Table 5. Description of the dataset sampling for the first experiment. The left-out delay length is highlighted in bold font.

Characteristic	Dataset	Possible Values	Sampling Type
T_s	All	512	
d	Train, validation	0, 16, 32, 48, 64, 80, 112, 128, 144, 160, 176	Uniform
	Test	0, 16, 32, 48, 64, 80, **96**, 112, 128, 144, 160, 176	
f	All	1/32	
σ	All	2	
A_1, A_2, A_3	Train	$[0, 60]$	$\mathcal{N}_{Integer}(30, 10)$
	Validation, test		$\mathcal{U}_{Integer}[0, 60]$

As mentioned in Section 3.3, by varying the delay length, we aimed to analyze the difference between short- and long-term dependencies. The length of A_2 was connected to the design parameter $\tau = T_s/8$. To learn the output, the model must observe a few samples from the A_1 part. If the delay length is too high, then preprocessing will cut off the relevant part from the input signal. At least half of the period should be part of the input signal such that the model can observe at least one local maximum of the amplitude. For the analysis of time dependencies, the minimum possible amount of samples which have to be fed to the model to be able to predict the output signal is $T_s/8 + 1/(2 * f)$. For $T_s = 512$, $f = 1/32$, which means that the maximum delay length is $d = T_s/2 - T_s/8 - 1/(2 * f) = 176$. A delay length of $d = 0$ means that the A_3 part of the time series is missing.

We restricted the values of the amplitudes to be integer values within the range $[-60, 60]$. The validation and test datasets were sampled uniformly from this range. During training, the values came from a normal distribution which was cut off at values smaller or

larger than the specified range, and then we randomly multiplied the amplitudes by -1. Figure 5 shows the distributions of the amplitudes in the test and train datasets.

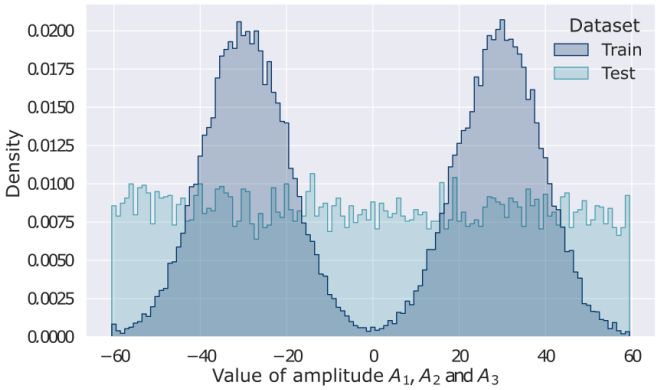

Figure 5. Distribution of the amplitudes in training and test datasets.

3.3.3. Experiment 2: Frequency and Noise

The second experiment examined the influence of the frequency and noise on a model's performance. We combined these experiments into one hyperparameter optimization since both tested the model's sensitivity to variations in the frequency domain. The optimization followed the same procedure as the previous experiment and had the same hyperparameter optimization settings.

For creation of the characteristics of the dataset, we reduced the number of different delay lengths we included. Instead, we sampled from a large range of frequencies and a large range of noise levels. We chose the frequency to capture the entire range of possible values for the sequence length. Theoretically, a frequency of $f = \frac{1}{4}$ would still be feasible. Still, we chose not to include it because for low noise levels, this frequency reduces to a problem where every second value is close to μ and every other value is close to the absolute value of the amplitude. The maximum possible frequency is limited by τ. We ensured that the time series reached the mode of the A_2 part of the signal and returned to zero at least once by setting the frequency to be larger than $1/(\tau * 2) = 1/128$. Similar to the first experiment, we chose to sample the amplitudes from different distributions. Table 6 summarizes the possible characteristics and the sampling strategies chosen for the second experiment.

Table 6. Description of the dataset sampling for the second experiment. The left-out frequency is highlighted in bold font.

Characteristic	Dataset	Possible Values	Sampling Type
T_s	All	512	
d	All	0, 64, 128	Uniform
f	Train, validation	$\frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{128}$	Uniform
	Test	$\frac{1}{8}, \frac{1}{16}, \mathbf{\frac{1}{32}}, \frac{1}{64}, \frac{1}{128}$	
σ	All	2, 3, 4, 5, 6, 7, 8, 9	Uniform
A_1, A_2, A_3	Train	$[0, 60]$	$\mathcal{N}_{Integer}(30, 10)$
	Validation, test		$\mathcal{U}_{Integer}[0, 60]$

3.3.4. Experiment 3: Sequence Length

In the final experiment, we investigated how easily the models could adjust to different sample sequence lengths. Different from the first two experiments, we shuffled and sorted the data as described in Section 3 such that training of the same models was possible for

different sequence lengths. Additionally, we set the maximum sequence length to 2048 (equivalent to 1024 input nodes) in order to fit it on a single NVidia GTX 2080 SUPER (6 GB) GPU. For the transformer time series, this means that the batch size could be as large as four. We expected this to be high enough to still successfully train a model once we obtained a sufficient number of GPUs training in parallel (i.e., the training of one single batch was distributed across multiple GPUs by splitting the batch, forwarding the input on separate GPUs, and then aggregating the loss as described in the documentation of *PyTorch Lightning*). This resulted in hyperparameter optimization with a batch size of 64 for all models.

Additionally, we slightly reduced the hyperparameter optimization range for the CNN to ensure that the number of model parameters across architectures stayed roughly the same. However, due to the longer sequences, the maximum dilation hyperparameter for each TC block was higher, increasing the number of parameters in each block. To compensate for this, we reduced the maximum model dimensionality from 56 to 48 for the CNN during the final experiment (see Table 1). Table 7 summarizes the process of data generation.

Table 7. Description of the dataset sampling for the third experiment. The left-out sequence length is highlighted in bold font.

Characteristic	Dataset	Possible Values	Sampling Type
T_s	Train, validation	128, 256, 1024, 2048	Uniform
	Test	128, 256, **512**, 1024, 2048	
d	Test	0, 32	Uniform
f	All	$\frac{1}{32}, \frac{1}{16}$	Uniform
σ	All	2, 5	Uniform
A_1, A_2, A_3	Train	$[0, 60]$	$\mathcal{N}_{Integer}(30, 10)$
	Validation, test		$\mathcal{U}_{Integer}[0, 60]$

4. Results

In this section, we will present the results of the aforementioned experiments. We analyzed the learning phase in terms of the training times of the different optimization trials and forward pass time per experiment, as well as the performance on the test dataset in terms of the MAE and average MAE per dataset sample. Additionally, we provide the average root mean squared error (RMSE) and mean absolute percentage error (MAPE) per architecture in our experiments.

4.1. Experiment 1: Delay Length

4.1.1. Learning Phase Analysis

To analyze the effect of different model initializations, Figure 6 shows how long each of the 25 separate trials took in seconds. As expected, the LSTM architecture took the longest to train due to its poor parallelization capability. On the other hand, the CNN was slightly faster than the transformer architecture, plausibly due to the large matrix multiplications in the self-attention layer. However, the transformer model suddenly became the slowest of all three architectures during inference time, as shown in Figure 7. The memory bottleneck of the transformer model was previously described by Li et al. [13]. On the other hand, the CNN approach was the fastest of the three by a large margin.

Figure 6. Training times during hyperparameter optimization in Experiment 1.

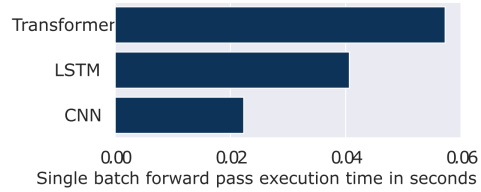

Figure 7. Single-batch forward pass averaged over five test iterations.

In Figure 8, the loss curves of the best runs from the validation set are displayed for each architecture. During the initial convergence, the curves of the transformer, CNN, and LSTM models are pretty similar. However, after 20 epochs, the transformer model reached a lower MAE than the CNN and LSTM models. It is observable that the curves of the CNN and LSTM approaches are also more erratic. After 100 epochs, the transformer architecture presented the lowest validation loss, which was close to the loss of the LSTM network, while the CNN model converged to a higher MAE level of 0.06.

The distribution of the validation MAE depicted in Figure 9 shows that the transformer model was more susceptible to changes made to the architecture, like the learning rate and model size. While not performing as well as the other two models, the CNN model did not show much variance. This may be related to the fact that many CNNs failed to learn and were close to the value of the MAE which a simplistic model reaches that uses the mean of the input as the value for prediction, depicted in the figure with a dashed line. Interestingly, the training time suggests that the CNN models with the longest training time did not perform the best. This agrees with the observations that the CNN model was already quite deep due to the TC dilations and consequently subject to vanishing gradient problems.

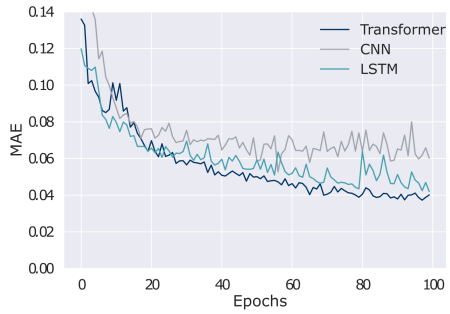

Figure 8. Validation loss curve of the best runs for each model architecture in Experiment 1.

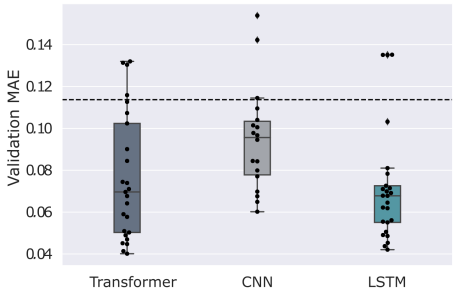

Figure 9. Validation MAE of each training session as a boxplot. The dashed line indicates the performance when taking the mean of the input as the output prediction.

4.1.2. Performance Analysis

Table 8 summarizes the average error metrics of the first experiment. The Transformer architecture displays the lowest values across MAE, RMSE, and MAPE. Figure 10 shows how the delay length influenced the confidence interval of the median MAE, which we calculated according to the description in Section 3.3.1. At first glance, this graph shows that the LSTM and transformer models performed comparably and were able to better deal with delays than the CNN approach. The transformer model generally showed more stable results, with smaller confidence intervals for the median.

Table 8. Average errors of the test dataset for the first experiment. Lowest error values are highlighted in bold font.

Model	Average MAE	Average RMSE	Average MAPE
Transformer	**0.040**	**0.066**	**9.925**
CNN	0.054	0.074	14.209
LSTM	0.043	0.069	11.317

The drop in performance for larger delays comes from the fact that for large delays, essentially all of the input time series only consists of noise. Therefore, earlier input values are generally more reliant, and the models should be able to learn this. Additionally, the A_1 part became relatively short for long delays, which means that the models could only observe rather few noisy observations of that amplitude. Figure 11 verifies this observation by showing that heavy outliers started to appear at delay lengths greater than 64.

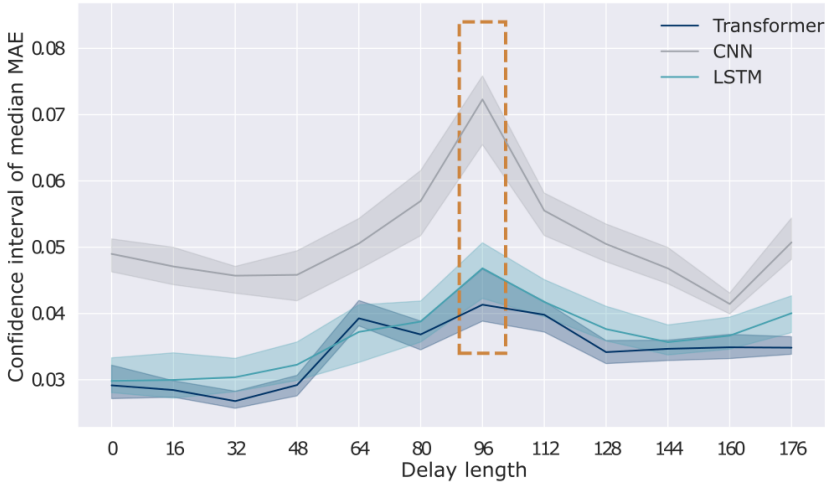

Figure 10. Confidence intervals for different delay lengths. Delay length $d = 96$, marked by a brown rectangle, was not present during training and validation time.

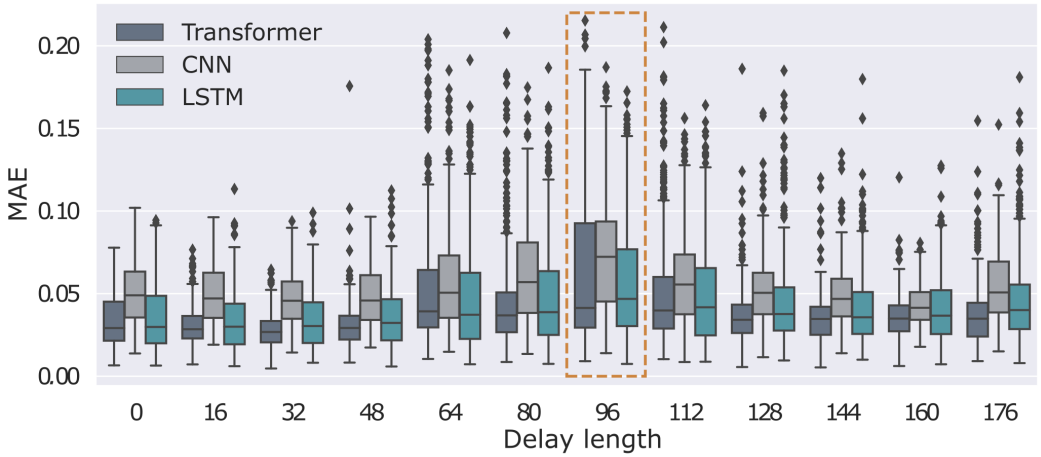

Figure 11. Boxplot of distributions of MAEs of single samples, separated by their delay length. Delay length $d = 96$, marked by a brown rectangle, was not present during training and validation time.

As described in Table 5, we tested the dataset on a delay length which did not appear in the training or validation datasets. Therefore, when looking only at the confidence interval of the median of the generalized delay length in Figure 10, we observe that the transformer model exhibited the best generalization performance for an unseen delay length. However, upon inspection of Figure 11, it becomes clear that the transformer model had a considerably higher number of outliers. Furthermore, the CNN model handled the generalization worst out of the three models. Overall, this shows that even though the transformer model has global attention, which makes the intuition behind long and short-term dependencies relatively unimportant, it did not outperform the LSTM approach for long-term dependencies as clearly as expected based on the work of Li et al. [13]. The cause for these results might be that changing the application to a multi-input multi-output (MIMO) approach reduced the disadvantage of vanishing gradients for the LSTM approach.

For the subsequent analysis, we investigated how different amplitudes affected the model performance. This analysis was slightly biased because high amplitudes resulted in higher values for the MAE loss. However, looking at the distribution and spread of the MAEs still yielded interesting insights into the influence of the differences in distributions between the test and train sets. Figure 12 shows the effect of the value of the first amplitude if the second amplitude was $\in [-60, -50)$. The LSTM model has a massive spike at the lowest amplitude, suggesting that it did not generalize well over the amplitude distributions and had the most trouble distinguishing between similar amplitudes A_1 and A_2. On the other hand, the transformer model was not influenced drastically by different amplitudes and therefore made a reasonably good generalization over the amplitudes. Figure 13 plots the average MAE for each combination of amplitudes of each model. The LSTM model had more modes around the areas where the data distributions had their modes, namely around 30 and -30. On the other hand, the transformer model showed a more consistent heatmap, further reinforcing the assumption that the transformer approach generalizes better among different amplitudes.

Figure 14 shows the results of the critical difference. The lower the score, the better. This confirms that the LSTM and transformer models significantly outperformed the CNN architecture. However, it also indicates that the difference between the LSTM and transformer architectures was not significant.

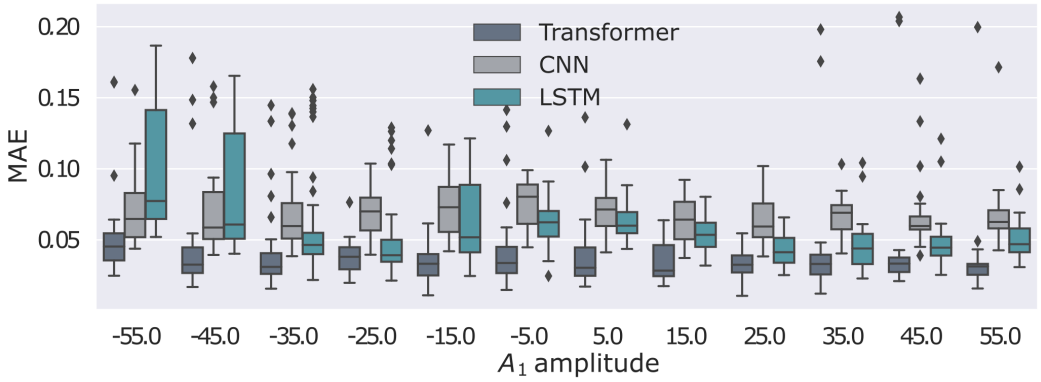

Figure 12. Boxplot of MAEs of samples with $A_2 \in [-60, -50)$, separated by their values for A_1.

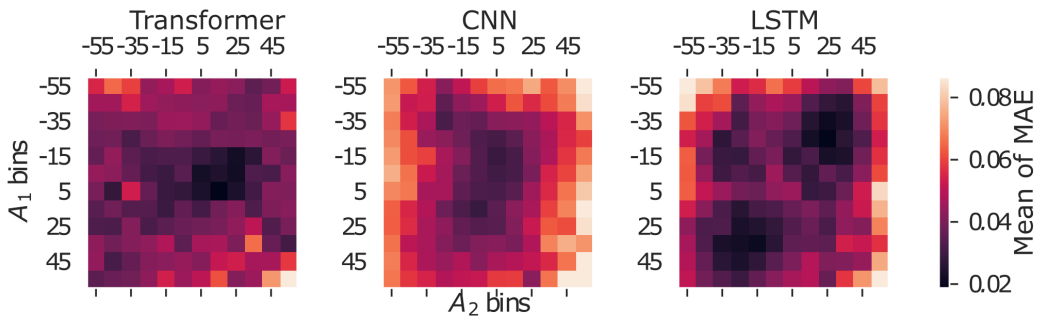

Figure 13. Heatmap of average MAEs of each model separated by amplitude.

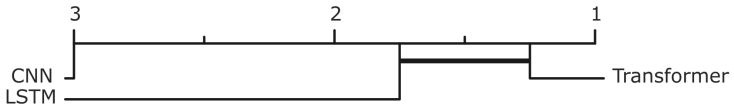

Figure 14. Critical difference of the models when separating the dataset by their delay lengths.

4.2. Experiment 2: Frequency and Noise

4.2.1. Learning Phase Analysis

Similar to the findings for Experiment 1, Figure 15 shows that the training time of the LSTM model was about twice as long as the training times of the other two approaches, but the overall time needed for hyperparameter optimization was comparable. The distribution of validation losses during hyperparameter optimization in Figure 16 shows that not only was the LSTM model the best model during this experiment, but the top five models in terms of performance during hyperparameter optimization were all LSTM models. The distribution of the CNN architecture's performance even shows that the best-performing model seemed more like an outlier in this case, further diminishing the suitability of this type of architecture for learning the designed dataset.

Figure 15. Training times during hyperparameter optimization in Experiment 2.

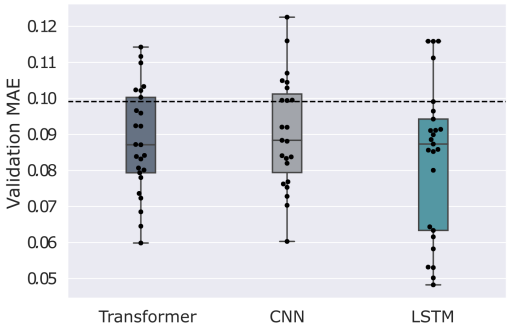

Figure 16. Validation loss of all models. A dashed horizontal line indicates the performance of the mean estimator.

The validation loss curves in Figure 17 look different for the three models. The transformer model's loss was the quickest to improve and only marginally continued to decrease for the rest of training. The CNN model's convergence was slow. For this model, it could be possible that training for even more epochs would further improve the performance. However, as previously mentioned, all models were given the same maximum number of training epochs.

In contrast to the previous experiment, the transformer models showed lower computation times than the LSTM models, as depicted in Figure 18. This was due to the fact that for this experiment, the depth of the optimal transformer model was smaller than that of the optimal LSTM model. This is interesting because, combined with the results from Section 4.1, we can conclude that it is not possible to establish a priori which of the two architectures will be more efficient in terms of computing time. As expected, the CNNs were consistently faster in computation.

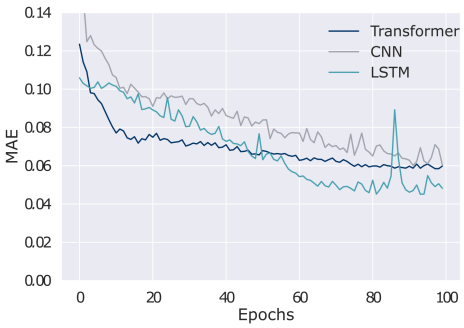

Figure 17. Validation loss curve of the best run for each model architecture in Experiment 2.

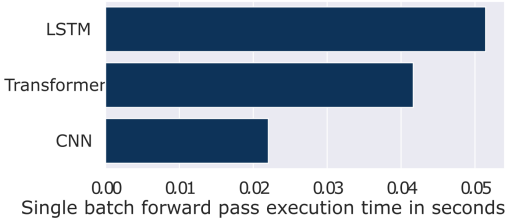

Figure 18. Inference time of a single forward pass with a batch size of 64 averaged over five runs on the complete test dataset.

4.2.2. Performance Analysis

Table 9 summarizes the error metrics of the test dataset. The LSTM model was the best model for this dataset. Both the transformer and CNN models performed considerably worse than the LSTM counterpart. The error values were noticeably higher than the best performances in the first experiment, suggesting that this training set was much more difficult for all the models. One possible reason for this is that the output now changes

not only the amplitudes but also the frequencies. As a result, the first experiment only required learning of the sinusoidal base function, which was the same for all of the samples. Furthermore, the amount of noise injected in this experiment was significantly higher, and outliers highly compromised the performance in terms of the MAE.

Table 9. Average errors of the test dataset for the second experiment. The lowest error values are highlighted in bold font.

Model	Average MAE	Average RMSE	Average MAPE
Transformer	0.073	0.102	16.731
CNN	0.071	0.097	15.734
LSTM	**0.054**	**0.087**	**12.825**

We analyzed the impact of changing the noise level of the dataset on the prediction performance. Figure 19 shows the performance of each model as a function of the noise level. As mentioned in Section 3.3.3, additive noise was only injected into the input. Therefore, the output and the ground truth which the model was trained on were not affected by σ. The graph clearly shows that all models could handle noise without any evident loss in performance, even when the noise energy of the signal was higher. This contrasts with previous findings by Greff et al. [27] on the influence of different hyperparameters on the LSTM architecture. They showed that adding noise to the input as a form of regularization did not help the model and negatively influenced the performance. A possible explanation for this discrepancy is that in our case, the models were trained on different noise levels simultaneously.

Next, we looked at the ability of the models to learn different frequencies. Figure 20 shows the MAEs of the samples separated by their frequencies. LSTM was the best model across the whole of this characteristic. It had the best-performing median in each case, and all the single best predictions in each category came from the LSTM model. It performed consistently well throughout the frequency range, although a slight increase could be observed for higher frequencies when looking at the confidence interval for the median. The performance of the CNN approach was qiute close to that of the transformer model for all frequencies, especially for low-frequency signals. The CNN model had a great deal of trouble capturing the information in the signal for extremely high frequencies, as evidenced by the increased median and variance for this dataset.

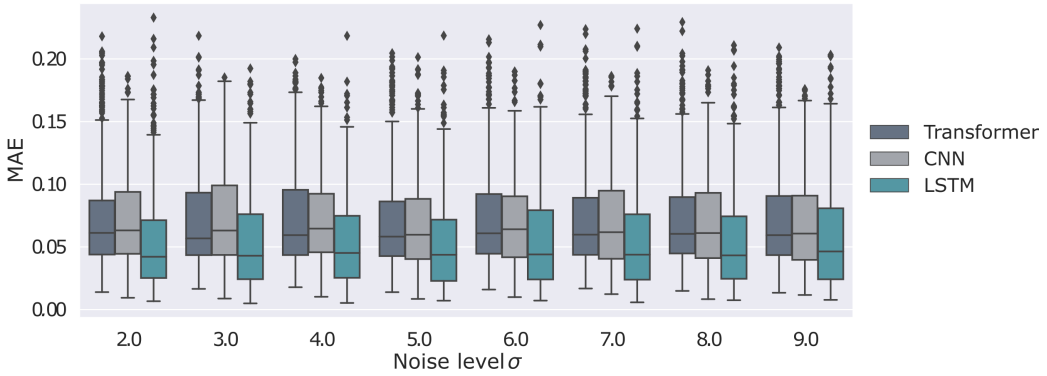

Figure 19. Boxplot of the impact of the noise level on the prediction performance.

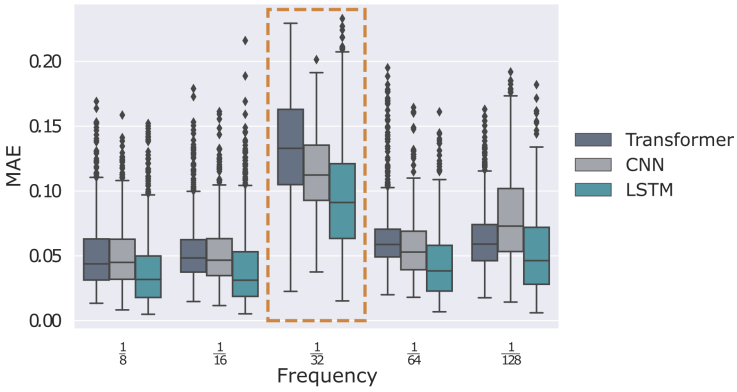

Figure 20. MAEs, separated by frequencies. Here, $f = 1/32$, marked by a brown rectangle, was not present during training or validation.

The frequency $f = 1/32$, marked by the brown rectangle in Figure 21, was not present during training. All of the models had difficulties generalizing toward this new frequency. The LSTM model was still significantly better than the other two models, but the performance was barely on par with what a predictor which only predicts the mean would achieve (see Figure 16). (For reference, the mean performance of this estimator was 0.0991 on the test dataset, whereas the LSTM model reached a mean of 0.0958 on the generalized frequency.) Figure 22 shows the mean of the MAEs with respect to the amplitude bins for the samples with a frequency $f = 1/32$. The transformer model only had reasonably good performance in the area where the amplitude was small, but this was also where failed interpretations, like predicting the wrong frequency f, had the slightest impact. This can also be confirmed by looking at some of the best predictions from this frequency, such as the samples with small amplitudes. In this case, the model did not learn to generalize over the frequency but instead predicted some signal which looked similar to a mean predictor.

We calculated the critical difference for the second experiment, as shown in Figure 23. The analysis included a total of $3 \times 5 \times 8 = 120$ different datasets, which were generated from the three different delay lengths, five different frequencies, and eight different noise levels used for synthesis, as described in Table 6. As anticipated by previous analysis, the LSTM model outperformed its competitors significantly during this training period, and the CNN model was considerably better than the transformer approach. Furthermore, the LSTM approach performed the best across all the models in all but two datasets.

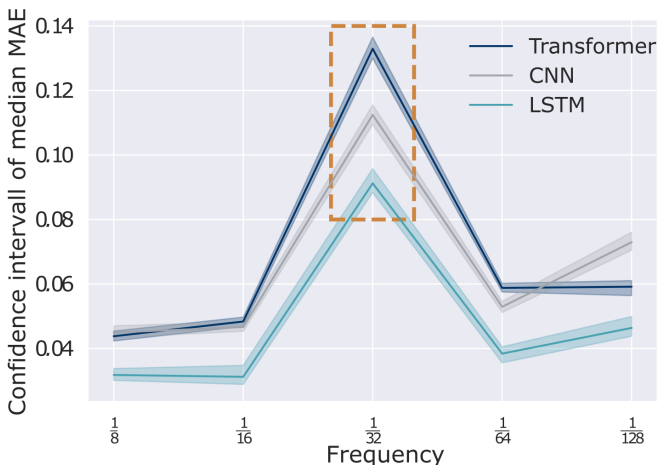

Figure 21. Confidence interval of the median for various frequencies. Here, $f = 1/32$, marked by a brown rectangle, was not present during training or validation.

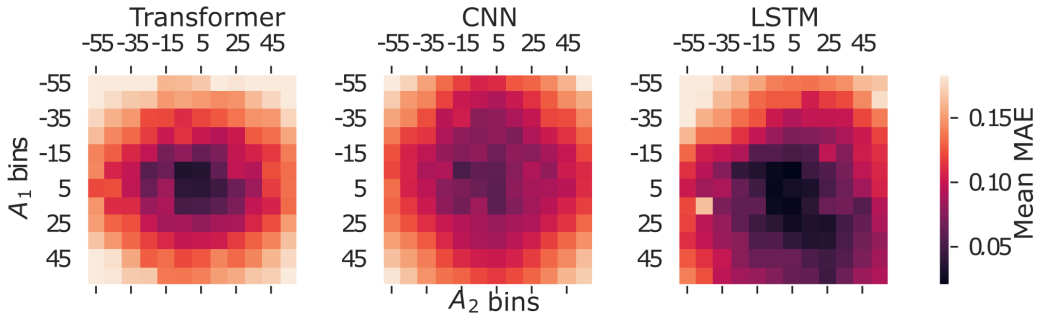

Figure 22. Mean MAEs of all three models, depending on the amplitude only for data with the frequency that did not appear during training.

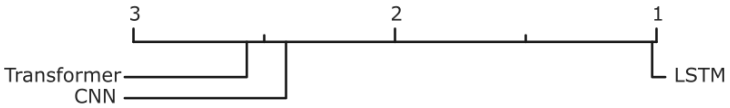

Figure 23. Critical differences of the architectures on the test dataset.

4.3. Experiment 3: Sequence Length

4.3.1. Learning Phase Analysis

In the last experiment, we investigated the effect of varying sequence lengths on model performance. The mean MAE across the entire test dataset, summarized in Table 10, shows that the LSTM approach had the best performance on the test data. The CNN model was, again, slightly ahead of the transformer model for this metric. Overall, lower MAE values suggest that the present task was generally easier than that of Experiment 2.

Table 10. Average errors of the test dataset for the third experiment. The lowest error values are highlighted in bold font.

Model	Average MAE	Average RMSE	Average MAPE
Transformer	0.049	0.079	11.23
CNN	0.044	0.067	11.615
LSTM	**0.034**	**0.059**	**8.161**

The training times shown in Figure 24 are similar to those observed in the previous experiments. The corresponding validation errors of all optimization trials are summarized in Figure 25. All models offered equal variance in their performance across different hyperparameter settings. It is essential to notice that the best transformer model seemed to be more of an outlier for this particular experiment than the other transformer trials. However, a closer analysis of the transformer training times in Figure 24 suggests that the best-performing model was also the largest model with the longest training time, which means that a longer training time could potentially lead to better performance for this architecture. However, as described in Section 3.3.1, we set the tuning trials to a certain amount to give all models the same resources for training.

Figure 26 shows the validation loss curves for each model. The CNN and transformer validation losses seemed relatively stable. The LSTM approach took the most epochs to converge but ended up with the best validation loss. The inference time, visualized in Figure 27, shows that the CNN model was the quickest model, followed by the transformer model.

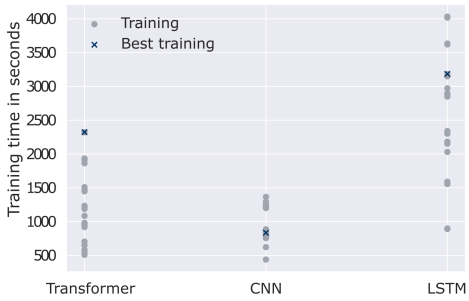

Figure 24. Training time during hyperparameter optimization in Experiment 3.

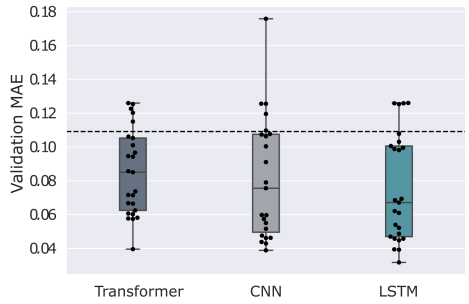

Figure 25. Best validation loss of all models.

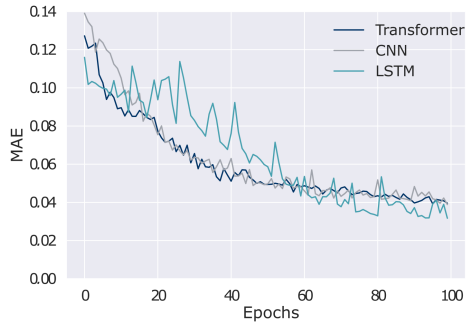

Figure 26. Validation loss curve of the best-performing models for each type in Experiment 3.

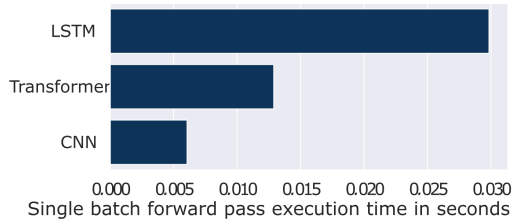

Figure 27. Inference time of a single forward pass with a batch size of four, averaged over five runs over the entire test dataset.

4.3.2. Performance Analysis

During training, each model only saw four different sample sequence lengths. The fifth one was only present during testing. In this way, we were able to analyze how increasing the sample sequence length influenced the model performance and explore how each model generalized to a new length.

In this experiment, the prediction task should have presented better outcomes for longer sequences, since longer sample sequence lengths meant the models would have

more observations, leading to better accuracy. Also, long sequence lengths cannot be entirely disconnected from long delay lengths. The parameter τ, introduced in Section 3.2, is connected to the sample sequence length, which means that the last observation of the mode of the A_1 amplitude is at $T_s/2 - (d + \tau + 1/(2 * f))$.

Figure 28 shows how each model performed for any given sample sequence length. All of the models could forecast the four sample sequence lengths present during training. The median of the LSTM model was best for all datasets in this experiment. The longer sequences showed less variance and fewer outliers, which was most evident in the case of the transformer models. This is potentially related to the fact that the transformer approach successfully extracted the information from the noisy input data even for extremely long signals, especially the sign and absolute values of A_1 and A_2. Upon closer qualitative inspection, we found that the transformer model often had difficulties learning that the output signal was only supposed to have two different amplitudes. Instead, it constantly increased or decreased the amplitude from higher to lower values. The LSTM approach, while being the best model, had the most outlier performances and many predictions which were significantly worse than the mean. These outliers, however, were quite rare and corresponded to predictions where the sign of one of the amplitudes was flipped. Therefore, the prediction was phase-shifted, resulting in a worse MAE. The CNN and transformer architectures, on the other hand, did not show these outliers.

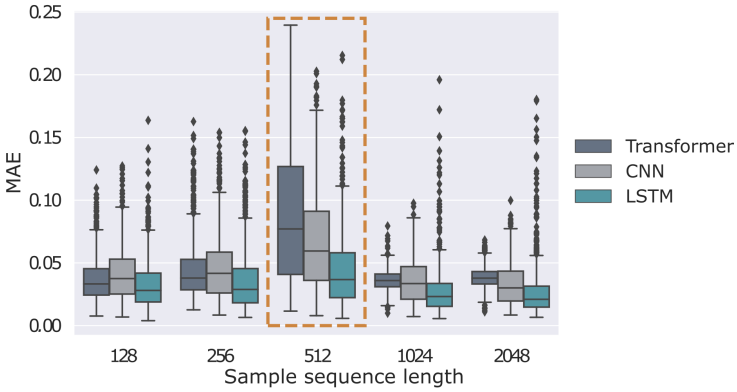

Figure 28. MAEs of different sample sequence lengths. Here, $T_s = 512$, marked by a brown rectangle, was not present during training or validation.

The transformer and CNN approaches were not able to generalize to a new sample sequence length (see the brown rectangle in Figure 28). Instead, the LSTM model only had a slight drop in performance and still produced good results. Figure 29 shows the best predicted sequence according to the MAEs from the entire dataset, which came from the generalized sample sequence length $T_s = 512$.

To summarize the findings, we again calculated the critical difference by treating each combination of characteristics as one dataset. This led to a total amount of 40 datasets. The LSTM approach was the best one in 38 out of 40 cases. (The transformer approach was better in both datasets with the following characteristics: $f = 1/32$, $d = 32$, $T = 128$, and $\sigma \in \{2, 5\}$). The difference in model performance was statistically significant, as shown in Figure 30. The CNN model once again was significantly better than the transformer model. The transformer approach struggled with generalizing to new sample sequence lengths but showed the most robust estimations toward long sample sequence lengths.

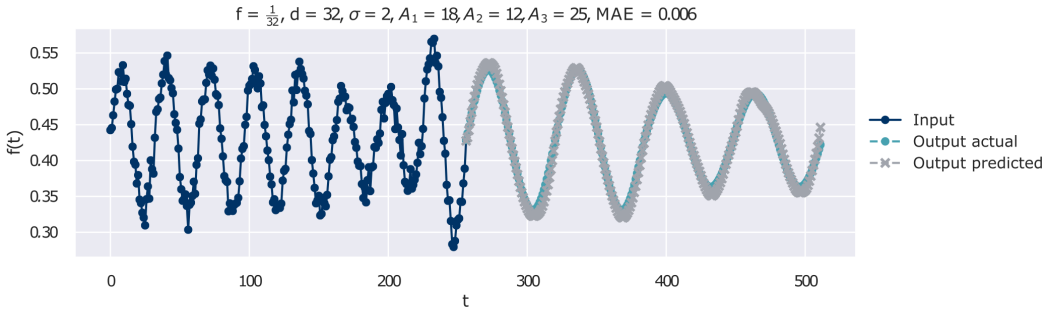

Figure 29. Best predicted sequence coming from a sample sequence length unseen during testing.

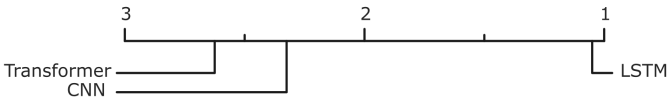

Figure 30. Critical difference of the architectures on all different datasets.

5. Discussion

This section discusses the results in Section 4. First, we derive recommendations based on the performance of the LSTM, CNN, and transformer architectures. Additionally, we elaborate on the limitations of our approach and the conducted experiments.

5.1. Recommendations

In our experiments, we demonstrated that all neural network models handled the given challenge successfully by producing low error values (see Tables 8–10). However, there were specific performance differences between the models, as described in Section 4. These differences allowed us to derive the following recommendations for machine learning practitioners who intend to apply neural networks to univariate time series forecasting use cases with individual real-world data.

Assessing dataset characteristics

Depending on the use case, it can be more or less challenging to assess the characteristics of a dataset. But more often than not, this is relatively straightforward and usually performed during the data exploration phase of data science projects [28]. We recommend the following studies to assess the characteristics investigated in this paper. To assess the parameter we denote as the delay length, one should look at the long- and short-term dependencies appearing in the sequences which are used as inputs for the models. This is also called trend analysis and is often given by the data [29]. Depending on the outcome, it can be assessed what delay lengths or temporal dependencies the model has to handle. Then, frequency and noise levels can be assessed by analyzing the frequency spectra of the training data by applying, for example, a Fourier transformation [30]. Additionally, one could assess the variance in the spectrogram and detect the most prominent frequencies. On the contrary, the sequence length is usually a factor defined by the given data or the application of the given use case. More detailed elaborations on how to perform thorough exploratory data analysis can be found in [31].

Model selection in consideration of delay lengths

When selecting a model regarding varying delay lengths, we recommend first looking into transformer- or LSTM-based architectures. While the transformer model was the best-performing model, as shown in Table 8, its margin was not significantly better than that of LSTM. Both architectures are well suited to modeling the temporal dependencies in data. As described in Section 3.1, LSTM uses a hidden state to memorize information which spans many time steps. During the training process, LSTM is tuned to recognize when parts of the input shall be memorized and when input information can be discarded. This explains the

good performance in the delay length experiment. Comparable findings regarding recurrent neural networks were found in the experiments of Hewamalge et al. [15]. In contrast, the transformer architecture processes the input sequence as a whole. By employing positional encoding, the position of relevant information in the sequence, and therefore the delay length, does not present a difference from the transformer approach's processing structure. With the multi-head attention blocks, as shown in Figure 3, each input time step is analyzed regardless of its sequential position. Li et al. [13] presented a similar finding, where the transformer approach could model long-term dependencies better than the LSTM model in their experiments. A CNN architecture, on the other hand, captures local patterns and features across a sequence. In our experiments, these locally receptive fields did not maintain the information as well as the cell memory of the LSTM model or the positional encoding or attention blocks of the transformer architecture. This appears to be contrary to the findings of Lara-Benitez et al. [32]. They reported that their CNN model with temporal convolutions outperformed LSTM models. However, they applied a more specialized CNN variant to their prediction task. From our experiments, we concluded that for assessing information where the context of its position in the sequence is a global factor, LSTM and transformer models are better equipped in terms of their processing structure.

Furthermore, only a few outliers were present for small delay lengths when inspecting the performances of the individual delay lengths in Figure 11. However, as the delay length parameter increased to values greater than 64, the variation in the model performances also increased. Therefore, we concluded that a larger delay length was a more challenging task for the models. This was apparent since the information had to be maintained for more time steps, and more information had to be processed. For the transformer approach, however, the spread in Figure 11 is not as prominent as those for the CNN and LSTM models since positional information was processed at once. The most significant variation in the performances can be seen for the delay length, which was only present in the test set. However, when looking at the delay lengths from 122 to 176, we can see that the transformer architecture presented the most minor distribution in the boxplots and was therefore less affected by larger delay values. In transformer-based architectures, the notion of delay length is no longer present. Attention modules allow for attending to any value of the past sequence in O(1), whereas convolutional blocks take O(log(n)) and LSTMs take O(n). This is an explanation for why they performed so well in this experiment. Hence, we recommend the transformer architecture as the first choice for data with long temporal dependencies when computational resources are not a limiting factor. With greater delay lengths, the input sequences are longer as well. This can drastically increase the training time and memory requirements of the transformer model. In this case, the LSTM approach can still be a good choice, despite its slightly reduced performance in the delay length experiment.

Model selection in consideration of frequency and noise

For selecting a model while observing a great amount of noise in the data, we could not identify a clear favorite since all model performances were unaffected by noise, as shown in Figure 19. All three—transformer, CNN, and LSTM architectures—present similar boxplots throughout the different noise levels we investigated during our experiments. On the contrary, we saw a more noticeable difference in the models when examining the frequency settings. With a held-out frequency value of $1/32$, the LSTM model showed the best results by a significant margin. Therefore, we recommend this architecture when dealing with varying frequencies in a dataset. Figure 20 and Table 9 display this performance difference. By examining many advanced LSTM variants, Hewamalage et al. [15] also concluded that recurrent neural networks are quite capable of adapting to seasonality in datasets, which aligns with our findings. Both the CNN and transformer approaches presented significantly lower performance than the LSTM model. The CNN model employs dilated convolutions on the time axis. It therefore learns the context of the information along this axis during training. When new frequencies appear in the test set, this becomes difficult to model, as shown in Figure 20. Here, the MAE increased for all three architectures for the held-out

frequency. Similar observations were present for the transformer model, which exhibited the highest MAE in Table 9. The attention heads were tuned with regard to the positional information of the input data during training. Generalizing to a changing frequency thus did not work as well as with the LSTM architecture. This was denoted as *locality-agnostics* by Li et al. [13], a major drawback of the transformer architecture which makes the model less sensitive to the local context. Simultaneously, the LSTM approach employs feature learning and temporal processing with its recurrent connections and cell state. The vanishing gradient problem is also diminished when information occurs not only in the long term but seasonally, as in our second experiment [33]. Additionally, this architecture is designed to handle variable-length inputs. In combination, this enables the LSTM approach to model different frequencies better than the CNN and transformer architectures.

For data containing frequent occurrences of relevant events, we recommend applying network architectures which use recurrent processing, since the LSTM model showed the best performance in the frequency experiment.

Model selection in consideration of sequence length

Our experimental findings suggest a preference for employing an LSTM architecture in scenarios involving varying sequence lengths. This assertion is supported by the MAE values in Table 10, which notably demonstrate a significantly lower MAE for the LSTM approach than those observed for the CNN and transformer models.

With its recurrent cell structure, the LSTM model is, by design, able to process sequences with variable lengths [15]. Also, the gating mechanism and the memory cell state make this architecture adaptive to sequence lengths. In contrast to the CNN and transformer architectures, this enables LSTM to process time series data without implicitly encoding the notion of time. The transformer model, on the other hand, utilizes a specific layer to encode the position of each time step numerically for the attention mechanism. In particular, with the held-out sequence lengths, the error of the transformer approach increased, as shown in Figure 28. Although the CNN approach generalized better to the held-out sequence length of 512, it showed the largest overall MAE in Table 10. As mentioned before, the CNN architecture operates with receptive fields which capture local features and dependencies. Varying sequence lengths could cause these features to shift on the time dimension. Consequently, the CNN model has to adapt to this positional independence during training, which adds to the complexity of the learning task.

Our recommendation finds further support in Figure 28. Upon analyzing the mean MAEs for the three architectures with respect to the trained sequence lengths, we observed their proximity to be within a narrow range. Nonetheless, the LSTM model presented an elevated number of outliers for longer sequences attributed to the challenge of vanishing gradients. Despite this, the LSTM model consistently yielded optimal performance across varying sequence lengths. Additionally, it is necessary to underline that the transformer architecture's computational demands significantly escalate with longer input sequences. Thus, it is advisable to explore novel strategies to enhance the efficiency of transformer models, as discussed in the work of Wen et al. [4]. Although not presenting the lowest mean MAE, the transformer approach's resilience to extended sequences contributed to the reduced existence of outliers and a narrower spread of the upper and lower quartiles within the depicted box plots.

5.2. Limitations

Contrary to other publications which evaluate neural network architectures for time series prediction, our approach focuses on the dataset's perspective by assessing model performance with regard to characteristics in the data. Unsurprisingly, this practice has certain limitations which will be discussed in the following.

Data synthesis

The way we synthesized our data is quite distant from real-world data regarding variability and complexity. We described the process in Section 3.2. However, we had to consider a trade-off between more real-looking data and pronounced characteristics. We understand that this scenario might not provide the most challenging learning task for the networks. However, a more complex or real-world dataset would not have allowed for connecting model performances to the characteristics we investigated in the respective experiments. Our results show that different dataset characteristics led to different performances in the network models. Consequently, our goal was not to identify the sole best-performing architecture for any univariate time series prediction task but to enable ML users to choose applicable models from a family of network architectures based on certain dataset characteristics. We acknowledge the increased complexity of real-world data. But more often than not, there is a dominant characteristic in the data which the network has to deal with or is relevant to the prediction task, such as the periodic property of biomedical signals or frequencies in sound-based signals.

Benchmarking time series has been approached from different angles. Other techniques often include a selection of datasets and models to be evaluated as in [34], which is specific to the use case of solar radiation. Additionally, approaches like the Libra framework propose automated benchmarking [35]. For all of these solutions, certain constraints have to be defined on the data or model side. While we do not see our approach as a replacement for other existing benchmarking techniques, we understand ours as an extension and different point of view on gaining a prior comprehensive understanding of which model to choose based on the data. If we evaluated our chosen architectures on a selection of real-world and synthetic datasets, then we would lose the ability to accurately control the characteristics we intended to investigate in the experiments. With our approach, we exclude influences other than the characteristics to be investigated and the model structure at hand. Since we changed the perspective of model benchmarking to a data-centric view, our vision for the future would be to evaluate every new state-of-the-art (SOTA) model on a volume of synthetic datasets with isolated characteristics in a standardized fashion. This would enable the direct assessment of new SOTA models on the given characteristics.

Basic model architectures

In our experiments, we purposely used the most basic form of the LSTM, CNN, and transformer architectures by closely following their first presentations in the literature. While we recognize that restricting ourselves to basic variants of the three architectures may limit the overall applicability of our conclusions, we believe that examining these fundamental networks allows us to establish a direct connection between performance outcomes and the intrinsic processing capabilities of these models. More advanced variants of these architectures may achieve improvements both performance-wise and efficiency-wise. However, the underlying procedure of how they encode information and predict the future is the same. The assumption we make is that we can reduce architectures to their individual way of modeling temporal dependencies and learn from observations based on them. By applying advanced models, we would not be able to tell whether the monitored performance in the time series prediction task originates from how the different neural network architectures process the data or whether it results from clever encoding or adjustment of the model. To the best of our knowledge, this work is the first instance of trying to change the perspective on model selection from a model-centric to a data-centric view. Therefore, this first investigation focused on these three fundamental neural network architectures for time series forecasting.

Also, classical ML models might solve the prediction tasks in our experiments with comparable or even better results. However, regression methods like ARIMA, gradient boost, or random forest can be applied with less resource investment. For neural networks, the implementation and training overhead can be a roadblock to evaluating these models.

Therefore, we want to provide insights on making an informed decision before selecting and applying a network architecture for a time series prediction task.

Univariate prediction task

Observing model performances from a data perspective required us to constrain the data structure and characteristics. For our experiments, we chose the univariate multiple-input multiple-output prediction task. Consequently, the derived recommendations can only be applied in such scenarios. Incorporating, for example, multivariate time series analysis requires a considerable rework of the experiments and opens up further design questions. Do we model each time series as independent of each other, or do we correlate them? Adding more experiments means that the search space for our experiments grows exponentially. In future experiments, this can provide further interesting insights. However, univariate time series prediction is a prevailing task in many domains which is still actively researched [1]. This underlines the relevance of this work.

Time series are not only processed for forecasting but also for the tasks of classification and anomaly detection. For classification scenarios, the general structure of our experiments could be adopted, which would be an interesting path of future research. Here, the delay characteristic could be exchanged for a parameter which is more prevalent in classification, such as the balance of classes in the dataset. In anomaly detection, one could investigate which architecture performs best on which type of anomaly (point, collective, or contextual). However, the data synthesis here needs to be changed. Generally, we encourage our approach being extended and transferred to other domains and applications. We further consider our work a first step in changing the performance assessment perspective to a data-centric view.

Training resources

We trained the models for 100 epochs on NVIDIA Tesla V100 GPUs. This limited the potential final performance of the models, since we could not preclude that more extensive training would lead to better results. However, our goal was not to tune one model to its fullest potential but rather to give each architecture the same resources for training to conduct a comparative evaluation with regard to the dataset.

Hyperparameter tuning

Some of the best model performances were observed from the last trails of the Bayesian hyperparameter optimization. This is a potential indicator that increasing the search space toward larger model sizes would also improve its performance. However, as mentioned in Section 4.3.1, all models were given the same budget. This also emphasizes the influence of stochasticity in our experiments.

6. Conclusions

This paper provides new insights into the performance of neural network-based architectures for time series prediction. Since transferring reported results from recent publications for real-world use cases can be challenging, we switched to a data-centric perspective and linked model performances with defined dataset characteristics.

Therefore, we created synthetically generated datasets, in which time series were obtained as a combination of sinusoidal functions, where we can control various levels of delay length, frequency and noise, and sequence lengths for our convenience. Finally, we conducted a separate experiment to test the network architectures for their performance during training with a held-out evaluation set for each of these characteristics.

The models we investigated for this work are the main neural network paradigms used for time series prediction in the literature: LSTM-based, CNN-based, and transformer-based models. We intended to assess these models by their most basic structures to relate the models' performances with specific data characteristics in the most direct way.

In our experiments, we evaluated the architectures during their training phase and their performance after training. The transformer model showed the best MAEs for various delay lengths (Experiment 1), although not by a statistically significant margin. Except for the positional encoding, the transformer approach has no sense of position, and thus we expected that this architecture would generalize well on multiple delay lengths (as shown in [13]). The LSTM approach performed significantly better when varying the frequencies and noise levels (Experiment 2) or sequence lengths (Experiment 3).

When observing the obtained results between our experiments, we can note that the MAE values were in a similar range between 0.01 and 0.2, including outliers. This implies that, on the one hand, the experiments were on par in terms of the learning challenge the models had to handle, and on the other hand, those challenges were suitable to all models. Nonetheless, we could monitor differences in the performances and draw the conclusions above, also demonstrating that we could use synthetically generated time series data to reveal those differences.

For future work, we suggest that the difficulty level of the experiments can be increased by synthesizing more complex datasets. Incorporating more variance in the data could already create a more challenging learning task. Furthermore, in our experiments, we were only looking at univariate time series prediction. It is the logical next step to investigate multi-variate time series as well. It would then be interesting to see how the models perform when multiple characteristics vary within one multi-variate time series dataset.

Author Contributions: Conceptualization, P.S.; methodology, P.S., M.D. and A.N.; data curation, M.D.; validation, P.S., M.D., A.N., D.Z. and B.E.; formal analysis, P.S., A.N. and D.Z.; writing—original draft preparation, P.S.; writing—review and editing, P.S.; visualization, P.S.; supervision, B.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The code for our training procedure and the synthesis of the custom datasets with controlled characteristics is publicly available on GitHub at https://github.com/MischaD/Benchmarking-Univariate-Time-Series-Prediction, accessed on 5 September 2023.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. He, X. A Survey on Time Series Forecasting. In *Proceedings of the 3D Imaging—Multidimensional Signal Processing and Deep Learning*; Springer Nature: Berlin/Heidelberg, Germany, 2023; pp. 13–23. [CrossRef]
2. Torres, J.F.; Hadjout, D.; Sebaa, A.; Martínez-Álvarez, F.; Troncoso, A. Deep Learning for Time Series Forecasting: A Survey. *Big Data* **2021**, *9*, 3–21. [CrossRef]
3. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is All you Need. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Sydney, Australia, 2017; Volume 30.
4. Wen, Q.; Zhou, T.; Zhang, C.; Chen, W.; Ma, Z.; Yan, J.; Sun, L. Transformers in Time Series: A Survey. *arXiv* **2023**, arXiv:2202.07125.
5. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
6. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
7. Koprinska, I.; Wu, D.; Wang, Z. Convolutional Neural Networks for Energy Time Series Forecasting. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8, ISSN: 2161-4407. [CrossRef]
8. Nassar, L.; Okwuchi, I.E.; Saad, M.; Karray, F.; Ponnambalam, K.; Agrawal, P. Prediction of Strawberry Yield and Farm Price Utilizing Deep Learning. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; pp. 1–7, ISSN: 2161-4407. [CrossRef]
9. Wu, N.; Green, B.; Ben, X.; O'Banion, S. Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case. *arXiv* **2020**, arXiv:2001.08317.
10. Wu, H.; Xu, J.; Wang, J.; Long, M. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Sydney, Australia, 2021; Volume 34, pp. 22419–22430.
11. Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; Zhang, W. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 11106–11115. [CrossRef]
12. Kitaev, N.; Kaiser, L.; Levskaya, A. Reformer: The Efficient Transformer. *arXiv* **2020**, arXiv:2001.04451.

13. Li, S.; Jin, X.; Xuan, Y.; Zhou, X.; Chen, W.; Wang, Y.X.; Yan, X. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Sydney, Australia, 2019; Volume 32.
14. Agarwal, K.; Dheekollu, L.; Dhama, G.; Arora, A.; Asthana, S.; Bhowmik, T. Deep Learning based Time Series Forecasting. In Proceedings of the 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 14–17 December 2020; pp. 859–864.
15. Hewamalage, H.; Bergmeir, C.; Bandara, K. Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *Int. J. Forecast.* **2021**, *37*, 388–427. [CrossRef]
16. Barić, D.; Fumić, P.; Horvatić, D.; Lipic, T. Benchmarking Attention-Based Interpretability of Deep Learning in Multivariate Time Series Predictions. *Entropy* **2021**, *23*, 143. [CrossRef] [PubMed]
17. Whang, S.E.; Roh, Y.; Song, H.; Lee, J.G. Data collection and quality challenges in deep learning: A data-centric AI perspective. *VLDB J.* **2023**, *32*, 791–813. [CrossRef]
18. Hegde, C. Anomaly Detection in Time Series Data using Data-Centric AI. In Proceedings of the 2022 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 8–10 July 2022; pp. 1–6, ISSN: 2766-2101. [CrossRef]
19. Mazumder, M.; Banbury, C.; Yao, X.; Karlaš, B.; Rojas, W.G.; Diamos, S.; Diamos, G.; He, L.; Parrish, A.; Kirk, H.R.; et al. DataPerf: Benchmarks for Data-Centric AI Development. *arXiv* **2023**, arXiv:2207.10062.
20. Devarajan, H.; Zheng, H.; Kougkas, A.; Sun, X.H.; Vishwanath, V. DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications. In Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 10–13 May 2021; pp. 81–91. [CrossRef]
21. Lim, B.; Zohren, S. Time-series forecasting with deep learning: A survey. *Philos. Trans. R. Soc. A* **2021**, *379*, 20200209. [CrossRef] [PubMed]
22. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
23. Bai, S.; Kolter, J.Z.; Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv* **2018**, arXiv:1803.01271.
24. Bergstra, J.; Yamins, D.; Cox, D. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In Proceedings of the 30th International Conference on Machine Learning, Atlanta, GA, USA, 17–19 June 2013; PMLR; pp. 115–123, ISSN: 1938-7228.
25. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. *arXiv* **2019**, arXiv:1907.10902.
26. Jones, G.R.D. Critical difference calculations revised: Inclusion of variation in standard deviation with analyte concentration. *Ann. Clin. Biochem.* **2009**, *46*, 517–519. [CrossRef] [PubMed]
27. Greff, K.; Srivastava, R.K.; Koutník, J.; Steunebrink, B.R.; Schmidhuber, J. LSTM: A Search Space Odyssey. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *28*, 2222–2232. [CrossRef] [PubMed]
28. Wirth, R.; Hipp, J. CRISP-DM: Towards a standard process model for data mining. In Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining, Lyon, France, 13–16 September 2000; Volume 1, pp. 29–39.
29. Mudelsee, M. Trend analysis of climate time series: A review of methods. *Earth-Sci. Rev.* **2019**, *190*, 310–322. [CrossRef]
30. Cooley, J.W.; Lewis, P.A.W.; Welch, P.D. The Fast Fourier Transform and Its Applications. *IEEE Trans. Educ.* **1969**, *12*, 27–34. [CrossRef]
31. Shumway, R.H.; Stoffer, D.S. Time Series Regression and Exploratory Data Analysis. In *Time Series Analysis and Its Applications: With R Examples*; Springer Texts in Statistics; Springer: Berlin/Heidelberg, Germany, 2006; pp. 48–83. [CrossRef]
32. Lara-Benítez, P.; Carranza-García, M.; Luna-Romera, J.M.; Riquelme, J.C. Temporal Convolutional Networks Applied to Energy-Related Time Series Forecasting. *Appl. Sci.* **2020**, *10*, 2322. [CrossRef]
33. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the 30th International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; PMLR; pp. 1310–1318, ISSN: 1938-7228.
34. Voyant, C.; Notton, G.; Duchaud, J.L.; Gutiérrez, L.A.G.; Bright, J.M.; Yang, D. Benchmarks for solar radiation time series forecasting. *Renew. Energy* **2022**, *191*, 747–762. [CrossRef]
35. Bauer, A.; Züfle, M.; Eismann, S.; Grohmann, J.; Herbst, N.; Kounev, S. Libra: A Benchmark for Time Series Forecasting Methods. In Proceedings of the ACM/SPEC International Conference on Performance Engineering, Virtual Event, France, 19–23 April 2021; ACM: New York, NY, USA, 2021; pp. 189–200. [CrossRef]