LOCALLY CONNECTED ECHO STATE NETWORKS FOR TIME SERIES FORECASTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Echo State Networks (ESNs) are a class of recurrent neural networks in which only a small readout regression layer is trained, while the weights of the recurrent network, termed the reservoir, are randomly assigned and remain fixed. Our work introduces the Locally Connected ESN (LCESN), a novel ESN variant with a locally connected reservoir, forced memory, and a weight adaptation strategy. LCESN significantly reduces the asymptotic time and space complexities compared to the conventional ESN, enabling substantially larger networks. LCESN also improves the memory properties of ESNs without affecting network stability. We evaluate LCESN's performance on the NARMA10 benchmark task and compare it to state-of-the-art models on nine real-world datasets. Despite the simplicity of our model and its one-shot training approach, LCESN achieves competitive results, even surpassing several state-of-the-art models. LCESN introduces a fresh approach to real-world time series forecasting and demonstrates that large, well-tuned random networks can rival complex gradient-trained models. Additionally, we provide a GPU-based implementation of LCESN as an open-source library.

1 INTRODUCTION

The field of time series forecasting (TSF) is currently dominated by feedforward, gradient-based neural networks, often based on the Transformer architecture introduced by Vaswani et al. (2017). Recurrent neural networks (RNNs) lag behind feedforward models to the point where they are only rarely compared in state-of-the-art papers (Nie et al., 2023). Despite being a natural fit for processing and predicting time series data, RNNs tend to be challenging to train (Pascanu et al., 2013).

Multiple approaches have been developed to avoid gradient pitfalls and facilitate the training of recurrent networks, with Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) being one of the most popular methods. A lesser-known approach is the Echo State Network (ESN) introduced by Jaeger (2001), which completely avoids gradient-based training by using a large random recurrent network called a *reservoir* that remains untrained. The only trained component in an ESN is a small linear readout layer computed via linear regression. The basic idea is that the random reservoir generates a wide variety of complex combinations of input data, making the extraction of almost any desired output a simple task.

The main advantages of ESNs over gradient-trained models are resistance to overfitting, a fast (one-shot) training procedure, and a simple architecture (Lukoševičius, 2012; Jaeger, 2001). Since the ESN is not trained via gradient descent, it can incorporate non-differentiable components and formulas with unstable gradients. ESNs have been successfully used for sequence prediction and sequence-to-sequence tasks, but on large real-world datasets, they have been overshadowed by modern large-scale gradient-based networks. Over the years, many authors have suggested improvements in reservoir topology and neuron design. In contrast, Matzner (2022) demonstrated that even a basic ESN can surpass those improvements when its hyperparameters are properly tuned. Unfortunately, even optimized ESNs have struggled to compete with state-of-the-art models.

We have identified the following two limitations of conventional ESN approaches. The first is the quadratic time and space complexity of each ESN step, which, in practical terms, limits the network size to hundreds or a few thousand neurons. The second is that reservoirs with long memory make the network unstable and oversensitive to hyperparameter perturbations.

1.1 RESEARCH OBJECTIVE

Our goal is to address the limitations of ESNs and make them competitive with state-of-the-art models on real-world datasets. Rather than aiming for top rankings in every benchmark, we aim to create a baseline recurrent model with a distinct architecture significantly different from the industry standard. Furthermore, we deliberately choose to provide a model that can be built from scratch on consumer hardware in a reasonable time.

2 RELATED WORK

Feedforward neural networks (multilayer perceptrons; MLPs) and recurrent neural networks (RNNs) have both been used in time series forecasting (TSF) (Zhang et al., 1998; Lipton et al., 2015) and are traditionally trained using gradient-based methods. However, as the context length and network size increase, the gradient magnitude can decrease or increase uncontrollably. This effect is known as the *vanishing and exploding gradient*, and it complicates the training process for deep models (Pascanu et al., 2013). To overcome these issues, Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and its variants, such as Random Connectivity LSTM (RCLSTM) (Hua et al., 2019), have been developed to facilitate long-term information retention. Despite their advantages, these models are known for their computational complexity. Jaeger (2001) introduced the Echo State Network (ESN) designed to mitigate the high computational costs and the issues of vanishing and exploding gradients. ESNs are described in detail in Section 3.1.

Following the tremendous success of Transformer-based models in natural language processing (Vaswani et al., 2017), Transformer-based models have been successfully adapted for TSF. iTransformer (Liu et al., 2024) applies independent attention across variates to build time series representations, which are combined using a feedforward network. PatchTST (Nie et al., 2023) leverages self-supervised pre-training to represent fixed-length patches, enabling compressed input processing and allowing efficient training while incorporating a longer history.

Multiple recent methods have explored time series decomposition, such as the Fourier transform employed by FEDFormer (Zhou et al., 2022) and TimesNet (Wu et al., 2023). While FEDFormer combines a transformer with a seasonal-trend decomposition method, TimesNet transforms the original 1-D input into a 2-D representation processed by 2-D kernels. AutoFormer (Wu et al., 2021) introduces a novel decomposition architecture featuring an autocorrelation mechanism that discovers dependencies at the subseries level.

Notably, Zeng et al. (2023) introduced a simple univariate one-layer linear model DLinear that surprisingly outperformed the state-of-the-art transformer models by a large margin. RLinear (Li et al., 2023) extended the linear approach of DLinear with RevIn (Kim et al., 2022), a model-agnostic normalization-denormalization method capable of eliminating the distribution shift within time series.

Finally, the recent introduction of TSMixer (Chen et al., 2023) marks a significant advancement, surpassing the aforementioned models. TSMixer alternates between time-mixing and feature-mixing MLPs, akin to the transposition processes of iTransformer. However, it uses MLPs instead of transformers with complex attention mechanisms.

3 METHODS

3.1 ECHO STATE NETWORKS

A basic Echo State Network with n neurons, n_{in} inputs, and n_{out} outputs consists of a *reservoir* represented by a connectivity matrix $W \in \mathbb{R}^{n\times n}$, an *input weight matrix* $W_{\text{in}} \in \mathbb{R}^{n\times n_{in}}$, a *readout weight matrix* $W_{out} \in \mathbb{R}^{n_{out}\times n}$, and a *feedback weight matrix* $W_{fb} \in \mathbb{R}^{n\times n_{out}}$ (see Figure 1). Neuron activations at time t are denoted by $a(t) \in \mathbb{R}^n$, the input values by $u(t) \in \mathbb{R}^{n_{in}}$, the output values by $x(t) \in \mathbb{R}^{n_{out}}$, and the target by $y(t) \in \mathbb{R}^{n_{out}}$. The following recurrence represents one discrete *step* of an ESN, transitioning it to the next time point:

$$\begin{aligned} z(t) &= Wa(t-1) + W_{in}u(t) + W_{fb}x(t-1) + \mu_b\,, \\ a(t) &= (1-\gamma)a(t-1) + \tanh(z(t))\,, \\ x(t) &= W_{out}a(t)\,, \end{aligned} \tag{1}$$

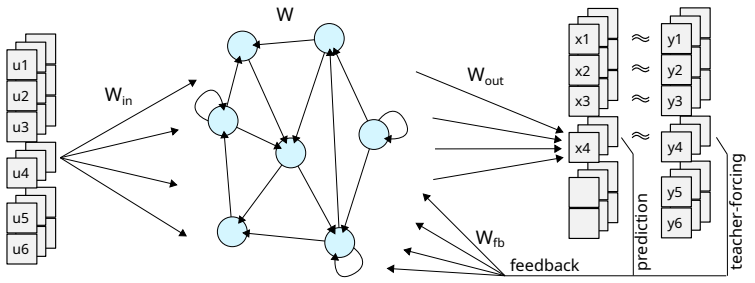

Figure 1: Overview of an Echo State Network.

where $z(t) \in \mathbb{R}^n$ represents the pre-activation potentials before being processed by the tanh activation function, $\gamma \in \mathbb{R}$ is the *leakage* parameter, and $\mu_b \in \mathbb{R}$ denotes a constant *bias* shared by all neurons.

The initial activation $a(0)$ is set to zero. The reservoir weights W are generated from the uniform distribution $\mathrm{U}(\mu_{res} - \sigma_{res}, \mu_{res} + \sigma_{res})$, the input weights for the i-th input W_{in}^i from $U(-\sigma_{in}^i, \sigma_{in}^i)$, and the feedback weights for the j-th output W_{fb}^j from the uniform distribution $U(-\sigma_{fb}^j, \sigma_{fb}^j)$, for hyperparameters $\mu_{res} \in \mathbb{R}$, $\sigma_{res}, \sigma_{in}^i, \sigma_{fb}^j \in \mathbb{R}_+$. Note that we specify the parameters of uniform distribution using its mean and spread, rather than its borders, to smooth the fitness space for hyperparameter optimization. After feeding the training data into the network, the output matrix W_{out} is trained using ridge regression to minimize the squared error between the predicted sequence $x(t)$ and the target sequence $y(t)$ (more details in Appendix C). The network output is undefined until the training procedure sets W_{out}. Until then, the feedback connections are driven by the target sequence $y(t)$. Using the target sequence instead of the network's own output is known as *teacher forcing* (Jaeger, 2001; Lukoševičius, 2012).

ESNs represent a dynamical system whose behavior can range from a stable, unchanging state to a chaotic white noise generator, depending on their configuration (e.g., σ_{res}). Between these two regimes lies the so-called *edge of chaos*, which, according to some authors, is the regime where the network performs best (Bertschinger and Natschläger, 2004; Matzner, 2017; Boedecker et al., 2011). In order to avoid the chaotic regime, the network has to exhibit the *echo state property* (also called *fading memory*), meaning that its state depends only on a finite history of its inputs (Jaeger, 2001). One of the metrics used to measure chaoticity is called the Lyapunov exponent, which can be estimated by introducing a tiny perturbation into the reservoir and measuring how much the perturbed and unperturbed networks diverge. Lyapunov exponent ranges from -1 to 1, where negative values denote order, positive values denote chaos, and zero indicates the edge of chaos. For more details on its properties and the estimation algorithm, see Sprott (2003).

3.2 Hyperparameter optimization

We use the hyperparameter tuning framework by Matzner (2022), based on the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001). For more details on CMA-ES, see Hansen (2016), and consult Matzner (2022) for its use in ESN hyperparameter tuning.

The optimized hyperparameters include the reservoir scaling parameters σ_{res}, μ_{res}, the bias term μ_b, the input scaling parameters σ_{in}^i for each input, the feedback scaling parameters σ_{fb}^j for each output (only for sequence-to-sequence tasks), and the ridge regression coefficient $\lambda_2 \in \mathbb{R}_+$. In total, this amounts to $4 + n_{in} + n_{out}$ hyperparameters, where n_{in} and n_{out} are the numbers of inputs and outputs, respectively, and depend on the task. For details on optimizer settings and parameter limits, please refer to Appendix D.

3.3 Local topology

Conventional ESNs introduced by Jaeger (2001) use a fully connected reservoir, making the state transition computationally expensive in both time and memory. Some authors have proposed reservoir topologies with lower time complexity, such as ring or chain topologies (Rodan and Tino, 2011), but these do not provide the same performance as conventional fully connected or sparse ESNs (Matzner,

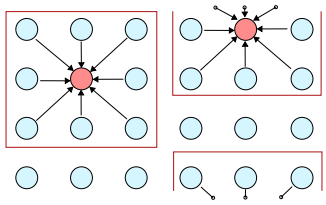

Figure 2: Demonstration of local topology on a grid of 4×6 neurons and 3×3 kernel. Two non-overlapping kernels are visualized; one wraps around the top edge.

2022). Analysis of reservoirs designed using the HyperNEAT evolutionary algorithm (Stanley et al., 2009) suggested that placing neurons on a grid and connecting each neuron only within its neighborhood may be beneficial for increasing the network's stability (Matzner, 2017). A similar idea has also been proposed under the term *small-world* connectivity (Watts and Strogatz, 1998), where, in addition to neighborhood connections, neurons occasionally form longer connections, thus reducing the network's diameter. Small-world reservoirs have been studied, e.g., in the fields of biologically plausible networks and neuromorphic hardware, where they provide better wiring cost than fully connected networks (Kawai et al., 2017; Suárez et al., 2021; Daniels et al., 2022).

We propose a local topology that reduces the asymptotic time complexity of the ESN step without compromising accuracy. This allows for larger reservoirs at the same computational cost. Neurons are placed on a torus-shaped grid, limiting the maximum distance between them, similar to small-world connectivity. Each neuron is connected to all neurons within its $K \times K$ neighborhood (see Figure 2). Adopting the notation of convolutional neural networks, we term the weights for this neighborhood a *kernel*. However, contrary to convolutional layers, each neuron has its own kernel, and no weights are shared. The weights are stored in a tensor $W_k \in \mathbb{R}^{N \times M \times K \times K}$, where N×M represents the reservoir dimensions (i.e., N rows and M columns, totaling $NM = n$ neurons). With a fully connected reservoir, the weights were stored in a larger matrix $W \in \mathbb{R}^{N \times M \times N \times M}$. Therefore, assuming a naive implementation of matrix multiplication, this technique reduces both the time and space complexity of the ESN step from $\mathcal{O}((NM)^2)$ to $\mathcal{O}(NMK^2)$. Furthermore, the proposed topology is designed with GPUs in mind, allowing efficient (coalesced) access to GPU memory and avoiding the overhead of generic sparse matrix computation. According to our measurements, our GPU implementation provides up to 15x speedup compared to a fully connected reservoir on a consumer-grade GPU for a network of size 80×100 and 7×7 kernel. For details on GPU implementation and comparison with other methods, please refer to Appendix F.

It may be tempting to use the same kernel for all neurons and turn the local topology into an even more efficient "recurrent convolutional network". Unfortunately, as presented in Appendix E.3, using convolutional kernels does not provide the same performance as the proposed topology.

3.4 Forced memory

The stability of ESNs is intrinsically linked to their memory capacity. In a conventional ESN, the network's entire memory is implicitly encoded within its state, propagating sequentially from one time step to the next. However, since the reservoir is composed of randomly initialized weights, it does not differentiate between useful information and noise present in the data. As the memory capacity of the network increases, its sensitivity to both signal and noise grows accordingly. Eventually, this heightened sensitivity can lead to the network being overwhelmed by noise, pushing it into a chaotic state. To prevent the network from crossing the edge of chaos, memory capacity must be limited, which unfortunately restricts the network's ability to retain a longer context.

Many authors have proposed methods to improve memory capacity (MC). To give a few examples, Holzmann and Hauser (2010) added trainable delays between the reservoir and the output layer, Echo Memory-Augmented Network (EMAN) by Ma et al. (2021) employs a complex low-learning weight attention mechanism, and Distance-Delayed-Networks (DDN) by Iacob and Dambre (2024) rely on physics-inspired distance-based delays. Our approach is more straightforward. It simply gives each neuron a lookback connection to its historical state at a random time horizon. Despite its simplicity,

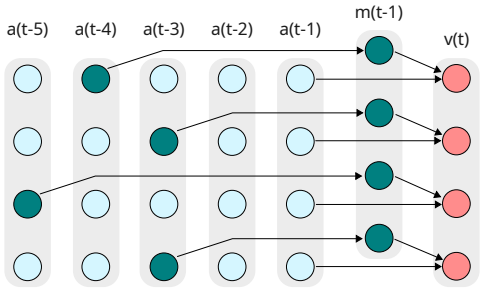

Figure 3: An example of forced memory with maximum delay $H = 5$. The vector $v(t)$, combining memory $m(t-1)$ with last state $a(t-1)$, is used instead of $a(t-1)$ during state transition.

this method avoids the need to propagate the entire memory through every step and helps the network to stay at the stable regime.

Formally, in *memory forcing*, the activation $a(t)_i$ of neuron i at time t is directly combined with its historical activation $a(t-h_i)_i$. The delay h_i and the combination coefficient $(w_{mem})_i$ are randomly chosen for each neuron separately and remain fixed. The recurrence equations (1) are modified as follows, and Figure 3 illustrates this process.

$$\begin{aligned} v(t) &= w_{mem} \circ m(t-1) + (1 - w_{mem}) \circ a(t-1)\,, \\ z(t) &= W v(t) + W_{in} u(t) + W_{fb} x(t-1) + \mu_b\,, \\ a(t) &= \tanh(z(t))\,, \\ x(t) &= W_{out} a(t)\,, \\ m(t)_i &= a(t-h_i)_i \quad \forall i \in \{1, 2, \ldots, n\}\,, \end{aligned} \tag{2}$$

where $\circ$ denotes elementwise multiplication, $m(t) \in \mathbb{R}^n$ is the *memory vector* with each element equal to a historical activation, $w_{mem} \in \mathbb{R}^n$ represents the preference of the historical state over the current state, and $v(t) \in \mathbb{R}^n$ denotes the state combined with its memory. The coefficients w_{mem} for the affine combination are generated from a uniform distribution $U(-1, 1)$. The delays h_i are initialized as random integers from a uniform distribution $U(0, H-1)$, where H denotes the maximum allowed delay (set to 100 in our experiments). Note that we have omitted the leakage coefficient γ because its role is fulfilled by the memory vector; in fact, leakage can be simulated by setting $h_i = 1$ and $(w_{mem})_i = \gamma$.

3.5 Prediction methodology

To be comparable to the state-of-the-art feedforward models in TSF benchmarks, we have used the same training/validation/testing split and the same data normalization technique (Nie et al., 2023). However, the use of recurrent networks fundamentally differs from that of feedforward models. Feedforward models expect an input of fixed length L_{in} and predict an output of fixed length L_{out}. The dataset is created by extracting patches of length $L_{in} + L_{out}$ from the original time series and feeding them to the model independently. Having a fixed input and output length has a few benefits, such as the ability to process many subsequences as a single GPU batch and the flexibility to feed random subsequences to the model until training convergence. However, it also comes with a few downsides. For instance, since the patches are independent, it is often necessary to encode the position of the patch within the whole sequence (e.g., the day, week, and year in the case of real-world datasets). Additionally, a separate model must be trained for each prediction horizon L_{out}.

RNNs, on the other hand, only predict the immediately following time point, and the user can repeat this step as many times as desired to predict the whole horizon. Unlike feedforward models, ESNs inherently track their position in the input sequence, removing the need for explicit position encoding due to their potentially unlimited context length. In fact, we do not enhance the input data in any way; we pass them to the network in their raw form. However, a natural downside of recurrent models with long memory is the necessity of feeding the entire time series up to the moment of prediction (or at least a substantial part of it). Fortunately, in the case of ESNs, training and prediction are rather fast, and we do not need to propagate any gradient through the whole sequence.

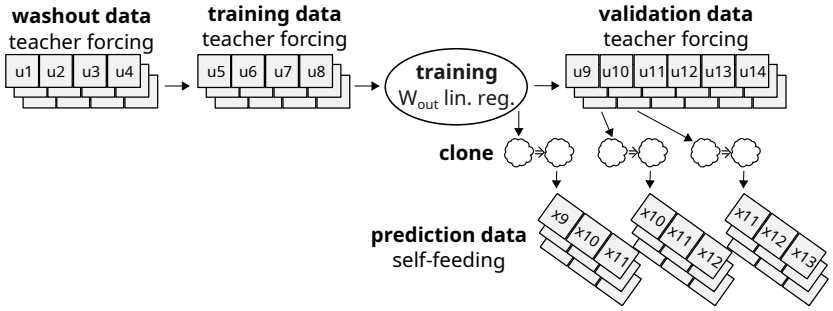

Figure 4: Overview of the prediction procedure.

Ultimately, for a fair comparison with the state of the art, we need to evaluate our model on all $\ell - L_{out} + 1$ subsequences of the testing data while running solely a self-feeding loop (without the ground truth data), where ℓ is the length of the test data. The testing procedure is demonstrated in Figure 4 and works as follows. The network is fed with the washout sequence, and its outputs and states are ignored. Afterwards, the network is fed with the training data (teacher forced) up to the beginning of the first validation subsequence. Next, the output weights are trained via linear regression (as described in Section 3.1). The network is now ready to be used for a self-feeding prediction loop.

A temporary clone of the network is created and driven by its own output for L_{out} steps. That is, at time t, it uses its own output $x(t-1)$ as input. Its outputs are recorded and compared with the target sequence. The clone is destroyed, and the original network is fed with the next time point (one step with teacher forcing). This process of cloning, prediction, and updating is repeated until the cloned network reaches the end of the validation sequence.

An important topic is the output weights (W_{out}) adaptation strategy when feeding the validation data with teacher forcing. Let us list a few options ordered by their computational and memory complexities and their notation used throughout the paper.

1. **LCESN:** Keep the output weights fixed after the initial training.
2. **LCESN-LMS:** Update the weights via a simple signal filter. There are many approaches addressing the online adaptation of ESNs, such as the popular Recursive Least Squares method called FORCE (Sussillo and Abbott, 2009), more computationally effective Backpropagation-Decorrelation (Steil, 2004), and many others (Lukoševičius, 2012). The evaluation of all existing methods is beyond the scope of this paper, and we chose Normalized Least Mean Squares (NLMS) (Haykin, 2002) as a minimalist baseline.
3. **LCESN-LR100:** Inherit the filtering from LCESN-LMS, and recalculate the full linear regression after every 100 steps. This option balances speed and accuracy, and the results will later show that a shorter interval would bring about a negligible effect.
4. **LCESN-LR1:** Recompute the linear regression before every self-feeding prediction loop. This option requires impractical computational resources, but represents an upper bound on the accuracy of online learning.

Note that all of these methods use the same hyperparameter-optimized network and only differ in the evaluation procedure. Therefore, there is no need to repeat the hyperparameter optimization, which is the most time-consuming part of the process. In Section 6, we will present practical measurements on real hardware.

4 DATASETS

We use the NARMA10 dataset, widely used in ESN literature, along with nine real-world datasets commonly utilized in state-of-the-art sequence forecasting studies. From longest to shortest, these datasets are ETTm1, ETTm2 (Electricity Transformer Temperature), Weather, Solar Energy, Electricity, Traffic, ETTh1, ETTh2, and Exchange. We focus only on the multivariate case, i.e., predicting all available features. For a more detailed description of the datasets, please refer to Appendix A.

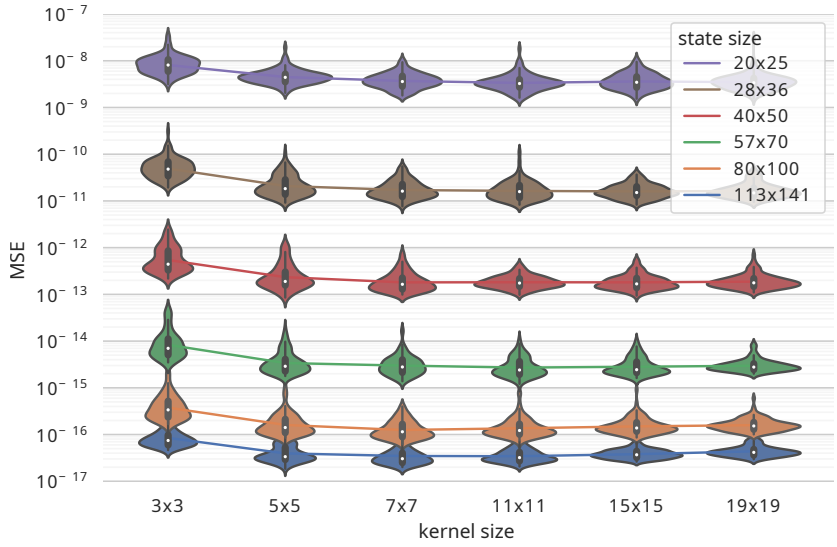

Figure 5: Comparison of various network and kernel sizes on NARMA10 benchmark tasks. Each violin represents the evaluation of 100 unique random sequences predicted by a network selected as the best of five optimization runs.

5 Experiment

First, we test the basic LCESN without forced memory and weight adaptation on the NARMA10 benchmark to demonstrate the effect of the local topology. Second, we evaluate the performance of LCESN in practical terms of consumed time and memory resources on a consumer-grade GPU. Third, we evaluate the effect of forced memory on ETTm datasets and compare LCESN with the conventional ESN. Finally, we compare the four full-featured LCESN variants presented in Section 3.5 with state-of-the-art models in time series forecasting on real-world datasets.

To enable building our model from scratch on consumer hardware, we limit each hyperparameter optimization run to 2000 evaluations, ensuring it fits within a 24-hour deadline on an older NVIDIA GTX 2080 Ti GPU (released in 2018) and a single core of an average desktop CPU. For a fair comparison, we use the same hyperparameter optimization technique for baseline ESN and for the proposed LCESN models. Further technical details of the experimental setup are provided in Appendix D.

6 Results and Discussion

6.1 Effect of Local Topology

We evaluated a grid of network sizes from 1,000 to 16,000 neurons and kernel sizes from 3×3 to 19×19 on the widely used NARMA10 benchmark task. The model used is the vanilla LCESN without forced memory and weight adaptation (i.e., a conventional ESN with local topology). As demonstrated in Figure 5, when the network size increases, the error on NARMA10 drops by orders of magnitude. Furthermore, the kernel size plays a much less significant role, with 7×7 kernel representing the best accuracy while still providing acceptable computational demands. An important observation is that increasing the kernel size further, and thus approaching the fully connected conventional ESN, has not provided any benefit. In fact, the results with the 19×19 kernel are statistically significantly worse ($p < 0.05$) compared to the 7×7 kernel for the two largest networks.

The results suggest that on the NARMA10 benchmark task, the widely used network sizes of 100 to 1,000 neurons are simply too small, and with proper hyperparameters, the performance depends mostly on the size of the network. Needless to say, the error on NARMA10 is below an interesting threshold, and it should no longer be used to compare state-of-the-art results in the Echo State

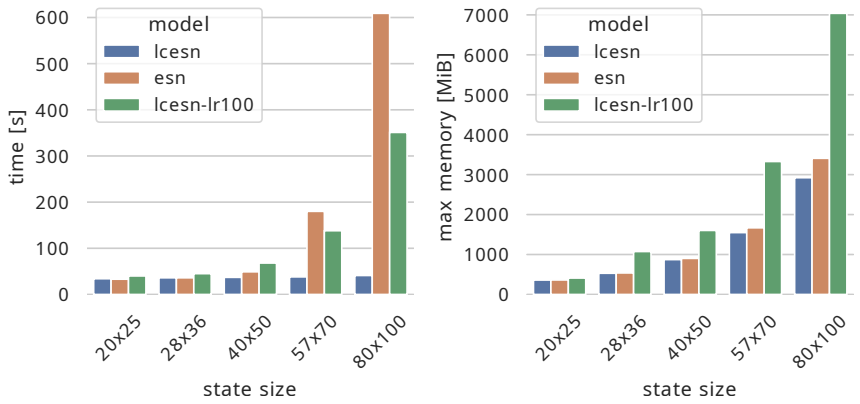

Figure 6: Wall-clock time and memory resources required to evaluate the entire ETTh1 dataset using the models presented in Section 3.5 with 7×7 kernel and known hyperparameters. The measurements encapsulate all the evaluation phases: generating the network, the washout period, linear regression training, and test set prediction. The ESN model represents the conventional ESN with a full connectivity matrix; LCESN reduces it to local topology and adds forced memory; and LCESN-LR100 additionally performs regression retraining every 100 teacher forced steps.

Network literature. However, it remains a convenient, hard-to-overfit benchmark for checking basic properties of ESNs and validating new ideas for sequence-to-sequence tasks.

Note that as the network size increases, the improvement gradually diminishes. One possible explanation is that the size of the training data (12,000 steps) is insufficient for the largest network (16,000 neurons). However, further increasing the size of the training set has not significantly improved the results (see Appendix E.1 for details). For comparing various state sizes and alternative topologies, such as convolution, please refer to Appendices E.2 and E.3.

6.2 Benchmarking

The effect of the lower time complexity of the local topology is demonstrated in Figure 6. Notably, in the case of large networks, the gain from the faster LCESN step fully compensates for the time needed to recalculate the linear regression every 100 steps (LCESN-LR100). However, the recalculation consumes a lot of memory resources; therefore, it must be used with caution on large networks and datasets. The measurements were performed on an NVIDIA GTX 1080 Ti GPU (released in 2017) supported by a single core of an Intel i7-4770 CPU. Training and evaluation took under 40 seconds for a 40x50 network.

Building the network from scratch also requires careful tuning of its hyperparameters (Matzner, 2022). In all our experiments and for all tested models, the optimizer is limited to 2,000 full evaluation cycles. Each cycle consists of generating a new network, performing regression training, and predicting the validation sequence. In the case of the ETTh1 dataset, this procedure took less than four hours. Each of our models is selected as the best from five hyperparameter optimization runs (on the validation set). Therefore, it took 20 hours in total to train the ETTh1 model.

For a more detailed decomposition and comparison with other models, please refer to Appendix G.

6.3 Effect of forced memory

In this experiment, we compared conventional ESN without forced memory and LCESN with increasing forced memory horizons. None of the tested models uses weight adaptation, meaning that the LCESN without forced memory denotes an ESN with local topology. The hyperparameters for all models, including the baseline ESN, are optimized using the same procedure described in Section 3.2.

Figure 7 shows the validation errors of five separate hyperparameter optimization runs on the ETTm1 dataset. Disabling forced memory or setting its maximum horizon to less than 50 steps increases noise, hindering the hyperparameter optimizer's ability to estimate the gradient and leading to substantially

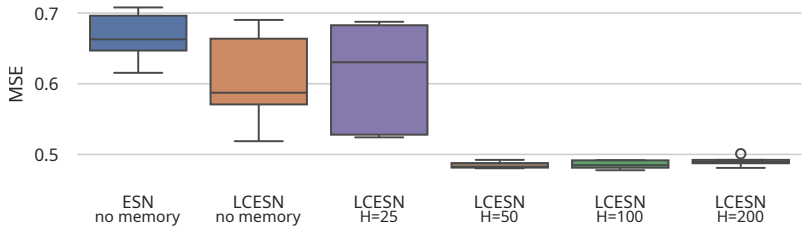

Figure 7: Comparison of the final MSE reached by the hyperparameter optimization on the validation sequence for five separate runs on the ETTm1 dataset. H denotes the forced memory horizon limit. The reservoir size is 40×50 with a 7×7 kernel.

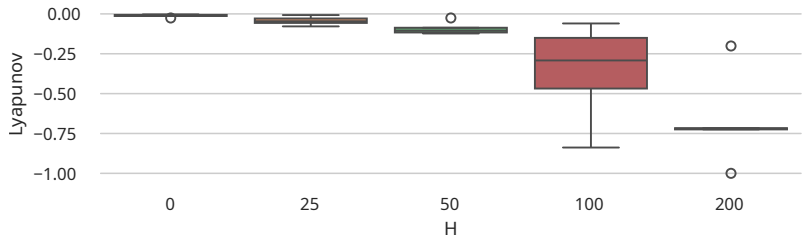

Figure 8: Lyapunov exponent versus forced memory horizon for LCESN runs evaluated in Figure 7

worse results with higher variance. Local topology is only supposed to reduce computational costs, not increase the accuracy. Therefore, unsurprisingly, the ESN baseline has performed similarly to LCESN with short memory horizons.

Figure 8 demonstrates the relation of forced memory and the network dynamics. Limiting the forced memory horizon clearly pushes the network toward the edge of chaos. In contrast, longer forced memory horizons allow the network to stay in a stable regime while still benefiting from long memory. The right choice of maximum memory horizon depends on the dataset at hand. According to Figure 7, for the ETTm1 dataset, a 50-step horizon limit would likely suffice, but other datasets may require longer memory. Therefore, we will use a memory horizon limit of 100 steps for the rest of this work.

6.4 Real-world datasets

We adopted the results of third-party models from Nie et al. (2023) and Liu et al. (2024), except for the results of the more recent TSMixer, which are obtained directly from its original paper Chen et al. (2023). Furthermore, Liu et al. (2024) only provide results for a prediction length of 96 time points, whereas Nie et al. (2023) provide stronger baselines by selecting the best results from six different prediction lengths for the Transformer-based models. When both baselines are available, we opt for the stronger results from Nie et al. (2023). TSMixer was not tested on the Solar Energy and Exchange datasets in its original paper, so those results are omitted from the comparison table, and TSMixer is not considered in their ranking. However, since it was the overall best-ranked model, it is possible that it would also perform well on the omitted datasets.

The comparison with existing third-party models and conventional ESN is provided in Table 1. For complete results with all prediction horizons and for relative ranking between models, see Appendix B. All four introduced LCESN variants improved upon the conventional ESN on all datasets, highlighting the effect of forced memory. The three weight adaptation methods implemented by LCESN-LMS, LCESN-LR100, and LCESN-LR1 notably improve the results of the basic LCESN. Therefore, if the data frequency and computational resources allow, as in the case of the evaluated real-world datasets, regular retraining can yield better performance. LCESN-LR1, however, did not significantly improve upon the results of LCESN-LR100, so its high computational demands have not proven to be justifiable on the tested datasets. According to the relative ranking across all datasets and horizons (Appendix B.2), LCESN-LR1 clinched the second place and LCESN-LR100 the third place among the evaluated models.

Table 1: The results of the LCESN approach on real-world datasets compared to the results of other authors. Every number is the average over four prediction horizons 96, 192, 336, and 720. The best result is **red** and the second best is blue. The four LCESN variants and the ESN are not compared to each other. The reservoir size for ESN and LCESNs is 40×50 with a 7×7 kernel. Datasets are sorted from longest (top) to shortest (bottom).

Models	ESN (2001)		**LCESN Ours**		**LCESN-LMS Ours**		**LCESN-LR100 Ours**		**LCESN-LR1 Ours**		TSMixer (2023)		DLinear (2023)		PatchTST (2023)		iTransformer (2024)		FEDformer (2022)		TimesNet (2023)		RLinear (2023)		Autoformer (2021)	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1	0.540	0.527	0.412	0.430	0.364	0.393	0.362	0.392	0.359	0.390	**0.347**	**0.375**	0.403	0.407	0.387	0.400	0.407	0.410	0.382	0.422	0.400	0.406	0.414	0.408	0.515	0.493
ETTm2	0.380	0.424	0.283	0.354	0.275	0.332	**0.257**	0.326	**0.255**	0.326	0.267	**0.322**	0.350	0.401	0.281	0.326	0.288	0.332	0.292	0.343	0.291	0.332	0.286	0.327	0.310	0.357
Weather	0.255	0.298	0.232	0.279	0.236	0.272	**0.221**	**0.264**	**0.221**	**0.264**	0.224	**0.264**	0.265	0.317	0.258	0.280	0.258	0.278	0.310	0.357	0.259	0.286	0.272	0.291	0.335	0.379
Solar	0.268	0.384	**0.202**	0.277	**0.202**	0.268	**0.201**	0.273	**0.201**	0.274	N/A	N/A	0.330	0.401	0.270	0.307	0.233	**0.262**	0.292	0.381	0.301	0.319	0.369	0.355	0.885	0.711
Electricity	0.264	0.351	0.225	0.329	0.197	0.308	0.184	0.294	0.183	0.293	**0.160**	**0.257**	0.212	0.300	0.205	0.290	0.178	0.270	0.207	0.321	0.192	0.295	0.218	0.298	0.214	0.326
Traffic	1.095	0.533	0.882	0.407	0.845	0.402	0.787	0.392	0.782	0.392	**0.408**	0.284	0.624	0.383	0.481	0.304	0.428	**0.282**	0.604	0.372	0.620	0.336	0.626	0.378	0.616	0.384
ETTh1	0.650	0.602	0.818	0.662	0.516	0.510	0.537	0.518	0.541	0.521	**0.412**	**0.428**	0.456	0.452	0.469	0.454	0.454	0.448	0.428	0.454	0.458	0.450	0.446	0.434	0.473	0.477
ETTh2	0.559	0.529	0.438	0.464	0.372	0.419	0.378	0.421	0.381	0.422	**0.355**	0.400	0.559	0.515	0.387	0.407	0.383	0.406	0.388	0.434	0.414	0.427	0.374	**0.398**	0.422	0.443
Exchange	0.703	0.575	0.889	0.661	0.718	0.602	0.656	0.574	0.639	0.566	N/A	N/A	**0.354**	0.414	0.366	0.404	0.360	**0.403**	0.518	0.429	0.416	0.443	0.378	0.418	0.613	0.539
# 1st	0/9	0/9	1/9	0/9	1/9	0/9	3/9	1/9	3/9	1/9	**5/7**	**5/7**	1/9	0/9	0/9	0/9	0/9	3/9	0/9	0/9	0/9	0/9	0/9	1/9	0/9	0/9

LCESN variants were most successful on the ETTm1, ETTm2, Weather, and Solar Energy datasets, even clinching first place in several cases. One of the reasons for the success on those particular datasets might be their length, as they represent the four longest of the nine tested series, and the difference in dataset length is substantial. Each of the four listed datasets has more than 52,000 time points, while each of the remaining five has fewer than 27,000 time points. This suggests that to fully utilize the potential of LCESN, tasks need to provide a sufficient amount of training data. For more details on dataset properties, see Appendix A.

We used a medium-sized model of size 40×50 to reduce the experiment duration. Whether a smaller or larger network could further improve the results remains an open question.

7 Conclusion

In recent years, recurrent neural networks (RNNs) have been overshadowed by gradient-trained feed-forward architectures. Modern time series forecasting papers often do not even consider comparing with RNNs due to their lack of recent successes.

To revitalize research in this area, we revisited the Echo State Network (ESN) approach — an older, often overlooked yet promising method — and addressed two of its limitations. First, implementing a local topology allows for much larger networks with the same computational demands. Second, introducing forced memory facilitates longer time dependencies and enhances network stability.

The resulting Locally Connected Echo State Network (LCESN) is simple and does not use gradient descent. It is very fast to train and infer when its hyperparameters are known and can predict at arbitrary horizons. Although our goal was not to outperform all existing methods, LCESN has achieved competitive accuracy compared to state-of-the-art models, even surpassing them on some of the longest tested datasets.

The proposed model can be built from scratch on consumer hardware in less than 24 hours. The majority of this time is spent on hyperparameter optimization, which we identify as an area with significant potential for future improvement. When the hyperparameters are known, training and inference usually take tens of seconds or minutes, depending on the dataset size.

We made the source code in C++ and CUDA publicly available under a permissive license on our GitHub repository[1]. We used fixed random seeds and included logs and network checkpoints to ensure reproducibility.

[1]Link omitted for anonymity

REFERENCES

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers, 2023.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–1310–III–1318. JMLR.org, 2013.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148, 01 2001.

Mantas Lukoševičius. *A Practical Guide to Applying Echo State Networks*, pages 659–686. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_36.

Filip Matzner. Hyperparameter tuning in echo state networks. In *GECCO '22: Proceedings of the Genetic and Evolutionary Computation Conference*, page 404–412, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 978-1-4503-9237-2. doi: 10.1145/3512290.3528721.

Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35–62, 1998. ISSN 0169-2070. doi: 10.1016/S0169-2070(97)00044-7.

Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, 2015.

Yuxiu Hua, Zhifeng Zhao, Rongpeng Li, Xianfu Chen, Zhiming Liu, and Honggang Zhang. Deep learning with long short-term memory for time series prediction. *IEEE Communications Magazine*, 57(6):114–119, 2019. doi: 10.1109/MCOM.2019.1800155.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting. *arXiv preprint arXiv:2310.06625*, 2024.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *Proc. 39th International Conference on Machine Learning (ICML 2022)*, volume 162 of *Proceedings of Machine Learning Research*, pages 27268–27286. PMLR, 2022.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2D-variation modeling for general time series analysis. In *International Conference on Learning Representations*, 2023.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with Auto-Correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems*, pages 22419–22430, 2021.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence*, 37:11121–11128, 06 2023. doi: 10.1609/aaai.v37i9.26317.

Zhe Li, Shiyi Qi, Yiduo Li, and Zenglin Xu. Revisiting long-term time series forecasting: An investigation on linear mapping. *ArXiv*, abs/2305.10721, 2023.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Si-An Chen, Chun-Liang Li, Sercan O Arik, Nathanael Christian Yoder, and Tomas Pfister. TSMixer: An all-MLP architecture for time series forecast-ing. *Transactions on Machine Learning Research*, September 2023. ISSN 2835-8856.

Nils Bertschinger and Thomas Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural computation*, 16(7):1413–1436, 2004.

Filip Matzner. Neuroevolution on the edge of chaos. In *GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 465–472, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3071292.

Joschka Boedecker, Oliver Obst, Joseph Lizier, Norbert Mayer, and Minoru Asada. Information processing in echo state networks at the edge of chaos. *Theory in biosciences = Theorie in den Biowissenschaften*, 131:205–13, 12 2011. doi: 10.1007/s12064-011-0146-8.

J.C. Sprott. *Chaos and Time-series Analysis*. Oxford University Press, 2003. ISBN 9780198508403.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

Nikolaus Hansen. The cma evolution strategy: A tutorial. *ArXiv*, abs/1604.00772, 2016.

Ali Rodan and Peter Tino. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–144, 2011. doi: 10.1109/TNN.2010.2089641.

Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, 15(2):185–212, 04 2009. ISSN 1064-5462. doi: 10.1162/artl.2009.15.2.15202.

Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

Yuji Kawai, Tatsuya Tokuno, Jihoon Park, and Minoru Asada. Echo in a small-world reservoir: Time-series prediction using an economical recurrent neural network. In *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 126–131, 2017. doi: 10.1109/DEVLRN.2017.8329797.

Laura E. Suárez, Blake A. Richards, Guillaume Lajoie, and Bratislav Misic. Learning function from structure in neuromorphic networks. *Nature Machine Intelligence*, 3(9):771–786, Sep 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00376-1.

Ryan Daniels, J.B. Mallinson, Z.E. Heywood, P.J. Bones, M.D. Arnold, and Simon Brown. Reservoir computing with 3d nanowire networks. *Neural Networks*, 154, 07 2022. doi: 10.1016/j.neunet.2022.07.001.

Georg Holzmann and Helmut Hauser. Echo state networks with filter neurons and a delay&sum readout. *Neural Networks*, 23(2):244–256, 2010.

Qianli Ma, Zhenjing Zheng, Wanqing Zhuang, Enhuan Chen, Jia Wei, and Jiabing Wang. Echo memory-augmented network for time series classification. *Neural Networks*, 133:177–192, 2021. ISSN 0893-6080. doi: 10.1016/j.neunet.2020.10.015.

Stefan Iacob and Joni Dambre. Exploiting signal propagation delays to match task memory requirements in reservoir computing. *Biomimetics*, 9(6), 2024. ISSN 2313-7673. doi: 10.3390/biomimetics9060355. URL https://www.mdpi.com/2313-7673/9/6/355.

David Sussillo and L. F. Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63:544–557, 2009.

J.J. Steil. Backpropagation-decorrelation: online recurrent learning with o(n) complexity. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 2, pages 843–848 vol.2, 2004. doi: 10.1109/IJCNN.2004.1380039.

Simon Haykin. *Adaptive filter theory*. Prentice Hall, Upper Saddle River, NJ, 4th edition, 2002.

Tomoyuki Kubota, Kohei Nakajima, and Hirokazu Takahashi. Dynamical anatomy of narma10 benchmark task. *ArXiv*, abs/1906.04608, 2019.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference*, volume 35, pages 11106–11115. AAAI Press, 2021.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, page 95–104, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356572. doi: 10.1145/3209978.3210006.

Claudio Gallicchio and Alessio Micheli. Reservoir topology in deep echo state networks. In Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2019: Workshop and Special Sessions*, pages 62–75, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30493-5.

Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605587448.

Pavan Yalamanchili, Umar Arshad, Zakiuddin Mohammed, Pradeep Garigipati, Peter Entschev, Brian Kloppenborg, James Malcolm, and John Melonakos. ArrayFire - A high performance software library for parallel computing with an easy-to-use API, 2022. URL `https://github.com/arrayfire/arrayfire`.

Table 2: Sizes of the real-world datasets measured in the number of time steps.

	ETTm1/2	Weather	Solar	Electricity	Traffic	ETTh1/2	Exchange
Training	34560	36887	36792	18412	12280	8640	5311
Validation	11520	5270	5256	2632	1756	2880	760
Test	11520	10539	10512	5260	3508	2880	1517
Features	7	21	136	321	862	7	8

A DATASETS

A.1 NARMA10

In *Nonlinear Autoregressive Moving Average* of 10th order (NARMA10), each element is a nonlinear combination of the last 10 inputs and outputs. Formally, the target sequence is defined as follows:

$$y(t+1) = 0.3y(t) + 0.05y(t)\sum_{i=0}^{9} y(t-i) + 1.5u(t-10)u(t) + 0.1\,,$$

where $u(t)$ is the random input sequence generated from uniform distribution $U(0, 0.5)$ and $y(t)$ is the target sequence, both in time t. It should be mentioned that NARMA10 is a chaotic sequence that can diverge (Kubota et al., 2019), in which case it has to be regenerated.

We use a washout period of 1000 steps, a training period of 12000 steps, and a testing period of 1000 steps. For evaluation, we generate 100 unique NARMA10 sequences (using a different initial random seed).

A.2 REAL-WORLD DATASTETS

We use nine publicly available datasets ETTh1, ETTh2, ETTm1, and ETTm2 proposed by Zhou et al. (2021), weather by Wu et al. (2021), and electricity, traffic, solar-energy, and exchange datasets used by Lai et al. (2018). We focus only on the multivariate case, i.e., predicting all the provided features. For a brief description of the contents of the datasets, see Appendix A.1 from Liu et al. (2024).

To be comparable to other works, we normalize each covariate in the training set independently to zero mean and unit variance. We apply the standardization coefficients computed on the training set to the rest of the data (validation and testing set). The reported results are based on the standardized data (Chen et al., 2023). See Table 2 for the dataset sizes and consult the source code of the original papers (Python) or ours (C++) for more details.

We noticed order-of-magnitude outliers in the weather dataset training data. It would be misleading to fit those using the least squares method (Section C). Therefore, we preventively clip the normalized training data and the network input to the interval $[-10; 10]$ for all the datasets. The validation and testing data are left untouched.

B FULL RESULTS ON REAL-WORLD DATASETS

B.1 DETAILED VIEW

Table 3: The full results of LCESN approach on real-world datasets compared to the results of other authors. The best result is **red** and the second best result is blue. The reservoir size for ESN and LCESNs is 40×50 with a 7×7 kernel. The four LCESN variants and ESN baseline are not compared to each other. Datasets are sorted from longest (top) to shortest (bottom).

Models		ESN (2001)		**LCESN Ours**		**LCESN-LMS Ours**		**LCESN-LR100 Ours**		**LCESN-LR1 Ours**		TSMixer (2023)		PatchTST (2023)		DLinear (2023)		iTransformer (2024)		FEDformer (2022)		TimesNet (2023)		RLinear (2023)		Autoformer (2021)	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1	96	0.392	0.425	0.307	0.360	0.297	0.350	0.291	0.344	0.289	0.343	**0.285**	**0.339**	0.329	0.367	0.345	0.372	0.334	0.368	0.326	0.390	0.338	0.375	0.355	0.376	0.510	0.492
	192	0.475	0.483	0.363	0.397	0.341	0.377	0.335	0.372	0.332	0.371	**0.327**	**0.365**	0.367	0.385	0.380	0.389	0.377	0.391	0.365	0.415	0.374	0.387	0.391	0.392	0.514	0.495
	336	0.558	0.544	0.425	0.440	0.379	0.402	0.374	0.400	0.371	0.398	**0.356**	**0.382**	0.399	0.410	0.413	0.413	0.426	0.420	0.392	0.425	0.410	0.411	0.424	0.415	0.510	0.492
	720	0.734	0.656	0.554	0.524	0.442	0.442	0.449	0.453	0.443	0.450	**0.419**	**0.414**	0.454	0.439	0.474	0.453	0.491	0.459	0.446	0.458	0.478	0.450	0.487	0.450	0.527	0.493
	Avg	0.540	0.527	0.412	0.430	0.364	0.393	0.362	0.392	0.359	0.390	**0.347**	**0.375**	0.387	0.400	0.403	0.407	0.407	0.410	0.382	0.422	0.400	0.406	0.414	0.408	0.515	0.493
ETTm2	96	0.196	0.301	0.170	0.269	0.166	0.259	**0.162**	0.256	**0.162**	0.256	0.163	**0.252**	0.175	0.259	0.193	0.292	0.180	0.264	0.180	0.271	0.187	0.267	0.182	0.265	0.205	0.293
	192	0.292	0.375	0.231	0.318	0.223	0.301	0.218	0.299	0.218	0.299	**0.216**	**0.290**	0.241	0.302	0.284	0.362	0.250	0.309	0.252	0.318	0.249	0.309	0.246	0.304	0.278	0.336
	336	0.402	0.449	0.295	0.368	0.283	0.342	0.272	0.339	0.271	0.339	**0.268**	**0.324**	0.305	0.343	0.369	0.427	0.311	0.348	0.324	0.364	0.321	0.351	0.307	0.342	0.343	0.379
	720	0.630	0.573	0.436	0.462	0.427	0.427	**0.374**	0.410	**0.371**	0.409	0.420	0.422	0.402	0.400	0.554	0.522	0.412	0.407	0.410	0.420	0.408	0.403	0.407	**0.398**	0.414	0.419
	Avg	0.380	0.424	0.283	0.354	0.275	0.332	**0.257**	0.326	**0.255**	0.326	0.267	**0.322**	0.281	0.326	0.350	0.401	0.288	0.332	0.292	0.343	0.291	0.332	0.286	0.327	0.310	0.357
Weather	96	0.172	0.229	0.160	0.213	0.154	0.203	0.149	0.200	0.149	0.200	**0.145**	**0.198**	0.177	0.218	0.196	0.255	0.174	0.214	0.238	0.314	0.172	0.220	0.192	0.232	0.249	0.329
	192	0.224	0.277	0.205	0.263	0.202	0.252	**0.191**	0.245	0.192	0.246	**0.191**	**0.242**	0.225	0.259	0.237	0.296	0.221	0.254	0.275	0.329	0.219	0.261	0.240	0.271	0.325	0.370
	336	0.279	0.318	0.256	0.305	0.262	0.298	0.243	0.287	0.243	0.288	**0.242**	**0.280**	0.278	0.297	0.283	0.335	0.278	0.296	0.339	0.377	0.280	0.306	0.292	0.307	0.351	0.391
	720	0.346	0.369	**0.307**	**0.335**	0.325	**0.333**	**0.302**	**0.323**	**0.299**	**0.322**	0.320	0.336	0.354	0.348	0.345	0.381	0.358	0.347	0.389	0.409	0.365	0.359	0.364	0.353	0.415	0.426
	Avg	0.255	0.298	0.232	0.279	0.236	0.272	**0.221**	**0.264**	**0.221**	**0.264**	0.224	**0.264**	0.258	0.280	0.265	0.317	0.258	0.278	0.310	0.357	0.259	0.286	0.272	0.291	0.335	0.379
Solar	96	0.236	0.355	**0.171**	0.252	**0.172**	0.249	**0.170**	0.249	**0.170**	0.250	N/A	N/A	0.234	0.286	0.290	0.378	0.203	**0.237**	0.242	0.342	0.250	0.292	0.322	0.339	0.884	0.711
	192	0.254	0.369	**0.195**	0.267	**0.197**	0.262	**0.195**	0.263	**0.195**	0.264	N/A	N/A	0.267	0.310	0.320	0.398	0.233	**0.261**	0.285	0.380	0.296	0.318	0.359	0.356	0.834	0.692
	336	0.279	0.392	**0.214**	0.283	**0.215**	0.276	**0.214**	0.280	**0.213**	0.281	N/A	N/A	0.290	0.315	0.353	0.415	0.248	**0.273**	0.282	0.376	0.319	0.330	0.397	0.369	0.941	0.723
	720	0.304	0.422	**0.228**	0.306	**0.225**	0.286	**0.226**	0.300	**0.225**	0.301	N/A	N/A	0.289	0.317	0.356	0.413	0.249	**0.275**	0.357	0.427	0.338	0.337	0.397	0.356	0.882	0.717
	Avg	0.268	0.384	**0.202**	0.277	**0.202**	0.268	**0.201**	0.273	**0.201**	0.274	N/A	N/A	0.270	0.307	0.330	0.401	0.233	**0.262**	0.292	0.381	0.301	0.319	0.369	0.355	0.885	0.711
Electricity	96	0.247	0.337	0.219	0.325	0.189	0.301	0.166	0.278	0.164	0.276	**0.131**	**0.229**	0.181	0.270	0.197	0.282	0.148	0.240	0.186	0.302	0.168	0.272	0.201	0.281	0.196	0.313
	192	0.260	0.346	0.220	0.325	0.190	0.302	0.174	0.286	0.172	0.284	**0.151**	**0.246**	0.188	0.274	0.196	0.285	0.162	0.253	0.197	0.311	0.184	0.289	0.201	0.283	0.211	0.324
	336	0.268	0.353	0.221	0.326	0.194	0.307	0.184	0.297	0.183	0.295	**0.161**	**0.261**	0.204	0.293	0.209	0.301	0.178	0.269	0.213	0.328	0.198	0.300	0.215	0.298	0.214	0.327
	720	0.282	0.365	0.240	0.339	0.213	0.320	0.212	0.317	0.212	0.317	**0.197**	**0.293**	0.246	0.324	0.245	0.333	0.225	0.317	0.233	0.344	0.220	0.320	0.257	0.331	0.236	0.342
	Avg	0.264	0.351	0.225	0.329	0.197	0.308	0.184	0.294	0.183	0.293	**0.160**	**0.257**	0.205	0.290	0.212	0.300	0.178	0.270	0.207	0.321	0.192	0.295	0.218	0.298	0.214	0.326
Traffic	96	0.901	0.490	0.768	0.378	0.732	0.372	0.683	0.364	0.677	0.363	**0.376**	**0.264**	0.462	0.295	0.650	0.396	0.395	0.268	0.576	0.359	0.593	0.321	0.649	0.389	0.597	0.371
	192	1.024	0.512	0.826	0.389	0.789	0.384	0.733	0.375	0.729	0.374	**0.397**	0.277	0.466	0.296	0.598	0.370	0.417	**0.276**	0.610	0.380	0.617	0.336	0.601	0.366	0.607	0.382
	336	1.142	0.539	0.892	0.406	0.855	0.402	0.791	0.390	0.787	0.390	**0.413**	0.290	0.482	0.304	0.605	0.373	0.433	**0.283**	0.608	0.375	0.629	0.336	0.609	0.369	0.623	0.387
	720	1.314	0.592	1.041	0.454	1.006	0.450	0.942	0.439	0.938	0.439	**0.444**	0.306	0.514	0.322	0.645	0.394	0.467	**0.302**	0.621	0.375	0.640	0.350	0.647	0.387	0.639	0.395
	Avg	1.095	0.533	0.882	0.407	0.845	0.402	0.787	0.392	0.782	0.392	**0.408**	0.284	0.481	0.304	0.624	0.383	0.428	**0.282**	0.604	0.372	0.620	0.336	0.626	0.378	0.616	0.384
ETTh1	96	0.460	0.471	0.515	0.510	0.411	0.437	0.407	0.431	0.407	0.431	**0.361**	**0.392**	0.414	0.419	0.386	0.400	0.386	0.405	0.376	0.415	0.384	0.402	0.386	0.395	0.435	0.446
	192	0.581	0.557	0.708	0.613	0.482	0.485	0.481	0.482	0.483	0.483	**0.404**	**0.418**	0.460	0.445	0.437	0.432	0.441	0.436	0.423	0.446	0.436	0.429	0.437	0.424	0.456	0.457
	336	0.701	0.640	0.920	0.711	0.543	0.525	0.559	0.532	0.563	0.535	**0.420**	**0.431**	0.501	0.466	0.481	0.459	0.487	0.458	0.444	0.462	0.491	0.469	0.479	0.446	0.486	0.487
	720	0.857	0.743	1.129	0.813	0.627	0.591	0.700	0.628	0.710	0.633	**0.463**	0.472	0.500	0.488	0.519	0.516	0.503	0.491	0.469	0.492	0.521	0.500	0.481	**0.470**	0.515	0.517
	Avg	0.650	0.602	0.818	0.662	0.516	0.510	0.537	0.518	0.541	0.521	**0.412**	**0.428**	0.469	0.454	0.456	0.452	0.454	0.448	0.428	0.454	0.458	0.450	0.446	0.434	0.473	0.477
ETTh2	96	0.344	0.408	0.323	0.387	0.296	0.364	0.300	0.364	0.301	0.364	**0.274**	0.341	0.302	0.348	0.333	0.387	0.297	0.349	0.332	0.374	0.340	0.374	0.288	**0.338**	0.332	0.368
	192	0.486	0.492	0.404	0.440	0.364	0.409	0.374	0.411	0.377	0.412	**0.339**	**0.385**	0.388	0.400	0.477	0.476	0.380	0.400	0.407	0.446	0.402	0.414	0.374	0.390	0.426	0.434
	336	0.645	0.571	0.470	0.487	0.399	0.439	0.409	0.442	0.413	0.444	**0.361**	**0.406**	0.426	0.433	0.594	0.541	0.428	0.432	0.400	0.447	0.452	0.452	0.415	0.426	0.477	0.479
	720	0.761	0.643	0.554	0.544	0.427	0.465	0.431	0.467	0.435	0.468	0.445	0.470	0.431	0.446	0.831	0.657	0.427	0.445	**0.412**	0.469	0.462	0.468	0.420	**0.440**	0.453	0.490
	Avg	0.559	0.529	0.438	0.464	0.372	0.419	0.378	0.421	0.381	0.422	**0.355**	0.400	0.387	0.407	0.559	0.515	0.383	0.406	0.388	0.434	0.414	0.427	0.374	**0.398**	0.422	0.443
Exchange	96	0.417	0.413	0.636	0.535	0.417	0.447	0.335	0.399	0.308	0.382	N/A	N/A	0.088	**0.205**	0.088	0.218	**0.086**	0.206	0.148	0.278	0.107	0.234	0.093	0.217	0.197	0.323
	192	0.568	0.510	0.844	0.642	0.606	0.557	0.545	0.531	0.525	0.521	N/A	N/A	**0.176**	**0.299**	**0.176**	0.315	0.177	**0.299**	0.271	0.315	0.226	0.344	0.184	0.307	0.300	0.369
	336	0.825	0.644	1.176	0.778	0.945	0.714	0.842	0.678	0.820	0.670	N/A	N/A	**0.301**	**0.397**	0.313	0.427	0.331	0.417	0.460	0.427	0.367	0.448	0.351	0.432	0.509	0.524
	720	1.003	0.733	0.902	**0.688**	0.903	**0.690**	0.903	**0.690**	0.903	**0.690**	N/A	N/A	0.901	0.714	**0.839**	0.695	0.847	0.691	1.195	0.695	0.964	0.746	0.886	0.714	1.447	0.941
	Avg	0.703	0.575	0.889	0.661	0.718	0.602	0.656	0.574	0.639	0.566	N/A	N/A	0.366	0.404	**0.354**	0.414	0.360	**0.403**	0.518	0.429	0.416	0.443	0.378	0.418	0.613	0.539
# 1st		0/36	0/36	5/36	2/36	4/36	2/36	8/36	2/36	7/36	2/36	**24/28**	**20/28**	2/36	3/36	2/36	0/36	1/36	8/36	1/36	0/36	0/36	0/36	0/36	4/36	0/36	0/36

B.2 RELATIVE RANKING

Table 4: Relative ranking for models and datasets presented in Table 3. Value at i-th row and j-th column represents the number of datasets and horizons on which model i outperformed model j. Models are sorted from best (top) to worst (bottom).

	ESN	**LCESN**	**LCESN-LMS**	**LCESN-LR100**	**LCESN-LR1**	TSMixer	PatchTST	DLinear	iTransformer	FEDformer	TimesNet	Autoformer	RLinear	**SUM**
TSMixer	28	27	27	23	24	0	26	28	26	26	27	27	26	**315**
LCESN-LR1	36	34	26	19	0	4	24	24	19	23	25	26	21	**281**
LCESN-LR100	35	33	27	0	8	4	24	24	19	22	25	26	21	**268**
iTransformer	34	22	16	17	17	2	22	27	0	24	26	35	23	**265**
PatchTST	31	22	14	11	12	2	0	28	13	24	27	33	27	**244**
LCESN-LMS	33	30	0	8	8	1	22	24	19	22	22	25	21	**235**
FEDformer	27	20	14	14	13	2	12	23	11	0	19	34	22	**211**
TimesNet	29	21	14	11	11	1	9	25	10	17	0	29	19	**196**
RLinear	28	22	15	14	15	2	9	12	12	14	17	31	0	**191**
DLinear	28	17	12	12	12	0	6	0	8	13	11	24	22	**165**
LCESN	29	0	6	1	1	1	14	19	14	16	15	18	14	**148**
Autoformer	24	18	11	10	10	1	3	12	1	1	7	0	5	**103**
ESN	0	7	2	1	0	0	5	8	2	9	6	12	8	**60**

Table 5: The list of optimized hyperparameters with their genotype-to-phenotype transformation, initial values in the phenotype space, initial σ for CMA-ES in the genotype space, and their allowed range in genotype space. n_{in} denotes the average number of nonzero inputs to a neuron. σ^i_{in} and σ^i_{fb} denote the coefficients for i-th input and feedback, respectively.

Hyperparameter	Phenotype transform	Initial value	Initial σ	Boundaries
σ_{res}	$x \longmapsto e^{-50x}$	$1/\sqrt{2n_{in}}$	0.01	$[-0.1; 1.1]$
μ_{res}	$x \longmapsto 2x\lvert x\rvert$	0	0.05	$[-1.1; 1.1]$
σ^i_{in}	$x \longmapsto e^{-50x}$	1×10^{-5}	0.01	$[-0.1; 2.0]$
σ^i_{fb}	$x \longmapsto e^{-50x}$	1×10^{-5}	0.01	$[-0.1; 2.0]$
μ_b	$x \longmapsto 2x\lvert x\rvert$	0	0.05	$[-1.1; 1.1]$
λ_2	$x \longmapsto e^{-50x}$	1×10^{-8}	0.01	$[-0.1; 2.0]$

C TRAINING DETAILS

The network is driven by the training data $U \in \mathbb{R}^{T \times n_{in}}$ (fed to the network row by row) with the target data $Y \in \mathbb{R}^{T \times n_{out}}$, where T denotes the number of training time steps. Let us define matrix $A \in \mathbb{R}^{T \times n}$, whose i-th row is the vector of reservoir activations $a(i)$. Afterward, the linear least squares method with ridge regularization is used to find W_{out} as follows:

$$W_{out} = \arg\min_W ||AW - Y||_2^2 + \lambda_2 ||W||_2^2 \ ,$$

where $|| \cdot ||_2$ denotes the *Euclidean norm* and λ_2 denotes the strength of the regularization.

D EXPERIMENTAL SETTINGS

To minimize the *curse of dimensionality*, we optimize only a restricted set of parameters. The maximum memory length H is fixed at 100 because larger values require more GPU memory, and the model would also require a longer washout sequence. *Leakage* is fixed at 1, because the optimizer had a tendency to avoid leakage in the NARMA task, and the model already has the history of states available through the memory mechanism described in 3.4. *Sparsity* is fixed at 0, because a randomly pruned reservoir does not appear to have a significant impact even for fully connected networks (Matzner, 2022). Furthermore, our topology is sparse by design. σ_{noise} is set to zero because its purpose is a regularization that is already covered by the ridge regression. The learning rate for NLMS weight updates μ_{lms} is set arbitrarily at 1×10^{-3}.

We adopt the CMA-ES hyperparameter tuning framework by Matzner (2022), including genotype-to-phenotype transformations. Generally, the hyperparameters that are supposed to be positive (e.g., σ_{res}) are optimized in the logarithmic space via $x \longmapsto e^{-50x}$, and hyperparameters that need to take cautious steps around zero (e.g., μ_{res}) are optimized in square root space via $x \longmapsto 2x|x|$. Parameters denoting convex or affine combination coefficients are optimized directly. Table 5 summarizes the optimized hyperparameters, the corresponding transformations, and their upper and lower bounds.

In contrast to Matzner (2022), we do not create a new random network for every fitness evaluation. Instead, we reuse the same random seed, so the optimizer searches for the optimal scaling of a single network. This technique reduces noise and makes the optimization more stable.

The initial neuron activation is set to zero, and the first 500 steps of training data are used as a washout period to eliminate the effect of state initialization. Each task is optimized five times, each time with a different random seed (in a reproducible manner). For NARMA10 validation, each optimization run is followed by evaluating the best network on 100 new unique random NARMA10 sequences, and the best result is reported (or visualized). For the real-world datasets, we select the hyperparameters using the train and validation set, and the network with the best validation accuracy over the five optimization runs is used for testing on the provided test set. We use a reservoir of 40×50 with a 7×7 kernel.

The models for all the real-world datasets comply with the same optimization rules. In fact, due to the same random seed, they are the same reservoir, just scaled with different hyperparameters.

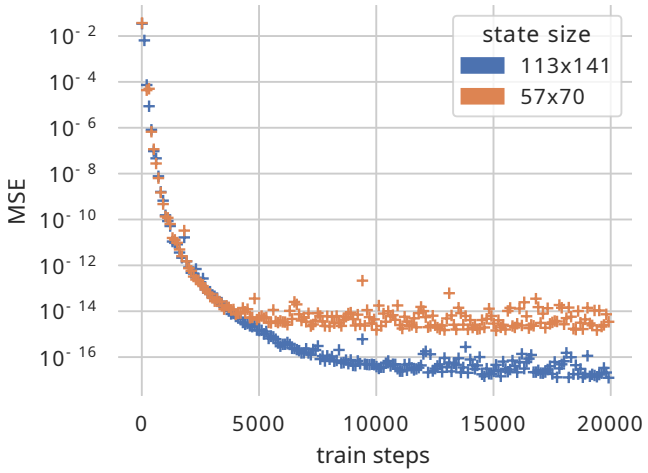

Figure 9: Comparison of various training set sizes of LCESN with 4000 and 16000 neurons and 7×7 kernel on NARMA10. Each point in the scatter plot represents a single evaluation of one of the best hyperparameters found during the NARMA10 experiments. The only parameter that is changing is the number of train steps.

By far, the most time-consuming part of the whole training procedure is the hyperparameter optimization. The hyperparameter tuning requires thousands of such evaluations, and thus, it is desired to minimize its computational time even at the cost of a rougher approximation. Therefore, we use the pure LCESN variant without linear regression recomputation and NLMS weight updates. We optimize our network for a prediction horizon of 192 steps and validate only every 30th subsequence.

For sequence prediction tasks, the most important part of the sequence tends to be just before the sequence to be predicted. To focus on this last part of the sequence, the training sample at time t is weighted by $e^{t/T}$, where T is the number of samples.

In sequence-to-sequence experiments (e.g., NARMA10), the feedback connections dramatically improve the results (Matzner, 2022) because the target sequence can be completely different from the input sequence. Without the feedback, the network would not be able to utilize its own output. On the other hand, sequence prediction tasks (e.g., our real-world datasets) already use the network output as the input for the next step, so using feedback connections would be redundant.

E Ablation Study

E.1 Effect of Training Size

According to the results presented in Section 6, larger networks seem to have a larger learning capacity. An important topic is whether the size of the training data was sufficient for the largest of the evaluated networks and whether their size also brings downsides to smaller training sets. According to Figure 9, which compares networks of 4,000 and 6,000 neurons on various training set sizes, larger network dominates the smaller network even with smaller training sets. As the training set size increases, the smaller network reaches its limit, and the larger one keeps improving.

At least on the NARMA10 task, choosing a larger network does not appear to have negative consequences, except for higher computational demands.

E.2 Effect of State Size

The aspect ratio of the state size does not appear to play a significant role in the performance of the network. Figure 10 demonstrates the effect of changing the aspect ratio of the state. Aspect ratio denotes the ratio of height to width, spanning from a square shape (aspect ratio 1) to a prolonged rectangle (aspect ratio 0). The state size 71×7 is significantly worse ($p < 0.05$) than all other evaluated state sizes. However, the other ratios performed similarly except for an outlier 26×19.

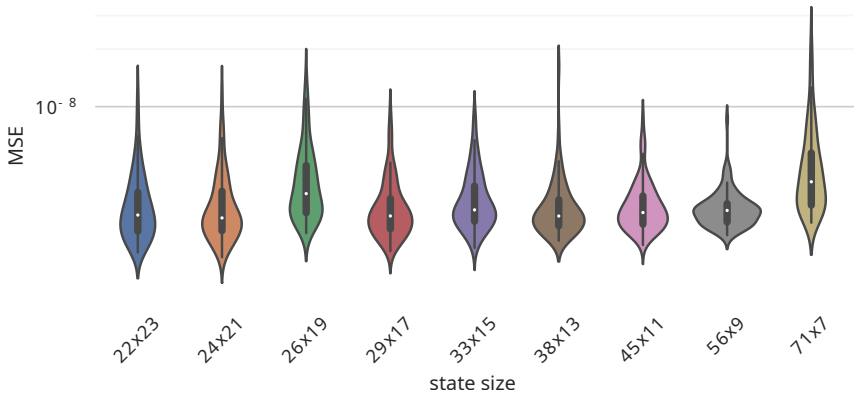

Figure 10: Comparison of various state aspect ratios of LCESN with 500 neurons and 7×7 kernel on NARMA10. Each violin represents the evaluation of 100 unique random sequences predicted by a network selected as the best of five optimization runs.

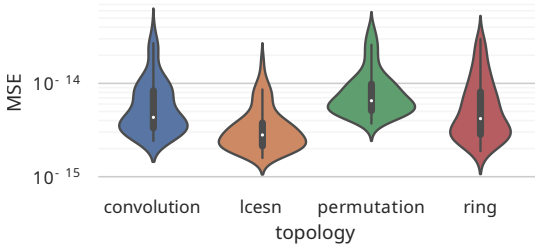

Figure 11: Comparison of various topologies of 4000 neurons on NARMA10. LCESN uses a 7×7 kernel. Each violin represents the evaluation of 100 unique random sequences predicted by a network selected as the best of five optimization runs.

According to the data, there is a chance that performance may be degraded in extreme cases where the kernel covers the entire height or width of the network. To mitigate the risk, we use state sizes closer to the 1:1 ratio.

E.3 Alternative topologies

Figure 11 compares multiple reservoir topologies. The ring topology was presented in Rodan and Tino (2011) as a minimum complexity ESN reservoir that does not affect performance. It simply connects the neurons to form a circle. The permutation topology (as evaluated, e.g., by Gallicchio and Micheli (2019) and Matzner (2022)) splits the neurons into multiple rings, each independent of the others. The convolution topology uses the same kernel technique as LCESN, but the kernels for all neurons are equal. In our experiments, LCESN significantly outperformed the three remaining topologies ($p < 0.05$).

F Details on GPU implementation

The topology of LCESN was designed with GPU memory properties in mind. LCESNs leverage that sparse reservoirs do not appear to harm accuracy (Matzner, 2022) while also providing opportunities for more efficient computation. Generic sparse matrix algorithms, however, inherently suffer from significant overhead due to irregular memory access, which negatively impacts GPU performance (Bell and Garland, 2009). To overcome these issues, we optimized our implementation to fully leverage the proposed local topology, minimizing memory overhead and ensuring coalesced GPU memory access (i.e., access in continuous blocks). The local connectivity is well-suited for GPU parallelization, allowing each neuron's state update to be handled independently by GPU threads, while still effectively utilizing shared GPU memory.

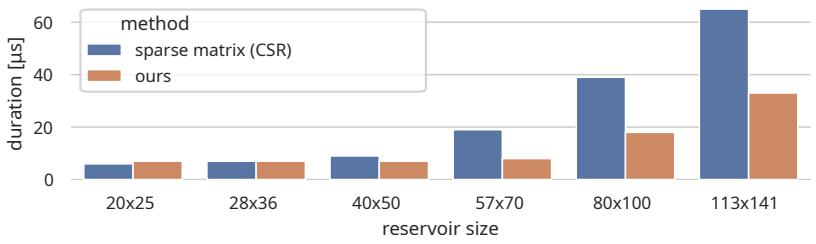

Figure 12: Average execution times of two different LCESN step implementations with a 7×7 kernel, based on 10 000 evaluations. Sparse matrix multiplication is performed using the Arrayfire CUDA library (Yalamanchili et al., 2022), utilizing the compressed sparse row (CSR) format. Experiments were conducted on an NVIDIA GTX 1080Ti GPU and an Intel i7-4770 CPU.

F.1 Performance

Our GPU implementation, tested on an NVIDIA GTX 1080 Ti, demonstrated a 14x speedup compared to a conventional fully connected ESN and 15x speedup compared a CPU implementation (Intel i7-4770), using a reservoir of 80x100 neurons and a 7x7 kernel.

The closest alternative to our custom implementation is sparse matrix multiplication. Figure 12 compares the execution times of the LCESN step implemented with our method versus the sparse matrix method. For very small reservoirs, the performance difference is minimal. However, for larger reservoirs of size 80×100 our implementation demonstrates an improvement of over 50%.

F.2 Algorithm overview

The reservoir weights are stored in a tensor of dimensions $N \times M \times K \times K$, where $N \times M$ represents the size of the reservoir, and $K \times K$ represents the kernel size. The element at index $[i, j, k, l]$ corresponds to the connection weight from the neuron at position $[i-K//2+k, j-K//2+l]$ to the neuron at position $[i, j]$ of the reservoir. In other words, each kernel slice $[i, j, :, :]$ stores the connections directed inward to the neuron at position $[i, j]$. The reservoir activations are stored as a tensor of dimensions $N \times M$. These tensors are stored in memory in the conventional column-wise order.

The order of dimensions is a crucial aspect of the implementation. Subsequent indices in the first dimension are contiguous in memory, while subsequent indices in other dimensions represent more distant memory addresses. Specifically for the reservoir weights matrix, each kernel is intentionally scattered across the memory.

The algorithm roughly consists of the following steps:

1. Allocate a perimeter of dimensions $(N + K) \times (M + K)$, which is the reservoir grid padded by state activations from the opposite side of the reservoir. This ensures that every neuron can access its entire kernel neighborhood locally, simulating a periodic (torus-shaped) reservoir. Allocate this perimeter to shared GPU memory.
2. Use the GPU thread index to access the reservoir grid in a top-down, left-right manner. In other words, allocate the GPU *warp* to subsequent neurons in the memory.
3. For each thread, traverse the weights of its kernel (top-down, left-right), multiply them by the corresponding presynaptic activation found in the perimeter, and accumulate the result in a local variable.
4. Write the accumulated result to the corresponding output in global memory.

By adhering to the specified dimension order and traversal sequence, GPU memory accesses are coalesced, ensuring optimal performance. For a more detailed description, please refer to the source code file `lcnn_step_cuda.cu`.

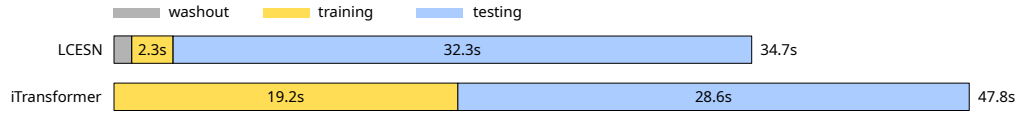

Figure 13: Decomposition of washout, training, and testing times on ETTh1 datasets for LCESN and iTransformer for prediction horizon of 96 steps.

Table 6: Comparison of inference speed in low-latency (1-step) and high-throughput (96-steps) scenarios for LCESN and iTransformer. In the 1-step scenario, only a single prediction is required as soon as possible. In the 96-steps scenario, LCESN has to be run subsequently 96 times in a self-feeding loop, while iTransformer predicts the entire horizon immediately. Both scenarios used the same LCESN model, while iTransformer was trained for each of the scenarios separately.

horizon	LCESN	iTransformer
1-step	**8361 steps/s**	107 steps/s
96-steps	87 steps/s	**97 steps/s**

G Detailed benchmarks

One of the main benefits of ESNs is their fast training procedure and inference. However, LCESN, as a recurrent model, differs from feedforward models in its usage (see Section 3.5). To clarify the most suitable tasks for ESNs, we present a detailed analysis of the benchmarking results from Section 6.2 and compare LCESN with alternative approaches.

The majority of time is spent on hyperparameter optimization using the CMA-ES procedure described in Section 3.2. We conducted 2,000 full evaluation cycles, which, in the case of the ETTh1 dataset, took less than four hours. The model selected for testing was the one with the best validation error across five independent hyperparameter runs, resulting in a total optimization time of 20 hours. While this approach provides a robust lower bound on achievable performance, it is also the weakest aspect of our method, offering significant opportunities for improvement. For example, early stopping or more efficient optimization algorithms could reduce this time. We adhered to the framework by Matzner (2022), leaving optimization improvements for future research.

It is rather difficult to compare running time of our model to others, because of hardware differences and often complicated reproducibility. Using information from the TSMixer paper (Chen et al., 2023), it is reported that the hyperparameter search grid required 1,600 training and validation cycles for each dataset (400 for each of four time horizons), which is similar to our model. Additionally, it is mentioned that a single training cycle of TSMixer took more than three hours on a high-end NVIDIA Tesla V100 GPU on a large M5 dataset. It suggests that the hyperparameter tuning was very time-consuming and may have required a cluster of GPUs. Although transformers may appear agnostic to hyperparameter perturbations, Liu et al. (2024) (Appendix C, Figure 9) noted considerable sensitivity in the tested TSF datasets. Unfortunately, the work does not detail their hyperparameter optimization procedure or its computational costs, making direct comparison challenging. To focus on LCESN's practical performance, we assume known hyperparameters in the subsequent experiments.

G.1 Training and Inference Time

Liu et al. (2024) (iTransformer) provided a quality code, and we were able to reproduce their results on the same hardware (NVIDIA GTX 1080 Ti GPU supported by a single core of an Intel i7-4770 CPU). Since iTransformer was the second strongest competitor, we will use it as a baseline measurement. Figure 13 illustrates the decomposition of training and testing times for LCESN and iTransformer on the ETTh1 datasets with a prediction horizon of 96 steps. For LCESN, the linear regression training procedure required just over two seconds, highlighting one of the strongest upsides of ESNs. The testing procedure involved running a 96-step self-feeding loop for each testing data point, which is inherently more time-consuming than training since the training phase traverses the data only once.

G.2 LATENCY VS. THROUGHPUT

Figure 13 does not show a complete picture. Feedforward models like iTranformer predict the entire prediction horizon at once and support parallel batch processing, while LCESN only predicts the next step. These differences offer distinct advantages. A fast 1-step prediction excels in low-latency scenarios, where only the immediate next prediction matters and the others can wait (e.g., control systems, trading, online signal filtering). In contrast, predicting the entire horizon and supporting parallel batches provides higher throughput (e.g., processing offline datasets or predicting many independent time series in parallel). Consult Table 6 for a comparison of latency and throughput of LCESN and iTransformer. Additionally, a 1-step prediction model like LCESN is flexible and can handle arbitrary horizons without retraining. Feedforward models like iTransformer, which predict fixed horizons, are traditionally retrained for each desired horizon separately.