Comparison of Gated Recurrent Neural Networks on Timeseries classification

Daniel Pistek

June 13, 2026

1 Introduction

In this project, we will compare how gated neural networks train on Timeseries classification data. The time series data tested will be diverse in origin and of varying sizes. This will achieve the goal that we will get good test results as regards how the models generalize on classification problem. We tested traditional gated neural networks such as LSTM[7] and GRU[3]. We will compare them with modern alternatives such as minLSTM[6], minGRU[6] and xLSTM[2].

2 Architecture of Neural Networks

In this section, we will explore the fundamental concepts of neural networks, focusing on their architecture and the reasoning that motivates their design.

2.1 LSTM

Originally introduced in [7], LSTM networks were designed to address a key limitation of recurrent Elman neural networks [5], namely the problems of exploding and vanishing gradients that arise when using Backpropagation Through Time (BPTT). LSTMs are particularly effective at mitigating the vanishing gradient issue, while exploding gradients are typically handled by gradient clipping. The LSTM architecture incorporates two types of memory: the cell state (long-term memory) and the hidden state (short-term memory). In addition, it uses three gates, the forget gate, the input gate, and the output gate. The forget gate $\mathbf{f}_t$ is used to decide how much to forget from the cell state $\mathbf{c}_t$, the input gate how much of a candidate state $\tilde{\mathbf{c}}_t$ to put into the cell state, and the output gate to decide how much of the output candidate $\tanh(\mathbf{c}_t)$ to use an output and the next hidden state.

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{H}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{1}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{H}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{2}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{H}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \tag{3}$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{H}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{4}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{5}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{6}$$

2.2 GRU

Introduced in [3], the main idea behind this neural network is to simplify LSTM. So instead of three gates, we have two gates, ant those are the reset gate $\mathbf{r}_t$ and the update gate $\mathbf{z}_t$. Also, we will only have the hidden state $\mathbf{h}_t$ in GRU. The reset gate controls how much of the previous hidden state we take into account when we calculate the candidate state $\tilde{\mathbf{h}}_t$, and the update

gate controls how much of the previous state and also a candidate state we use to calculate the new hidden state.

$$\mathbf{z}_t = \sigma\left(\mathbf{W}_z\mathbf{x}_t + \mathbf{H}_z\mathbf{h}_{t-1} + \mathbf{b}_z\right) \tag{7}$$

$$\mathbf{r}_t = \sigma\left(\mathbf{W}_r\mathbf{x}_t + \mathbf{H}_r\mathbf{h}_{t-1} + \mathbf{b}_r\right) \tag{8}$$

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}_h\mathbf{x}_t + \mathbf{H}_h\big(\mathbf{r}_t \odot \mathbf{h}_{t-1}\big) + \mathbf{b}_h\right) \tag{9}$$

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{10}$$

2.3 Min models

Introduced in [6]. They try to fix a problem of LSTM and GRU and that is the parallelization problem. Original models suffer from dependency on the previous state $\mathbf{h}_{t-1}$ in the gates, so we cannot calculate all time steps at once. The main idea of min models is to get rid of previous state dependency so they can use parallel scan algorithm to parallelize the computation. The parallel scan [6] is used to calculate a recurrence relation $v_t = a_t v_{t-1} + b_t$ in parallel.

2.3.1 minLSTM

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{h}}_t \tag{11}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{b}_f) \tag{12}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{b}_i) \tag{13}$$

$$\tilde{\mathbf{h}}_t = \mathbf{W}_h\mathbf{x}_t + \mathbf{b}_h \tag{14}$$

We can see that the dependency on the previous state in gates is gone and $\mathbf{h}_t$ is the only equation with recurrence, so the parallel scan can be applied here. The output gate was also removed.

2.3.2 minGRU

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{15}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{b}_z) \tag{16}$$

$$\tilde{\mathbf{h}}_t = \mathbf{W}_h\mathbf{x}_t + \mathbf{b}_h \tag{17}$$

Even more simplified model with only one gate. The reset gate was removed, and here we also have only one recurrent relation on which we can apply a parallel scan.

2.4 xLSTM

The xLSTM introduced in [2] is an improvement in LSTM architecture to be competitive with Transformers[10]. The xLSTM is made up of two cells, namely sLSTM and mLSTM. The sLSTM is a modified LSTM where the main difference is the introduction of exponential gating and normalizer state. The mLSTM is the highly parallelizable cell of the xLSTM because there is no memory mixing and is dependent only on the previous state matrix.

2.4.1 sLSTM [2]

$$\boldsymbol{c}_t = \mathbf{f}_t \odot \boldsymbol{c}_{t-1} + \mathbf{i}_t \odot \boldsymbol{z}_t \quad \text{cell state} \quad (18)$$

$$\boldsymbol{n}_t = \mathbf{f}_t \odot \boldsymbol{n}_{t-1} + \mathbf{i}_t \quad \text{normalizer state} \quad (19)$$

$$\boldsymbol{h}_t = \mathbf{o}_t \odot \tilde{\boldsymbol{h}}_t, \quad \tilde{\boldsymbol{h}}_t = \boldsymbol{c}_t \odot \boldsymbol{n}_t^{-1} \quad \text{hidden state} \quad (20)$$

$$\boldsymbol{z}_t = \varphi(\tilde{\boldsymbol{z}}_t), \quad \tilde{\boldsymbol{z}}_t = \boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{R}_z \boldsymbol{h}_{t-1} + \boldsymbol{b}_z \quad \text{cell input} \quad (21)$$

$$\mathbf{i}_t = \exp\left(\tilde{\mathbf{i}}_t\right), \quad \tilde{\mathbf{i}}_t = \boldsymbol{W}_{\mathbf{i}} \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{i}} \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{i}} \quad \text{input gate} \quad (22)$$

$$\mathbf{f}_t = \exp\left(\tilde{\mathbf{f}}_t\right) \text{ OR } \sigma\left(\tilde{\mathbf{f}}_t\right), \quad \tilde{\mathbf{f}}_t = \boldsymbol{W}_{\mathbf{f}} \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{f}} \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{f}} \quad \text{forget gate} \quad (23)$$

$$\mathbf{o}_t = \sigma(\tilde{\mathbf{o}}_t), \quad \tilde{\mathbf{o}}_t = \boldsymbol{W}_{\mathbf{o}} \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{o}} \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{o}} \quad \text{output gate} \quad (24)$$

2.4.2 mLSTM [2]

$$\mathbf{C}_t = \mathrm{f}_t \mathbf{C}_{t-1} + \mathrm{i}_t \boldsymbol{v}_t \boldsymbol{k}_t^\top \quad \text{cell state} \quad (25)$$

$$\boldsymbol{n}_t = \mathrm{f}_t \boldsymbol{n}_{t-1} + \mathrm{i}_t \boldsymbol{k}_t \quad \text{normalizer state} \quad (26)$$

$$\boldsymbol{h}_t = \mathbf{o}_t \odot \tilde{\boldsymbol{h}}_t, \quad \tilde{\boldsymbol{h}}_t = \mathbf{C}_t \boldsymbol{q}_t / \max\left\{\left|\boldsymbol{n}_t^\top \boldsymbol{q}_t\right|, 1\right\} \quad \text{hidden state} \quad (27)$$

$$\boldsymbol{q}_t = \boldsymbol{W}_q \boldsymbol{x}_t + \boldsymbol{b}_q \quad \text{query input} \quad (28)$$

$$\boldsymbol{k}_t = \frac{1}{\sqrt{d}} \boldsymbol{W}_k \boldsymbol{x}_t + \boldsymbol{b}_k \quad \text{key input} \quad (29)$$

$$\boldsymbol{v}_t = \boldsymbol{W}_v \boldsymbol{x}_t + \boldsymbol{b}_v \quad \text{value input} \quad (30)$$

$$\mathrm{i}_t = \exp(\tilde{\mathrm{i}}_t), \quad \tilde{\mathrm{i}}_t = \boldsymbol{w}_{\mathrm{i}}^\top \boldsymbol{x}_t + b_{\mathrm{i}} \quad \text{input gate} \quad (31)$$

$$\mathrm{f}_t = \sigma(\tilde{\mathrm{f}}_t) \text{ OR } \exp(\tilde{\mathrm{f}}_t), \quad \tilde{\mathrm{f}}_t = \boldsymbol{w}_{\mathrm{f}}^\top \boldsymbol{x}_t + b_{\mathrm{f}} \quad \text{forget gate} \quad (32)$$

$$\mathbf{o}_t = \sigma(\tilde{\mathbf{o}}_t), \quad \tilde{\mathbf{o}}_t = \boldsymbol{W}_{\mathbf{o}} \boldsymbol{x}_t + \boldsymbol{b}_{\mathbf{o}} \quad \text{output gate} \quad (33)$$

3 Datasets

In this section, we will explore the tested datasets. We will look at the size of the dataset and what it represents. All datasets from [1] are scalar.

3.1 Sleep

Sleep[1] is a classification dataset of 5 classes representing the stages of sleep. It is recorded as an EEG signal of length 178. The split is 478785 train samples and 90315 test samples. So, it is one of the larger datasets tested.

3.2 Mallat

Mallat[1] is a classification data set in which the goal is to properly classify a signal signature. There are 8 such signatures, so that means 8 classes. The split here is a little bit different because the train samples is only 55 and the test samples is 2345. So, the number of test samples far exceeds the number of train samples. The timeseries length is 1024.

3.3 Adiac

Adiac[1] is a data set for the classification of 37 classes. It originates from a project on classifications of diatoms(unicellular algae). The data set is sinusoidal. The split is 390 train samples and 391 test samples. Timeseries length is 176.

3.4 S2Agri

S2Agri[4] is a satellite imagery dataset of crops where the classification task is to classify the correct crops according to the satellite image pixel. In this case the pixel is not a scalar but a vector of size 10 and there are 24 steps in the time series. It is a large data set with 59 000 000 samples together that can be split. For this project, we will use a subset of this dataset S2Agri-10pc-34. This subset is more balanced compared to the full data set. It has around 5 000 000 samples.

4 Testing methodology and Implementation

4.1 Testing methodology

We will test the model so that they have a similar number of parameters. Usually it will be around 200 000 or 1 000 000 parameters. For xLSTM we will use only mLSTM cells for the implementation reason. We were unable to compile the sLSTM implementation, and the efficiency of not compiled sLSTM is very poor. All models will be trained five times on every dataset, so we can get good average performance and also the stability of the training. For results, we will provide the loss and accuracy of the models. The models were trained mainly for 100 epochs in the datasets.

4.2 Implementation

The training code is implemented in Python 3.12 [9] and using Pytorch 2.11 CUDA 12.8 [8]. For LSTM and GRU pytorch implementation is used. For minLSTM and minGRU implementation from this link. For xLSTM implementation this link is used. The NVIDIA H200 GPU is used for training these models.

5 Results

5.1 Sleep dataset

All models have around 200 000 parameters except min models, where binary search was not able to find better result. They were trained for 100 epochs with AdamW optimizer with a learning rate of 0.001 . We can see from 1 that xLSTM has the best convergence. But also we can see from 2 that the xLSTM, LSTM and GRU overfit on this dataset when training for a 100 epochs. The xLSTM has the fastest convergence.

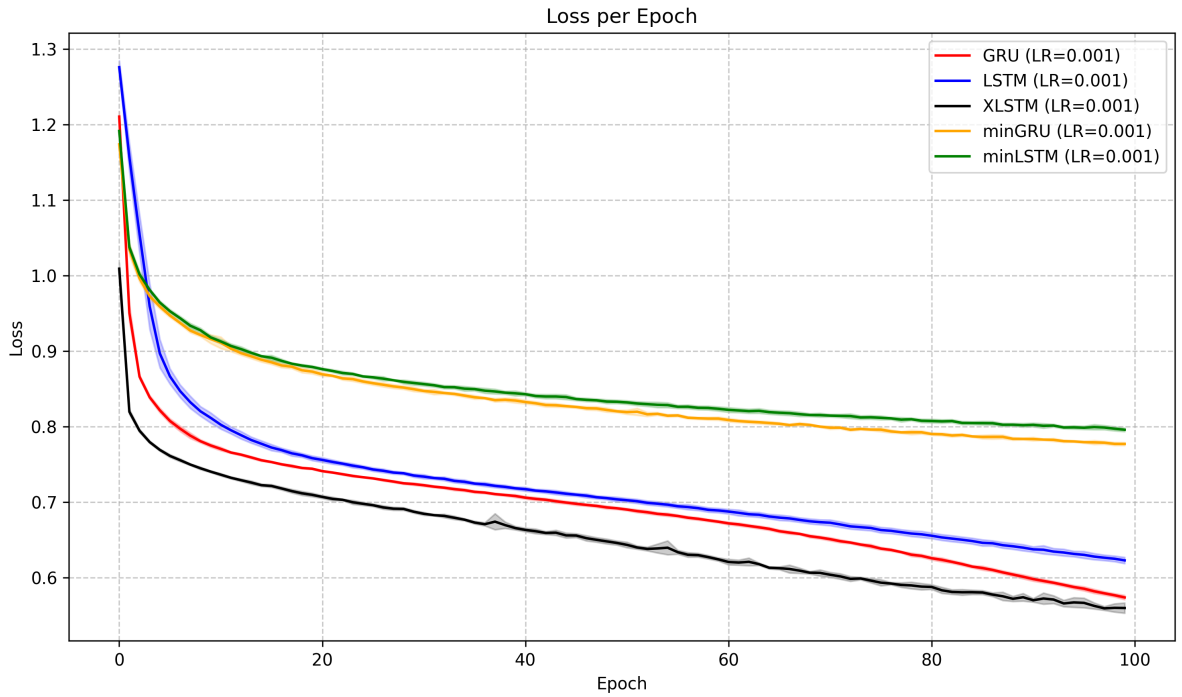

Figure 1: Loss Sleep

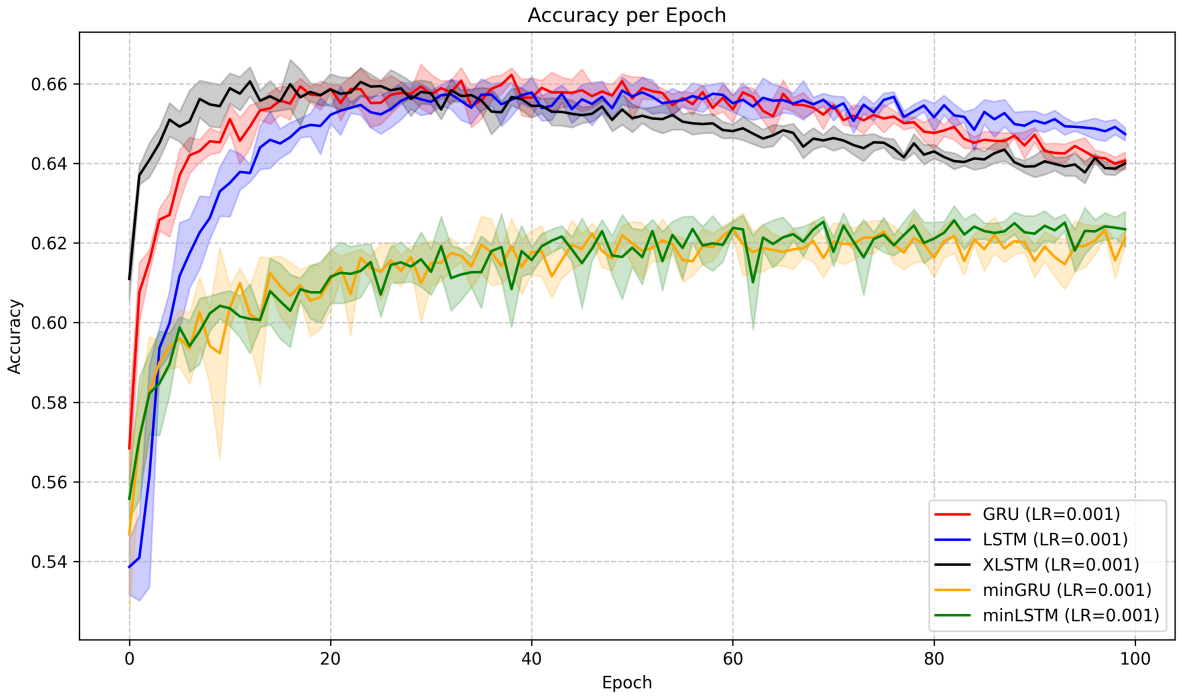

Figure 2: Accuracy Sleep

Model	Hidden Size	Parameters	FLOPs
LSTM	128	199,813	113,920
GRU	148	200,101	131,720
minLSTM	2	160,705	28,586,800
minGRU	3	180,705	32,146,800
xLSTM	64	213,765	71,038,248

Table 1: Comparison of sequential models based on Hidden Size, Parameters, and FLOPs.

5.2 Mallat dataset

All models have around 200 000 parameters except min models, where binary search was not able to find better result. They were trained for 100 epochs with AdamW optimizer with a learning rate of 0.001 . We can see from 3 that xLSTM has a much better convergence. This also reflects in 4 where it gets the best accuracy by a large margin. It is also interesting to see that on model with only 55 train samples the xLSTM gets such a good result. We can see that with lower learning rate the performance degrades.

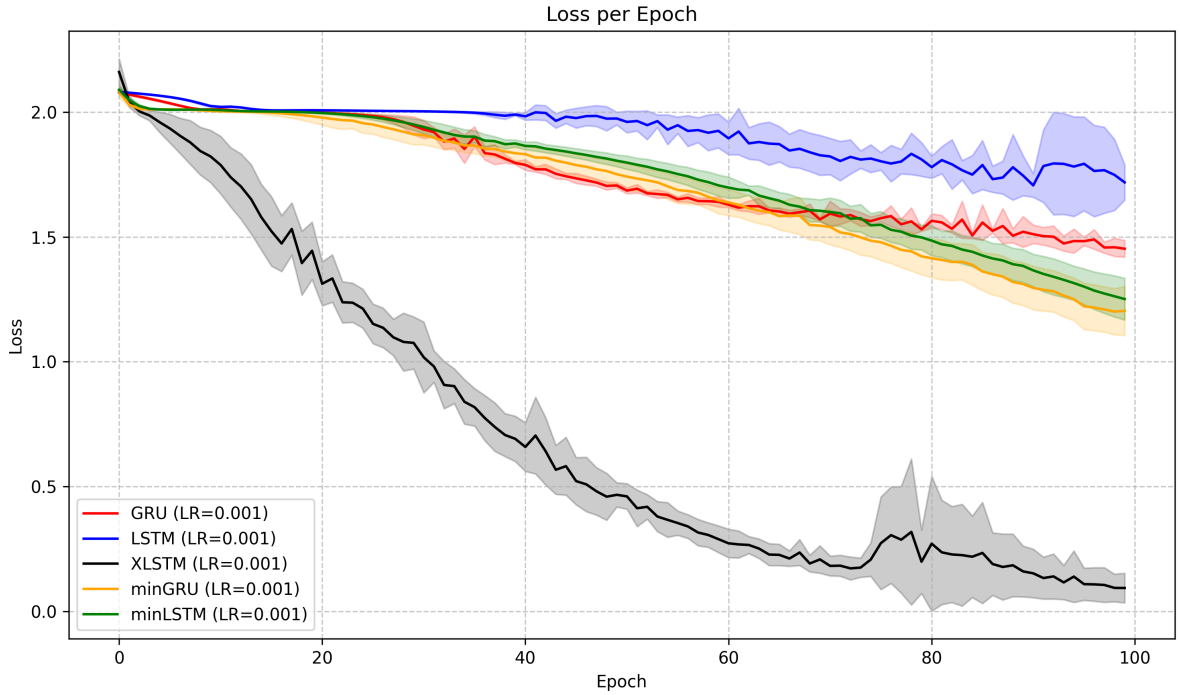

Figure 3: Loss Mallat 1e-3

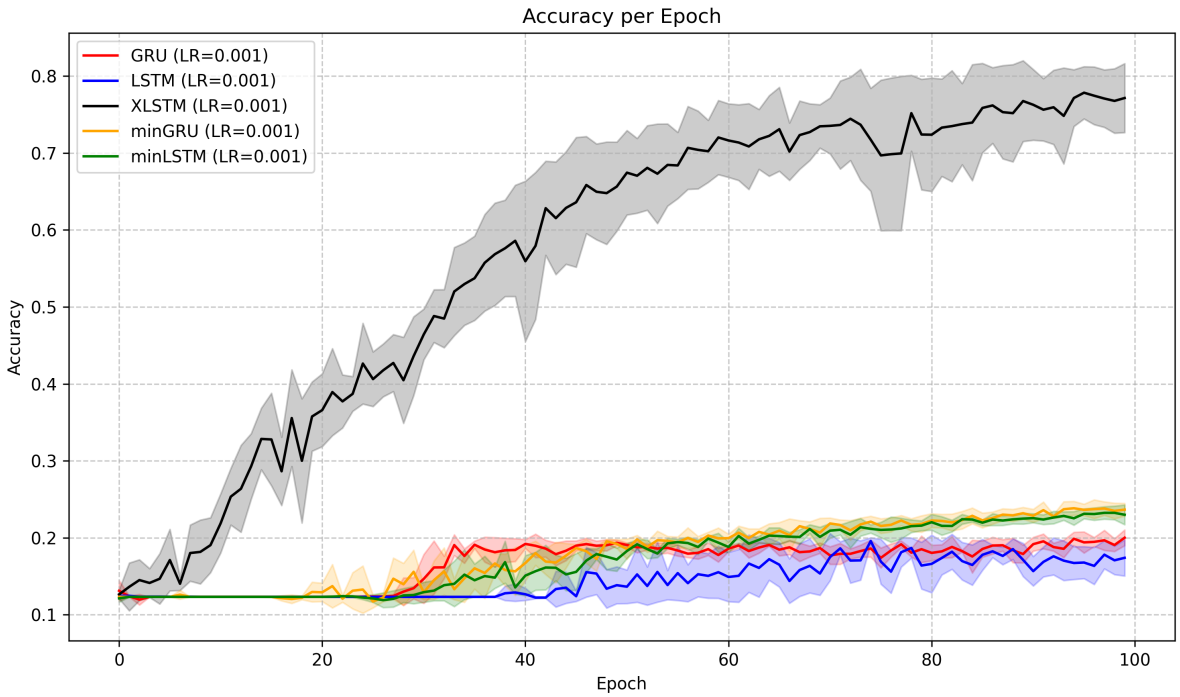

Figure 4: Accuracy Mallat 1e-3

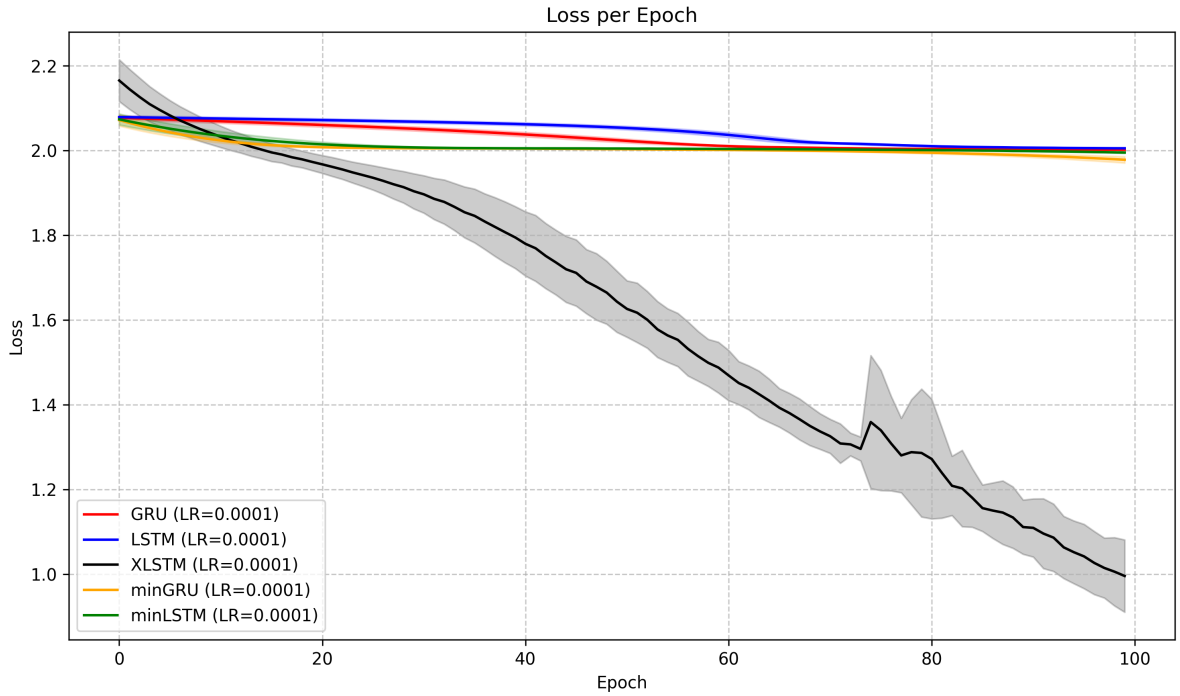

Figure 5: Loss Mallat 1e-4

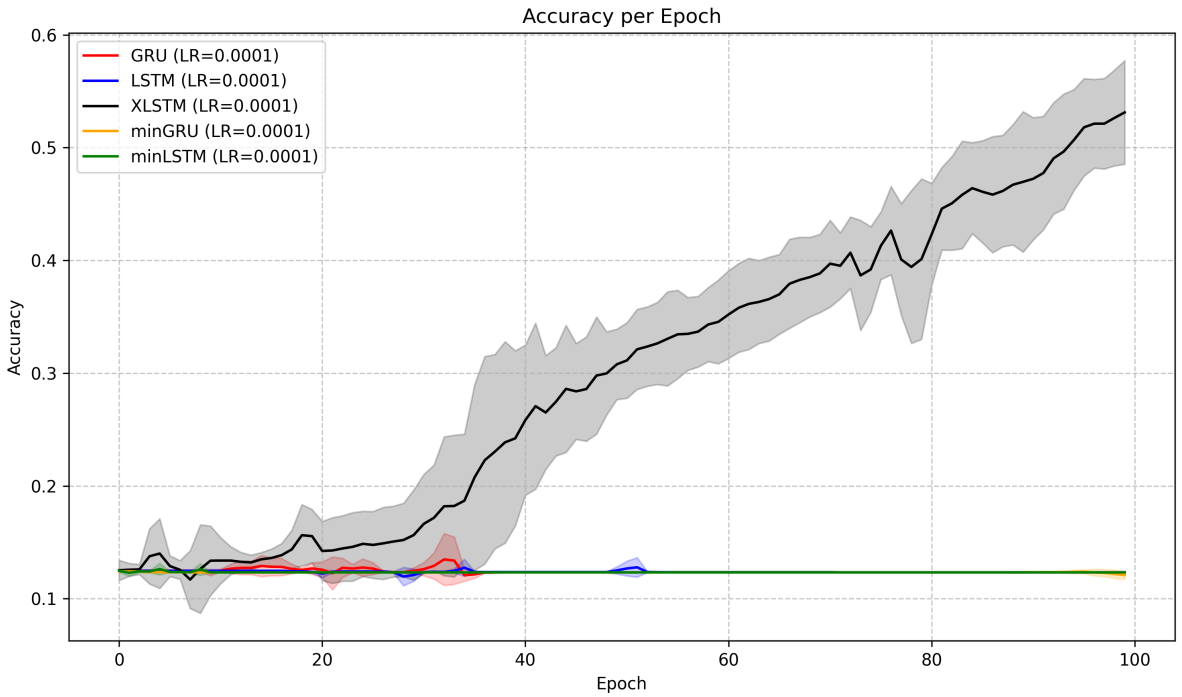

Figure 6: Accuracy Mallat 1e-4

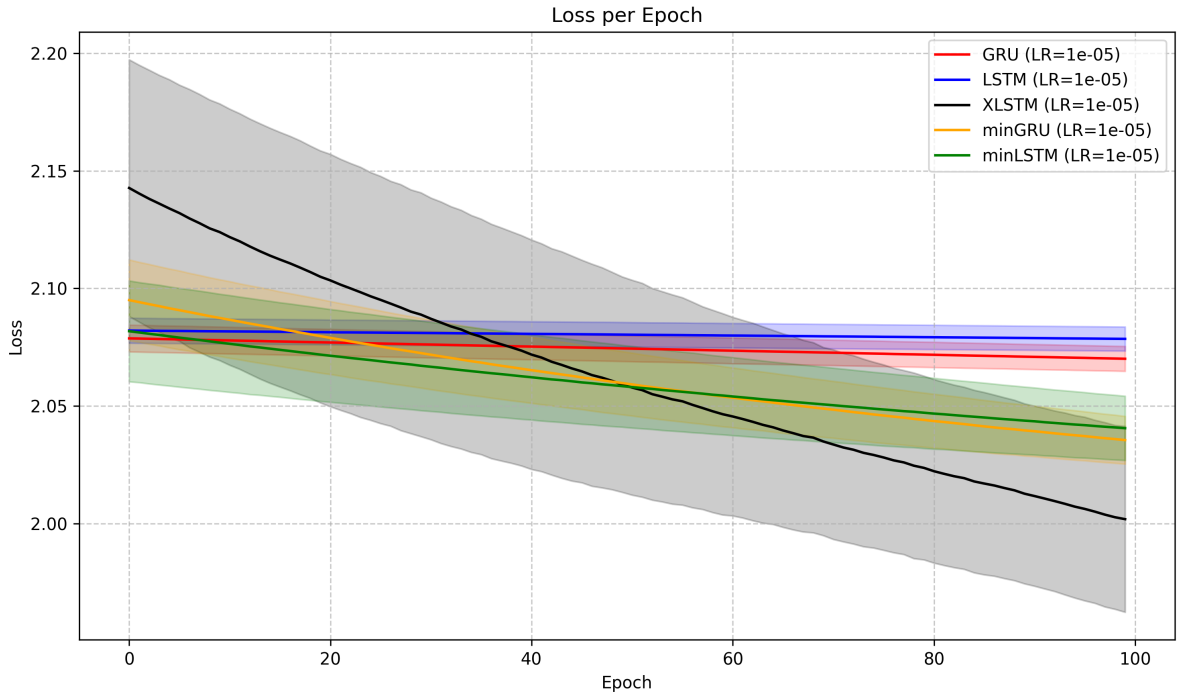

Figure 7: Loss Mallat 1e-5

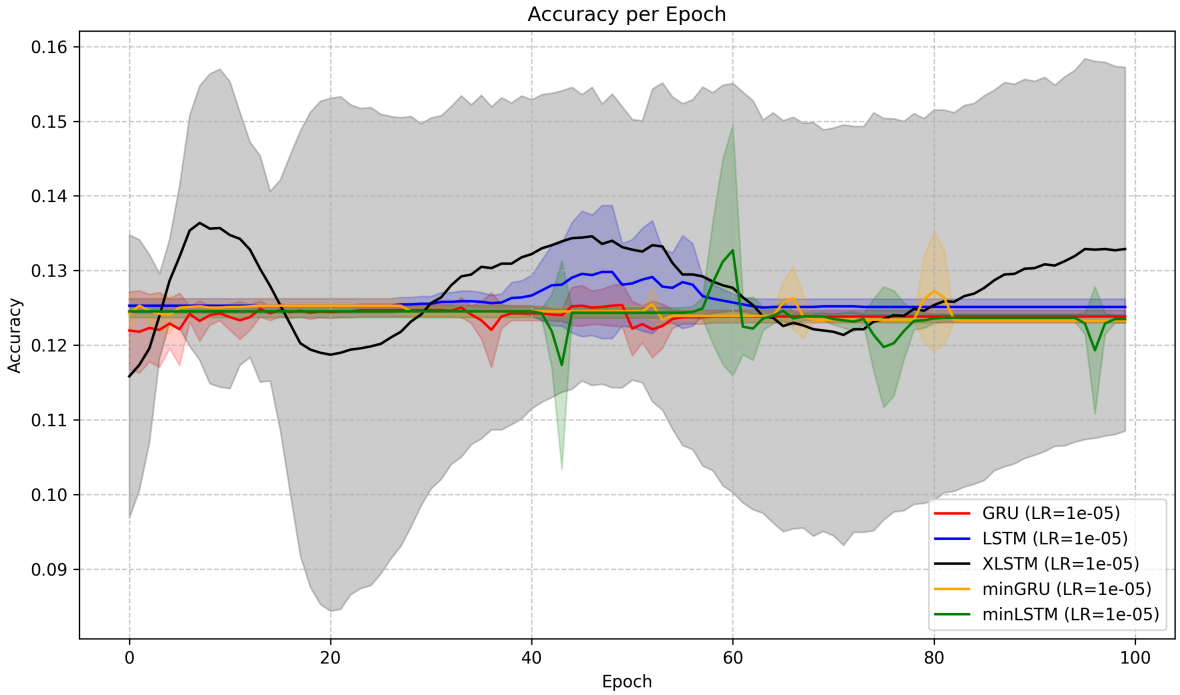

Figure 8: Accuracy Mallat 1e-5

Model	**Hidden Size**	**Parameters**	**FLOPs**
LSTM	128	200,200	1,048,576
GRU	148	200,548	1,212,416
minLSTM	2	161,008	164,761,600
minGRU	3	181,008	185,241,600
xLSTM	64	213,960	1,295,378,928

Table 2: Comparison of models based on Hidden Size, Parameters, and FLOPs.

5.3 Adiac dataset

All models have around 100 000 parameters. They were trained for 150 epochs with AdamW optimizer with a learning rate of 0.001 . We can see from 10 that the GRU encountered problems during training. Some training runs caused the loss to become NaN. Otherwise, all other models were quite simmilar at the end, but at earlier epochs the GRU was best. That means that with fewer epochs from data, it shows that for GRU the sweetspot for training is 40-60 epochs.

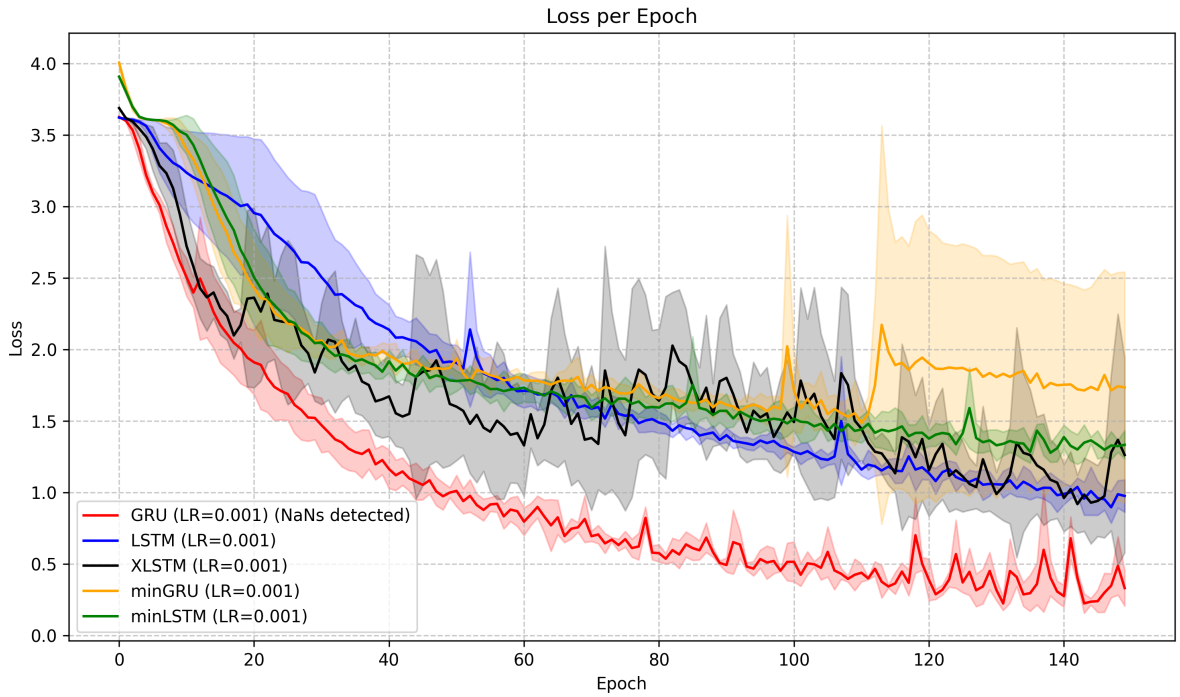

Figure 9: Loss Adiac

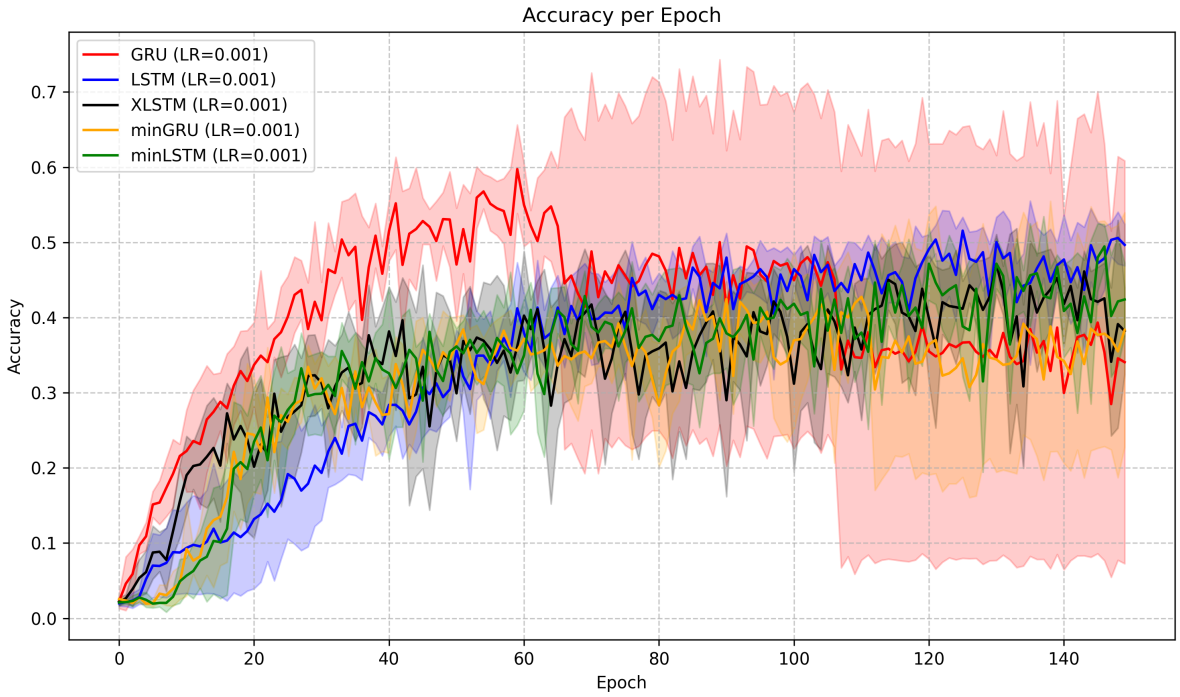

Figure 10: Accuracy Adiac

Model	Hidden Size	Parameters	FLOPs
LSTM	89	100,162	579,568
GRU	103	100,874	670,736
minLSTM	55	99,622	17,524,320
minGRU	74	100,522	17,682,720
xLSTM	40	85,445	47,107,824

Table 3: Comparison of models based on Hidden Size, Parameters, and FLOPs.

5.4 S2Agri-10pc-34 dataset

All models have around 200 000 parameters except min models, where binary search was not able to find better result. The batch size was twice as small. As we can see, only xLSTM , LSTM, and GRU were able to learn from the dataset. The Min models did not learn anything.

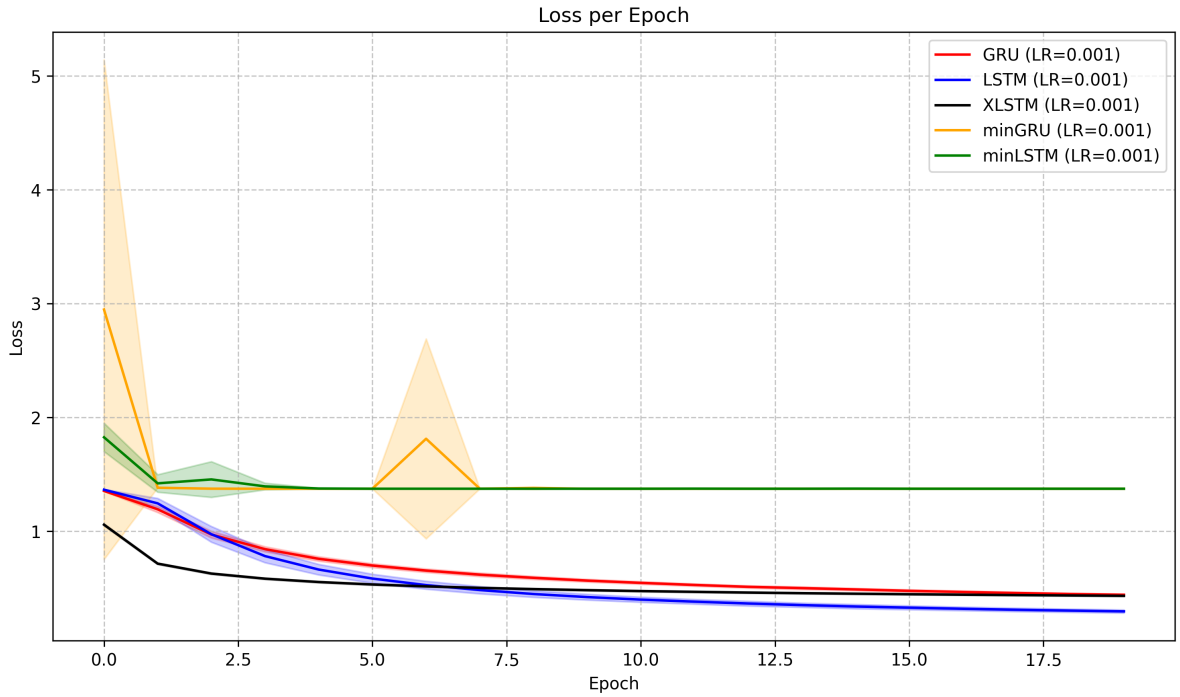

Figure 11: Loss S2Agri

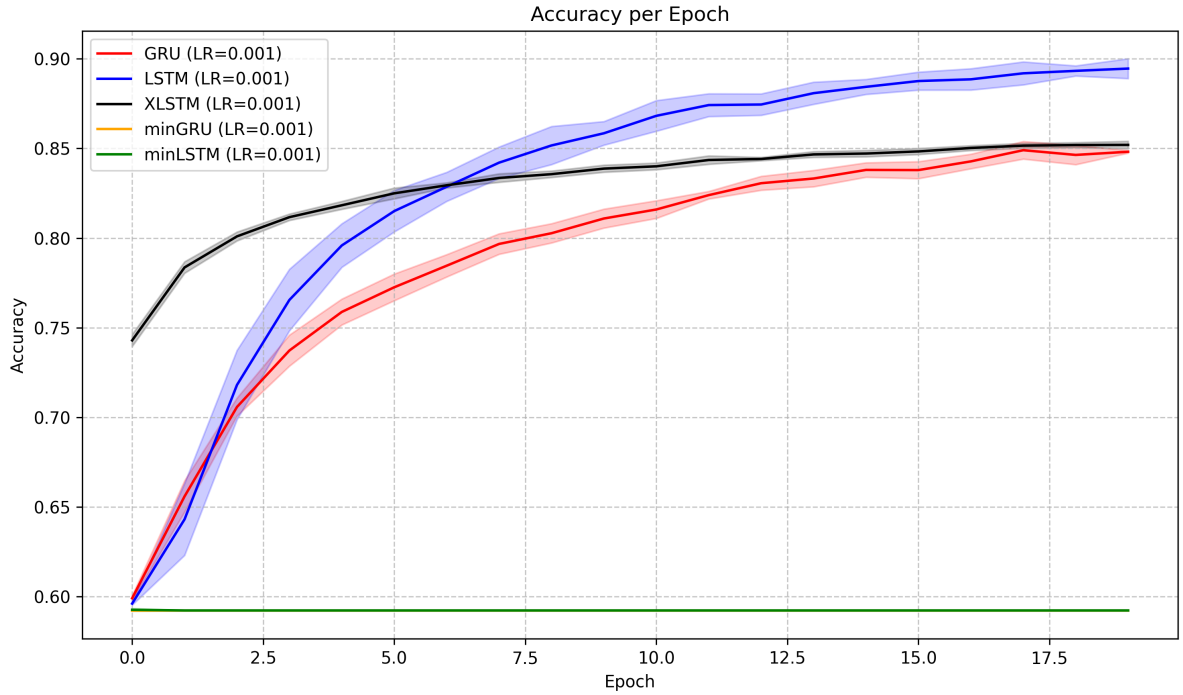

Figure 12: Accuracy S2Agri

Model	Hidden Size	Parameters	FLOPs
LSTM	125	198,784	102,000
GRU	145	200,279	118,320
minLSTM	2	164,534	3,945,600
minGRU	3	184,534	4,425,600
xLSTM	64	216,226	5,958,168

Table 4: Comparison of sequential models based on Hidden Size, Parameters, and FLOPs.

Note: Min models

Reason why the binary search was not able to find better result for parameters is because it searches based on expansion factor in min models. The base for 200 000 parameter model is 100 neurons and expanded from there. So the hidden size for min models is expansion factor. The plan is to improve this in later testing.

6 Conclusion

The best overall model is xLSTM. In most datasets, it either outperformed others or stays competitive even. The big disadvantage of the model is that it is expensive to run and train. LSTM and GRU performed well on most datasets. The min models performed the worst overall.

Some improvements for the future would be to change the accuracy metric. The problem with accuracy is that it does not say how well the model generalizes, but we can compare generalization between models that use it. The main alternative is AUC but it has problems. It can only be used for binary classification. There are some methods like OvO(One vs One) and

OvR(One vs Rest) that solve the problem by changing the n-class classification into multiple binary classifications that can be averaged. OvR is the simplest to run because I just need to do AUC so that I choose a class and compare it against the rest of the unified classes as one. This means that the number of AUC calculations grows linearly with the number of classes. For OvO we need to do AUC for every pair of classes, which means it grows quadratically with the number of classes. The last thing is to also improve the binary search algorithm for parameters so it is more robust.

Acknowledgments

This work was supported by the use of computational resources of the supercomputer PERUN, operated by the Supercomputing Centre at the Technical University of Košice (TUKE), Slovakia with the support of the European Union from the funds of the Recovery and Resilience Plan of the Slovak Republic within the framework of project No. 17I03-04-P03-00001, Development and design of a supercomputer for the National Supercomputing Center.

References

[1] Anthony Bagnall. Time series classification website. `https://www.timeseriesclassification.com/dataset.php`, 2026.

[2] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael K. Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. In *Advances in Neural Information Processing Systems*, 2024.

[3] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, 2014. Association for Computational Linguistics.

[4] Angus Dempster, Navid Mohammadi Foumani, Chang Wei Tan, Lynn Miller, Amish Mishra, Mahsa Salehi, Charlotte Pelletier, Daniel F. Schmidt, and Geoffrey I. Webb. MONSTER: Monash scalable time series evaluation repository. *Journal of Data-centric Machine Learning Research*, 2025. Dataset Certification.

[5] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[6] Leo Feng, Frederick Tung, Mohamed Osama Ahmed, Yoshua Bengio, and Hossein Hajimirsadeghi. Were rnns all we needed?, 2024. Unpublished.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[8] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[9] Guido van Rossum and Jelke de Boer. Interactively testing remote servers using the python programming language. *CWI Quarterly*, 4(4):283–303, 1991.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.