WERE RNNS ALL WE NEEDED?

Anonymous authors

000

001 002 003

004

005 006 007

008 009

010

011

012

013

014

015

016

017

018

019

021

023 024

025

Paper under double-blind review

Abstract

The introduction of Transformers in 2017 reshaped the landscape of deep learning. Originally proposed for sequence modelling, Transformers have since achieved widespread success across various domains. However, the scalability limitations of Transformers—particularly with respect to sequence length—have sparked renewed interest in novel recurrent models that are parallelizable during training, offer comparable performance, and scale more effectively. In this work, we revisit sequence modelling from a historical perspective, focusing on Recurrent Neural Networks (RNNs), which dominated the field for two decades before the rise of Transformers. Specifically, we examine LSTMs (1997) and GRUs (2014). We demonstrate that by simplifying these models, we can derive minimal versions (minLSTMs and minGRUs) that (1) use fewer parameters than their traditional counterparts, (2) are fully parallelizable during training, and (3) achieve surprisingly competitive performance on a range of tasks, rivalling recent models including Transformers.

1 INTRODUCTION

Since the 1990s, Recurrent Neural Networks (RNNs) (Elman, 1990), such as Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) networks and later Gated Recurrent Units (GRUs) (Cho et al., 2014), have been go-to methods for sequence modelling tasks like machine translation and text generation. However, their inherently sequential nature, which limits parallelization, made these models computationally inefficient and too slow to train on long sequences, a common challenge in real-world applications.

032 In 2017, Transformers (Vaswani et al., 2017) revolutionized deep learning by introducing a paral-033 lelizable training mechanism through self-attention, achieving immediate success in sequence mod-034 elling. This breakthrough led to the development of popular large language models and quickly extended to other domains, including computer vision (Dosovitskiy et al., 2021), reinforcement learning (Chen et al., 2021), and bioinformatics (Jumper et al., 2021). However, while self-attention 037 allows for efficient modelling of token-to-token interactions, it suffers from quadratic computa-038 tional complexity, making Transformers prohibitively expensive for long sequences, especially in resource-constrained settings. To address this, numerous approaches have focused on improving Transformer efficiency, exploring ideas such as sparsity (Kitaev et al., 2019), low-rank approxima-040 tions (Wang et al., 2020), and tiling (Dao et al., 2022). 041

- Recently, the scalability limitations of Transformers have sparked renewed interest in alternative approaches: novel recurrent models that are parallelizable and scale more efficiently. Several promising methods have emerged in this space, including state-space models (Gu et al., 2021), linearized attention (Peng et al., 2023), and more recently, linear recurrent neural networks (Orvieto et al., 2023). Notably, these state-of-the-art recurrent models leverage input-dependent transitions and demonstrate strong performance similar to Transformers. These methods have shown success not only in scaling to large language models but also in extending to other domains, such as image (Zhu et al., 2024a) and graph-based data (Wang et al., 2024a).
- In this work, we revisit sequence modelling from a historical perspective, focusing on the RNNs that dominated the field for two decades before the rise of Transformers. Specifically, we explore LSTMs (1997) and GRUs (2014), which are early examples of input-dependent recurrent models. We show that by removing the dependencies of their gates on previous states, we can train these models in parallel. Further simplification leads to minimal versions (minLSTMs and minGRUs)

054 that (1) use fewer parameters than their traditional counterparts, (2) are fully parallelizable during 055 training, and (3) achieve surprisingly competitive performance on a range of tasks despite their 056 simplicity, challenging the prevailing trend in the community toward increasing architectural and 057 algorithmic complexity. In the appendix, we provide implementations of minGRU and minLSTM in 058 plain PyTorch, with just a few lines of code, making these models lightweight and highly adaptable for beginners, practitioners, and researchers.

060 061

062

069

070

2 BACKGROUND

063 In this section, we review traditional recurrent neural networks (RNNs). RNNs are sequence models 064 that maintain a hidden state across time steps, capturing temporal dependencies. As such, they are 065 particularly well-suited for tasks involving sequential data, such as time series forecasting, natural 066 language processing, and other tasks where context from previous steps informs current predictions. However, vanilla RNNs (Elman, 1990) face challenges related to vanishing and exploding gradients, 067 which limit their ability to learn long-term dependencies. 068

2.1 LSTM

071 To address these issues, Hochreiter & Schmidhuber (1997) introduced Long Short-Term Memory 072 (LSTM) networks. LSTMs are a highly successful type of RNN specifically designed to mitigate 073 the vanishing gradient problem, enabling the model to effectively capture long-term dependencies. 074 LSTMs are computed as follows: 075

076	(Hidden State)	$oldsymbol{h}_t = oldsymbol{o}_t \odot anh(oldsymbol{c}_t)$
077	(Output Gate)	$\boldsymbol{o}_t = \sigma(\operatorname{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))$
078	(Cell State Recurrence)	$oldsymbol{c}_t = oldsymbol{f}_t \odot oldsymbol{c}_{t-1} + oldsymbol{i}_t \odot ilde{oldsymbol{c}}_t$
079	(Forget Gate)	$f_t = \sigma(\operatorname{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))$
080	(Input Gate)	$i_t = \sigma(\text{Linear}_d, ([x_t, h_{t-1}]))$
081	(Candidata Call Stata)	$\tilde{a} = \tanh(\text{Lincor}([\mathbf{a}, \mathbf{b}, 1]))$
082	(Canudate Cell State)	$\boldsymbol{c}_t = \operatorname{tann}(\operatorname{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{n}_{t-1}]))$

083

096

097

084 where \odot denotes element-wise multiplication of vectors, t is the current timestep, and h_t is the 085 outputted hidden state. $[x_t, h_{t-1}]$ represents the concatenation of the input vector x_t at time step t with the previous hidden state h_{t-1} . d_h denotes the size of the hidden state, while c_t is the cell state, 086 which carries information across time steps, and \tilde{c}_t is the candidate cell state that will be added to 087 the cell state. 088

The gates i_t , f_t , and o_t control the flow of information through the LSTM. The input gate i_t deter-090 mines how much new information from the candidate cell state \tilde{c}_t should be added to the cell state c_t . The forget gate f_t determines what portion of the previous cell state c_{t-1} should be discarded. 091 The output gate o_t determines what information from the cell state should be output as the hidden 092 state h_t . The functions σ (sigmoid) and tanh are used for scaling the values, ensuring that the out-093 puts do not explode or vanish during training. An LSTM module maintains both a cell state and a 094 hidden state, and, in total, contains $O(4d_h(d_x + d_h))$ parameters, where d_x is the input size. 095

2.2 GRU

098 Simplifying LSTM, Cho et al. (2014) introduced the Gated Recurrent Unit (GRU), which uses only 099 two gates and a single state (hidden state), in contrast to the LSTM's three gates and two states 100 (hidden state and cell state). This reduced complexity allows GRUs to achieve faster training and 101 inference times while still performing competitively on many tasks. GRUs are computed as follows:

102	(II: Idea State Decompose)	\mathbf{h} $(1$ $\mathbf{n}) \cap \mathbf{h}$ \mathbf{h} $\mathbf{n} \cap \tilde{\mathbf{h}}$
103	(Hidden State Recurrence)	$\boldsymbol{n}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{n}_{t-1} + \boldsymbol{z}_t \odot \boldsymbol{n}_t$
104	(Update Gate)	$oldsymbol{z}_t = \sigma(ext{Linear}_{d_h}([oldsymbol{x}_t,oldsymbol{h}_{t-1}]))$
105	(Reset Gate)	$oldsymbol{r}_t = \sigma(\mathrm{Linear}_{d_h}([oldsymbol{x}_t,oldsymbol{h}_{t-1}]))$
106	(Candidate Hidden State)	$\tilde{\boldsymbol{h}}_t = \tanh(\operatorname{Linear}_d ([\boldsymbol{x}_t, \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}]))$
107	(Canalance Indeen State)	$u_{t} = u_{t} (u_{t}, u_{t} \in u_{t}))$

where \tilde{h}_t represents the candidate hidden state, a potential new value for the hidden state. GRU combines the forget and input gates of LSTM into a single update gate, $z_t \in (0, 1)$, which determines how much of the past information should be carried forward (i.e., $1 - z_t$) and how much new information from the candidate hidden state should be added (i.e., z_t). Additionally, GRU removes LSTM's output gate and introduces a reset gate r_t , which controls how much of the past hidden state h_{t-1} is used when computing the candidate hidden state \tilde{h}_t .

By reducing the number of gates and states, GRU also decreases the total number of parameters and computations, requiring only $O(3d_h(d_x + d_h))$ parameters. However, both GRUs and LSTMs are still sequential-only models. As such, they require backpropagation through time (BPTT) during training, resulting in linear training time and limiting their ability to scale to long contexts.

118

120

119 2.3 PARALLEL SCAN

Due to this limitation, the introduction of Transformers in 2017 revolutionized the field by replacing
LSTMs and GRUs as the de facto method for sequence modelling. Transformers leverage parallelization during training, overcoming the sequential bottleneck of traditional recurrent models.
However, instead, Transformers have a quadratic complexity with respect to the sequence length,
limiting their ability to scale to very long contexts, especially in resource-constrained settings.

In response, a resurgence of new recurrent sequence models has emerged, offering alternatives to
 Transformers. These models achieve comparable performance while being trainable in parallel and
 avoid the backpropagation through time (BPTT) issues that plagued traditional RNNs (e.g., LSTMs
 and GRUs). Among these innovations, many architectures rely on the parallel prefix scan algorithm (Blelloch, 1990) for efficient training.

131 The parallel scan algorithm is a parallel computation method for computing N prefix computations 132 from N sequential data points via an associative operator \oplus (e.g., + and \times). The algorithm efficiently computes the sequence of prefix sums $\{\bigoplus_{i=1}^{k} u_i\}_{k=1}^{N}$ from the input sequence $\{u_k\}_{k=1}^{N}$. One 133 134 important application of the parallel scan algorithm is in computing a popular class of recurrence 135 relations of the form $v_t = a_t v_{t-1} + b_t$, where v_t, a_t , and b_t are real numbers and $v_0 \leftarrow b_0$ (Martin & Cundy, 2018). This method takes as input the sequences a_1, \ldots, a_n and b_0, b_1, \ldots, b_n , and computes 136 the sequence v_1, \ldots, v_n in parallel. This approach naturally extends to vector-valued recurrences, 137 such as $v_t = a_t \odot v_{t-1} + b_t$, where \odot denotes element-wise multiplication. 138

139

3 Methodology

140 141

Interestingly, we can see that the GRU's hidden state and LSTM's cell state recurrences resemble the
vector formulation. In this section, we demonstrate that GRUs and LSTMs are trainable via parallel
scan by removing their previous state dependencies from their various gates. Building on this, we
further simplify these RNNs by removing their constraints on output range (i.e., tanh). Combining
the steps, we describe minimal versions of GRUs and LSTMs (minGRUs and minLSTMs) that are
trainable in parallel.

147 148 149

153

3.1 A MINIMAL GRU: MINGRU

3.1.1 STEP 1: DROP PREVIOUS STATE DEPENDENCIES FROM GATES

152 Revisiting GRU's hidden state recurrence which works as follows:

$$oldsymbol{h}_t = (oldsymbol{1} - oldsymbol{z}_t) \odot oldsymbol{h}_{t-1} + oldsymbol{z}_t \odot oldsymbol{ heta}_t$$

We can observe that the recurrence resembles the aforementioned parallel scan's formulation $v_t = a_t \odot v_{t-1} + b_t$ where $a_t \leftarrow (1 - z_t)$, $b_t \leftarrow z_t \odot \tilde{h}_t$, and $v_t \leftarrow h_t$. However, z_t and \tilde{h}_t are dependent on the previous hidden state h_{t-1} , i.e., $z_t = \sigma(\text{Linear}_{d_h}([x_t, h_{t-1}]))$ and $\tilde{h}_t = \text{tanh}(\text{Linear}_{d_h}([x_t, r_t \odot h_{t-1}]))$. As a result, it is not possible to apply the parallel scan as is since the algorithm's inputs a_1, \ldots, a_n and b_1, \ldots, b_n are conditional on already knowing its outputs h_1, \ldots, h_{n-1} .

161 A simple remedy to this is to simplify GRU by removing their previous hidden state (i.e., h_{t-1}) dependencies. Specifically, the changes are as follows:

163 $z_t = \sigma(\text{Linear}_{d_h}([x_t, h_{t-1}]))$ 164 $z_t = \sigma(\text{Linear}_{d_h}([x_t, h_{t-1}]))$ \Rightarrow $z_t = \sigma(\text{Linear}_{d_h}(x_t))$

$$egin{aligned} &m{r}_t = \sigma(ext{Linear}_{d_h}([m{x}_t,m{h}_{t-1}])) &\Rightarrow \ &m{ ilde{h}}_t = ext{tanh}(ext{Linear}_{d_h}([m{x}_t,m{r}_t\odotm{h}_{t-1}])) &\Rightarrow \ &m{ ilde{h}}_t = ext{tanh}(ext{Linear}_{d_h}(m{x}_t,m{h}_t\odotm{h}_{t-1}])) \end{aligned}$$

165 166 167

181

191 192 193

194 195

196 197

162

By removing the dependence on h_{t-1} from the candidate hidden state \tilde{h}_t , the reset gate r_t that would control h_{t-1} weight is also no longer needed and is removed. Without the dependencies on previous hidden states, the inputs to the algorithm a_1, \ldots, a_n and b_1, \ldots, b_n are all easily computed in parallel and can thus be used to compute h_1, \ldots, h_n efficiently via the parallel scan.

 (\boldsymbol{x}_t)

Although there have been theoretical concerns about the absence of previous state dependen-172 cies (Merrill et al., 2024), there has also been substantial empirical evidence supporting the effec-173 tiveness of models that omit these dependencies, such as xLSTM (Beck et al., 2024) and Mamba (Gu 174 & Dao, 2024). Instead of explicitly modelling dependencies on previous states to capture long-range 175 dependencies, these kinds of recurrent models can learn them by stacking multiple layers. Notably, 176 in the xLSTM paper, their fully parallelized version (xLSTM[1:0]), which eliminates hidden state 177 dependencies, performed similarly to — and in some cases, better than — versions that retain these 178 dependencies (e.g., xLSTM[7:1]). 179

180 3.1.2 Step 2: Drop range restriction of candidate states

In GRU's hidden state recurrence, the proportion carried over from the previous hidden state $(1-z_t)$ 182 and the amount added for the new candidate hidden state (z_t) sum to 1. As a result, the scale of 183 GRU's hidden state value is time-independent. Instead, the scale of its hidden state depends on that of its candidate hidden states h_t . The hyperbolic tangent function (tanh) plays a crucial role in 185 LSTMs and GRUs, restricting the range of (candidate) hidden states, i.e., $\tilde{h}_t, h_t \in (-1, 1)^{d_h}$. The tanh helps stabilize the training and mitigates vanishing gradients that result from applying sigmoid 187 (σ) activations to linear transformations of the hidden state (e.g., $z_t = \sigma(\text{Linear}_{d_h}([x_t, h_{t-1}])))$). In 188 the previous step, these hidden state dependencies were removed. As such, we simplify GRU further 189 by removing the range restriction (tanh) on the (candidate) hidden states as follows: 190

$$\tilde{\boldsymbol{h}}_t = anh(ext{Linear}_{d_h}(\boldsymbol{x}_t)) \quad \Rightarrow \quad \tilde{\boldsymbol{h}}_t = ext{Linear}_{d_h}(\boldsymbol{x}_t)$$

3.1.3 MINGRU

Combining the two simplification steps results in a minimal version of GRU (minGRU):



The resulting model is significantly more efficient than the original GRU, requiring only $O(2d_hd_x)$ parameters, compared to GRU's $O(3d_h(d_x + d_h))$ parameters, where d_x and d_h denote the sizes of the input x_t and the hidden state h_t , respectively. In RNNs, state expansion is often used (i.e., $d_h = \alpha d_x$, where $\alpha \ge 1$), which helps the models better capture features from the input data. minGRU uses approximately 33%, 22%, 17%, and 13% of the parameters of a GRU when $\alpha = 1, 2, 3, 4$, respectively.

Additionally, the minimal version of GRU can now be trained in parallel using the parallel scan
 algorithm, bypassing the need for backpropagation through time (BPTT). Pseudocode and a simple
 PyTorch implementation are included in the Appendix.

3.2 A MINIMAL LSTM: MINLSTM 3.2.1 STEP 1: DROP PREVIOUS STATE DEPENDENCIES FROM GATES Revisiting LSTM's cell state recurrence which works as follows: $oldsymbol{c}_t = oldsymbol{f}_t \odot oldsymbol{c}_{t-1} + oldsymbol{i}_t \odot ilde{oldsymbol{c}}_t$ Similar to GRU's hidden state, we can see that LSTM's cell state recurrence resembles the afore-mentioned parallel scan's formulation $v_t = a_t \odot v_{t-1} + b_t$ where $a_t \leftarrow f_t, b_t \leftarrow i_t \odot \tilde{c}_t$, and $v_t \leftarrow c_t$. However, f_t , i_t and \tilde{c}_t are dependent on the previous hidden state h_t . As such, LSTM's cell state recurrence is unable to apply the parallel scan algorithm as is. We can address this in a similar fashion to GRU by removing their hidden state dependencies as follows: $\begin{array}{ll} \boldsymbol{f}_t = \sigma(\operatorname{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) & \boldsymbol{f}_t = \sigma(\operatorname{Linear}_{d_h}(\boldsymbol{x}_t)) \\ \boldsymbol{i}_t = \sigma(\operatorname{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) & \Rightarrow & \boldsymbol{i}_t = \sigma(\operatorname{Linear}_{d_h}(\boldsymbol{x}_t)) \\ \boldsymbol{\tilde{c}}_t = \operatorname{tanh}(\operatorname{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) & \quad \boldsymbol{\tilde{c}}_t = \operatorname{tanh}(\operatorname{Linear}_{d_h}(\boldsymbol{x}_t)) \end{array}$ 3.2.2 STEP 2: DROP RANGE RESTRICTION OF CANDIDATE STATES Similar to GRUs, LSTMs leverage the hyperbolic tangent function (tanh) to restrict the range of

its states between (-1, 1). LSTMs apply the range restriction twice: once when computing the candidate cell state and once when computing its hidden state. In this step, we drop both as follows:

$$egin{array}{ll} ilde{m{c}}_t = anh(ext{Linear}_{d_h}(m{x}_t)) & & & ilde{m{c}}_t = ext{Linear}_{d_h}(m{x}_t) \ m{h}_t = m{o}_t \odot anh(m{c}_t) & & & m{h}_t = m{o}_t \odot m{c}_t \end{array}$$

3.2.3 STEP 3: SIMPLIFYING SCALING OF OUTPUT

Continuing the trend of simplification, we drop the output gate o_t which scales the hidden state. Without the output gate, the normalized hidden state is equal to the cell state, i.e., $h_t = o_t \odot c_t \Rightarrow$ $h_t = c_t$, making having both a hidden and cell state unnecessary. As such, we drop the cell state as well, resulting in the following modification:

$$egin{aligned} egin{aligned} egin{aligned} eta_t &= oldsymbol{o}_t \in \sigma(\operatorname{Linear}_{d_h}(oldsymbol{x}_t)) \ oldsymbol{c}_t &= oldsymbol{f}_t \odot oldsymbol{c}_{t-1} + oldsymbol{i}_t \odot ildsymbol{ ilde c}_t \ oldsymbol{ ilde t} &= oldsymbol{Linear}_{d_h}(oldsymbol{x}_t) \ oldsymbol{ ilde c}_t &= \operatorname{Linear}_{d_h}(oldsymbol{x}_t) \ oldsymbol{ ilde t} &= \operatorname{Linear}_{d_h}(oldsymbol{x}_t) \end{aligned}
ightarrow egin{aligned} oldsymbol{h}_t &= oldsymbol{f}_t \odot oldsymbol{h}_{t-1} + oldsymbol{i}_t \odot ilde oldsymbol{ ilde h}_t \ oldsymbol{ ilde h}_t &= \operatorname{Linear}_{d_h}(oldsymbol{x}_t) \end{aligned}$$

In many sequence modelling settings (e.g., text generation), the optimization objective/target is timeindependent in scale. Recall LSTM's cell state recurrence $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ where $i_t, f_t \in (0, 1)^{d_h}$, and GRU's hidden state recurrence¹, $h_t^{GRU} = (1 - z_t) \odot h_{t-1}^{GRU} + z_t \odot \tilde{h}_t^{GRU}$ where $z_t \in (0, 1)^{d_h}$. GRUs retain $(1 - z_t) \in (0, 1)$ of the previous hidden state and add z_t of the new candidate state. Since these proportions sum to 1, the model ensures its outputs (i.e., hidden states) are *time-independent* in scale. In contrast, LSTM's forget and input gates are computed independently (e.g., $f_t, i_t \to 1$ or $f_t, i_t \to 0$), making its states *time-dependent* in scale². For tasks where time-independence is important, we can ensure LSTM's output is time-independent in scale by simply normalizing its input and forget gates, i.e., $f'_t, i'_t \leftarrow \frac{f_t}{f_t+i_t}, \frac{i_t}{f_t+i_t}$, ensuring that $f'_t + i'_t = 1$ and the scale of LSTM's state is time-independent.

3.2.4 MINLSTM

 Combining the three steps results in a minimal version of LSTM (minLSTM):

¹A superscript is added to differentiate GRU's hidden state from LSTM's.

²For example, $c_t \to c_0 + \sum_{i=1}^t \tilde{c}_t$ when $f_{1:t}, i_{1:t} \to 1$, growing in scale as the sequence length increases.



Figure 1: Training runtime (left), speedup (middle), and memory footprint (right) on a T4 GPU for a batch size of 64. In the training runtime plot (left), minGRU, minLSTM, and Mamba lines overlap. These methods are approximately the same in training runtime.

LSTM		
		minLSTM
$egin{aligned} m{h}_t &= m{o}_t \odot anh(m{c}_t) \ m{o}_t &= \sigma(ext{Linear}_{d_h}([m{x}_t,m{h}_{t-1}])) \ m{c}_t &= m{f}_t \odot m{c}_{t-1} + m{i}_t \odot m{ ilde c}_t \ m{f}_t &= \sigma(ext{Linear}_{d_h}([m{x}_t,m{h}_{t-1}])) \ m{i}_t &= \sigma(ext{Linear}_{d_h}([m{x}_t,m{h}_{t-1}])) \ m{ ilde c}_t &= ext{tanh}(ext{Linear}_{d_h}([m{x}_t,m{h}_{t-1}])) \end{aligned}$	⇒	$egin{aligned} m{h}_t &= m{f}_t \odot m{h}_{t-1} + m{i}_t \odot ilde{m{h}}_t \ m{f}_t &= \sigma(ext{Linear}_{d_h}(m{x}_t)) \ m{i}_t &= \sigma(ext{Linear}_{d_h}(m{x}_t)) \ m{ ilde{m{h}}}_t &= ext{Linear}_{d_h}(m{x}_t)) \ m{ ilde{m{h}}}_t &= ext{Linear}_{d_h}(m{x}_t) \end{aligned}$

where time-independent outputs can be achieved using a hidden state recurrence $h_t = f'_t \odot h_{t-1} + i'_t \odot \tilde{h}_t$ with normalized forget f'_t and input i_t gates computed as $f'_t, i'_t \leftarrow \frac{f_t}{f_t + i_t}, \frac{i_t}{f_t + i_t}$.

The resulting model is significantly more efficient than the original LSTM, requiring only $O(3d_hd_x)$ parameters compared to LSTM's $O(4d_h(d_x + d_h))$. Considering state expansion (i.e., $d_h = \alpha d_x$, where $\alpha \ge 1$), minLSTM uses approximately 38%, 25%, 19%, or 15% of the parameters of a LSTM when $\alpha = 1, 2, 3$, or 4 respectively.

Additionally, the minimal version of LSTM can now be trained in parallel using the parallel scan algorithm, bypassing the need for backpropagation through time (BPTT). Pseudocode and a simple PyTorch implementation are included in the Appendix.

4 WERE RNNS ALL WE NEEDED?

In this section, we compare the minimal versions (minLSTMs and minGRUs) with their traditional counterparts (LSTMs and GRUs) and modern sequence models. Pseudocode, PyTorch implementation, and detailed information regarding the experiment setup are available in the Appendix.

312 4.1 MINIMAL LSTMS AND GRUS ARE EFFICIENT

At test time, recurrent sequence models are typically rolled out sequentially, which makes inference relatively efficient. However, the main bottleneck for traditional RNNs lies in their training, which requires linear time due to backpropagation through time (BPTT). This computational inefficiency contributed to the eventual deprecation of many earlier RNN-based models.

In this section, we compare the resource requirements for training traditional RNNs (LSTM and GRU), their simplified counterparts (minLSTM and minGRU)³, and Mamba (using the official implementation), a recent popular recurrent sequence model.

For these experiments, a fixed batch size of 64 was used while varying the sequence length. We measure both the total runtime and memory complexity involved in performing a forward pass,

³See Appendix for the PyTorch implementations of minLSTM and minGRU written in a few lines.

324
 325
 326
 326
 327
 328
 329
 329
 320
 320
 320
 321
 322
 322
 323
 324
 324
 325
 326
 326
 326
 327
 328
 328
 329
 329
 320
 320
 321
 321
 322
 322
 323
 324
 325
 326
 326
 326
 326
 327
 328
 328
 328
 329
 329
 320
 320
 321
 321
 322
 321
 322
 321
 322
 321
 322
 321
 322
 321
 322
 321
 321
 322
 321
 322
 322
 322
 322
 323
 324
 325
 325
 325
 325
 326
 321
 321
 322
 321
 322
 321
 322
 321
 322
 321
 322
 321
 322
 321
 321
 322
 322
 322
 322
 322
 323
 324
 325
 325
 325
 324
 325
 325
 325
 325
 325
 326
 326
 325
 326
 326
 326
 326
 325
 326
 326
 326
 326
 326
 326

Runtime. We would like to highlight that inference speed can vary depending on hardware and 327 implementation. PyTorch's built-in RNNs are highly optimized low-level GPU implementations. 328 For a more fair comparison, in these experiments, minGRU, minLSTM, GRU, and LSTM were all written in plain Pytorch. In terms of runtime (see Figure 1 (left)), the simplified versions of LSTM 330 and GRU (minLSTM and minGRU) Mamba achieve similar runtimes. Averaging over 100 runs, the 331 runtime for sequence lengths of 512 for minLSTM, minGRU, and Mamba were 2.97, 2.72, and 2.71 332 milliseconds respectively. For a sequence with length 4096, the runtime were 3.41, 3.25, and 3.15333 respectively. In contrast, the traditional RNN counterparts (LSTMs and GRUs) required a runtime 334 that scaled linearly with respect to sequence length. For a sequence length of 512, minGRUs and minLSTMs were $175 \times$ and $235 \times$ faster per training step (see Figure 1 (middle)) than GRUs and 335 LSTMs on a T4 GPU. The improvement is even more significant as sequences grow in length with 336 minGRUs and minLSTMs being $1324 \times$ and $1361 \times$ faster for a sequence length of 4096. As such, 337 in a setting where minGRU would take a day to finish training for a fixed number of epochs, its 338 traditional counterpart GRU could take over 3 years. 339

Memory. By leveraging a parallel scan algorithm to compute the outputs in parallel efficiently, minGRU, minLSTM, and Mamba create a larger computational graph, thus needing more memory compared to traditional RNNs (see Figure 1 (right)). The minimal variants (minGRU and minL-STM) use ~ 88% more memory compared to their traditional counterparts (GRU and LSTM).
 Mamba uses 56% more memory compared to minGRU. In practice, however, runtime is often the bottleneck when training RNNs.

Effect of removing h_{t-1} . The original LSTM and GRU compute their various gates using their inputs x_t and previous hidden states h_{t-1} . These models leverage their time-dependent gates to learn complex functions. However, minLSTM and minGRU's training efficiencies are achieved by dropping their gates' dependencies on the previous hidden states h_{t-1} . As a result, minLSTM and minGRU's gates are dependent only on their inputs x_t , resulting in a simpler recurrent module. As such, the gates of a model consisting of a single layer of minLSTM or minGRU are *time-independent*

due to being conditioned on *time-independent* inputs $x_{1:n}^{(1)}$.

353 However, in deep learning, models are constructed by stacking modules. Although the inputs to the first layer 354 $x_{1:n}^{(1)}$ is *time-independent*, its outputs $h_{1:n}^{(1)}$ are *time-dependent* and are used as the inputs to the second layer, 355 356 i.e., $\boldsymbol{x}_{1:n}^{(2)} \leftarrow \boldsymbol{h}_{1:n}^{(1)}$. As such, beginning from the second 357 358 layer onwards, minLSTM and minGRU's gates will also 359 be time-dependent, resulting in the modelling of more 360 complex functions. In Table 1, we compare the performance of the models with varying numbers of layers on 361 the Selective Copying Task from the Mamba paper (Gu 362 & Dao, 2024). We can immediately see the impact of the time dependencies: increasing the number of layers 364 to 2 or more drastically increases performance.

Model	# Layers	Accuracy
	1	37.6 ± 2.0
MinLSTM	2	85.7 ± 5.8
	3	96.0 ± 2.8
	1	37.0 ± 2.3
MinGRU	2	96.8 ± 3.2
	3	99.5 ± 0.2

Table 1: Comparison of the number of layers on the Selective Copying Task (Gu & Dao, 2024).

Training Stability. Another effect of the number of lay-

ers is increased stability with decreased variance in the accuracy as the number of layers increases
(see Table 1). Furthermore, although minLSTM and minGRU both solve the Selective Copying
task, we can see that minGRU is an empirically more stable method than minLSTM, solving the
task with more consistency and lower variance. minLSTM discards old information and adds new
information, controlling the ratio with two sets of parameters (forget and input gate). During training, the two sets of parameters are tuned in different directions, making the ratio harder to control
and optimize. In contrast, minGRU's discarding and adding of information is controlled by a single
set of parameters (update gate).

- 374
- 375
- 376 377

4.2 MINIMAL RNNS PERFORM SURPRISINGLY WELL

378	Dataset	DT	DS4	DAaren	DMamba	minLSTM	minGRU
379	HalfCheetah-M	42.6	42.5	42.2	42.8	42.7 ± 0.7	43.0 ± 0.4
380	Hopper-M	68.4	54.2	80.9	83.5	85.0 ± 4.4	79.4 ± 8.2
381	Walker-M	75.5	78.0	74.4	78.2	72.0 ± 7.5	73.3 ± 3.3
382	HalfCheetah-M-R	37.0	15.2	37.9	39.6	38.6 ± 1.1	38.5 ± 1.1
383	Hopper-M-R	85.6	49.6	77.9	82.6	88.5 ± 4.7	90.5 ± 0.9
384	Walker-M-R	71.2	69.0	71.4	70.9	69.7 ± 10.7	72.8 ± 8.9
385	HalfCheetah-M-E	88.8	92.7	75.7	91.9	85.4 ± 1.7	86.3 ± 0.5
386	Hopper-M-E	109.6	110.8	103.9	111.1	110.3 ± 1.6	109.7 ± 2.7
387	Walker-M-E	109.3	105.7	110.5	108.3	110.3 ± 0.5	110.3 ± 0.4
388	Average	76.4	68.6	75.0	78.8	78.1	78.2
389							

Table 3: Reinforcement Learning results on the D4RL (Fu et al., 2020) datasets. We report the expert normalized returns (higher is better), following (Fu et al., 2020), averaged across five random seeds. The minimal versions of LSTM and GRU, minLSTM and minGRU outperform Decision S4 (David et al., 2023) and perform comparably with Decision Mamba (Ota, 2024), (Decision) Aaren (Feng et al., 2024) and Decision Transformer (Chen et al., 2021).

In this section, we focus on the empirical performance of these minimal versions of decades-old
models LSTMs (1997) and GRUs (2014), comparing them to several modern sequence models.
It is important to note that the primary goal of our work is not to attain the best performance on
specific tasks but to demonstrate that simplifying traditional architectures can yield competitive
results, comparable to those of recent sequence models.

402 Selective Copy. We begin by considering the Selec-403 tive Copying task, originally introduced in the influ-404 ential Mamba paper (Gu & Dao, 2024). This task 405 served as a key benchmark that demonstrated the improvements made by Mamba's state-space model, S6, 406 over previous state-of-the-art models such as S4 (Gu 407 et al., 2021) and Hyena (Poli et al., 2023). The task 408 requires models to perform content-aware reasoning, 409 where they must selectively memorize relevant tokens 410 while filtering out irrelevant ones. 411

390

391

392

393

394 395 396

In Table 2, we compare the simplified versions of 412 LSTMs and GRUs (minLSTM and minGRU) with sev-413 eral well-known recurrent sequence models that can 414 be trained in parallel, including S4 (Gu et al., 2021), 415 H3 (Fu et al., 2023), Hyena (Poli et al., 2023), and 416 Mamba (S6) (Gu & Dao, 2024). The results for these 417 baselines are directly quoted from the Mamba paper. 418 Among these, only Mamba's S6 model succeeds in 419 solving the task.

Model	Layer	Accuracy
H3	Hyena	30.1
Mamba	Hyena	28.4
S4	S4	18.3
H3	S 4	57.0
Mamba	S 4	56.4
S4	S 6	97.0
H3	S 6	99.7
Mamba	S 6	99.8
minGRU	minGRU	99.5 ± 0.2
minLSTM	minLSTM	96.0 ± 2.8

Table 2: Selective Copy Task. minL-STM, minGRU, and Mamba's S6 (Gu & Dao, 2024) are capable of solving this task. Other methods such as S4, H3, and Hyena at best only partially solve the task.

- 420 Both minGRU and minLSTM are able to solve the Se-
- lective Copying task as well, achieving performance
- comparable to S6 and surpassing the other modern baselines, highlighting the effectiveness of these
 traditional models LSTMs and GRUs, which utilize content-aware gating mechanisms.

Reinforcement Learning. Next, we consider the MuJoCo locomotion tasks from the D4RL bench mark (Fu et al., 2020). Specifically, we consider the three environments: HalfCheetah, Hopper, and
 Walker. For each environment, the models are trained on three datasets of varying data quality:
 Medium (M), Medium-Replay (M-R), and Medium-Expert (M-E).

In Table 3, we compare minLSTM and minGRU with various Decision Transformer variants, including the original Decision Transformer (DT) (Chen et al., 2021), Decision S4 (DS4) (David et al., 2023), Decision Mamba (Ota, 2024), and (Decision) Aaren (Feng et al., 2024). The baseline results are retrieved from the Decision Mamba and Aaren papers. minLSTM and minGRU

432 outperform Decision S4 and achieve performance competitive with Decision Transformer, Aaren, 433 and Mamba. Unlike other recurrent methods, Decision S4 is a model whose recurrence transitions 434 are not input-aware, affecting their performance. In terms of average score across the $3 \times 3 = 9$ 435 datasets, minLSTM and minGRU outperform all the baselines except for Decision Mamba where 436 the difference is marginal.



Figure 2: Language Modelling results on the Shakespeare dataset. Minimal versions of decadeold RNNs (LSTMs and GRUs) performed comparably to Mamba and Transformers. Transformers required $\sim 2.5 \times$ more training steps to achieve comparable performance, overfitting eventually.

452 Language Modelling. Finally, we consider a language modelling task. In this setting, we train a character-level GPT on the works of Shakespeare using the nanoGPT (Karpathy, 2022) framework. In Figure 2, we plot the learning curves with a cross-entropy loss comparing the proposed minimal LSTM and GRU (minLSTM and minGRU) with Mamba and Transformers. We found that minGRU, minLSTM, Mamba, and Transformers achieved comparable test losses of 1.548, 1.555, 456 1.575, and 1.547 respectively. Mamba performed slightly worse than the other models but trained faster, particularly in the early stages, achieving its best performance at 400 steps while minGRU and minLSTM continued training until 575 and 625 steps respectively. In contrast, Transform-459 ers trained significantly slower, requiring 2000 steps ($\sim 2.5 \times$) more training steps than minGRU 460 to achieve comparable performance, making it significantly slower and more resource-intensive to train (quadratic complexity compared to minGRU, minLSTM, and Mamba's linear complexity).

437 438

439

440

441

442

443

444

445

446

447 448

449

450

451

453

454

455

457

458

5 **RELATED WORK**

465 In this section, we provide a brief overview of recent efficient recurrent sequence models that have 466 demonstrated strong empirical performance, rivalling Transformers, while offering better scalability. 467 For a more comprehensive discussion on the resurgence of efficient recurrent models, we refer the 468 reader to recent surveys (Tiezzi et al., 2024). Broadly speaking, these models have evolved in three 469 key directions:

470 (Deep) State-Space Models (SSMs). Building on continuous-time linear systems, Gu et al. (2021) 471 introduced S4, a state-space model that can be unrolled like an RNN during inference and trained 472 similarly to a convolutional neural network. S4's success paved the way for numerous subsequent 473 developments in the field (Gu et al., 2022; Gupta et al., 2022; Hasani et al., 2023; Smith et al., 2023) 474 and their applications across various domains such as language processing (Mehta et al., 2023) and 475 audio analysis (Goel et al., 2022). More recently, Mamba emerged as a significant breakthrough in 476 SSMs, surpassing previous models and attracting considerable attention. One of the key innovations 477 in Mamba was the introduction of S6, a state-space model with input-dependent transition matrices, contrasting with earlier models that used input-independent transition matrices. The success of 478 Mamba and other state-space models has led to the publication of several comprehensive surveys on 479 the topic (Wang et al., 2024b; Patro & Agneeswaran, 2024; Qu et al., 2024). 480

481 Recurrent Versions of Attention. Another popular direction is that of attention, specifically related 482 to linear attention (Katharopoulos et al., 2020). For example, Sun et al. (2023) and Qin et al. (2023) 483 introduced linear attention models that use an input-independent gating mechanism (decay factor). In contrast, Katsch (2023) and Yang et al. (2024) proposed linear attention variants that use input-484 dependent gating. Recently, Feng et al. (2024) showed that softmax attention can also be viewed as 485 an RNN and proposed a recurrent model based on their RNN formulation.

486 **Parallelizable RNNs.** Our work is closely related to several notable papers that parallelize RNNs. 487 Bradbury et al. (2017) modified classical gated RNNs to leverage convolutional layers for efficiency, 488 applying them temporally. Martin & Cundy (2018) demonstrated that RNNs with linear dependen-489 cies can be efficiently trained using a parallel scan. Building on this work, the authors introduced 490 GILR, a gated linear RNN, where the outputs can serve as a surrogate for the previous state dependencies in traditional RNNs (e.g., LSTMs), enabling parallel training. Notably, minGRU is equiv-491 alent to GILR but without an activation function. More recently, Orvieto et al. (2023) proposed 492 a linear gated RNN that leverages complex diagonal recurrences and an exponential parameteri-493 zation, achieving comparable performance to state-space models. Qin et al. (2024b) introduced 494 HGRN whose token mixer HGRU is a linear gated RNN augmented with complex value (polar co-495 ordinate) recurrences, lower bounds on their forget gate, and an output gate. HGRN2 (Qin et al., 496 2024a) improved HGRN by incorporating state expansion. Beck et al. (2024) extends LSTM using 497 exponential gating and a normalizer state. Their xLSTM consists of parallelizable (mLSTM) and 498 sequential-only (sLSTM) versions. mLSTM removes the hidden state dependencies to enable paral-499 lelization, introduces a matrix memory cell, and uses a query vector for retrieval from the memory. 500 Zhu et al. (2024b) builds on insights from HGRN and revisits GRUs, introducing a parallelizable 501 token mixer that removes matrix multiplications and leverages ternary weight quantization.

502 503

504

6 CONCLUSION

505 In this work, we revisited the history of sequence modelling, focusing on traditional RNNs, specif-506 ically LSTMs (1997) and GRUs (2014), which dominated the field for two decades before the rise 507 of Transformer models. We demonstrated that we can enable parallel training of traditional RNNs 508 by removing their gates' dependencies on previous states. Further simplification of these archi-509 tectures led to minimal versions—minLSTMs and minGRUs—which offer several advantages: (1) 510 fewer parameters than their traditional counterparts, (2) full parallelizability during training, and 511 (3) surprisingly competitive performance across a range of tasks, rivalling modern models despite 512 their simplicity. In the appendix, we provide implementations of minGRU and minLSTM in plain 513 PyTorch, requiring only a few lines of code. This makes them lightweight and accessible to beginners, practitioners, and researchers alike. We hope this work sparks a broader conversation about 514 the evolution of sequence modelling, encouraging a reevaluation of simpler foundational models 515 like LSTM and GRU in light of newer, more complex architectures. Given the surprising effective-516 ness of these minimal versions of decades-old RNNs, alongside the recent success of modern RNN 517 architectures, we pose the question: "Were RNNs all we needed?" 518

519

520 LIMITATIONS

522 Modern models such as Mamba and xLSTM were run on modern A100 GPUs with 80 GB of mem-523 ory. In contrast, our experiments were conducted on older GPUs (i.e., P100, T4, and Quadro 5000) 524 with only 16 GB of memory (roughly 20% of the memory available to the other models). These 525 hardware constraints impacted our ability to perform large-scale experiments. To accommodate the memory limitations, we used gradient accumulation for training some tasks, reducing the effective 526 batch size by half, which resulted in significantly slower training times. While this approach al-527 lowed us to run experiments within the available memory constraints, it also limited the scale of our 528 evaluations. 529

Despite these limitations, we believe that the conclusions drawn from our experiments are likely
 to generalize to larger-scale settings. The minimal versions of traditional RNNs share fundamen tal similarities with many recent recurrent sequence models (e.g., input-dependent gating), which
 suggests that their performance would likely be consistent on larger datasets given sufficient com putational resources.

535

536 REFERENCES 537

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova,
 Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended
 long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

555

562

569

575

580

- Guy E Blelloch. Prefix sums and their applications. *Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University*, 1990.
- James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural net works. In *International Conference on Learning Representations*, 2017.
- Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.
- 549 Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and
 550 Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical ma 551 chine translation. In *EMNLP*, 2014.
- Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.
- Shmuel Bar David, Itamar Zimerman, Eliya Nachmani, and Lior Wolf. Decision s4: Efficient
 sequence-based rl via state spaces layers. In *The Eleventh International Conference on Learning Representations*, 2023.
- Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural networks and the chomsky hierarchy. In *The Eleventh International Conference on Learning Representations*, 2023.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas
 Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszko reit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at
 scale. *ICLR*, 2021.
- Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 0364-0213.
- Leo Feng, Frederick Tung, Hossein Hajimirsadeghi, Mohamed Osama Ahmed, Yoshua Bengio, and
 Greg Mori. Attention as an rnn. *arXiv preprint arXiv:2405.13956*, 2024.
- 572 Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re.
 573 Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh*574 *International Conference on Learning Representations*, 2023.
- Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.
- Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. It's raw! audio generation with state space models. In *International Conference on Machine Learning*, pp. 7616–7633. PMLR, 2022.
- Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2024.
- Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.
- 587 Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured 588 state spaces. In *International Conference on Learning Representations*, 2021.
- Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022.
- 593 Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022.

594 595 596	Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid structural state-space models. In <i>The Eleventh International Conference on Learning Representations</i> , 2023.
597 598 599	Franz A Heinsen. Parallelization of an ubiquitous sequential computation. <i>arXiv preprint arXiv:2311.06281</i> , 2023.
600 601	S Hochreiter and J Schmidhuber. Long short-term memory. <i>Neural Computation</i> , 9(8):1735–1780, 1997.
602 603 604 605	John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. <i>nature</i> , 596(7873):583–589, 2021.
606	Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.
607 608 609 610	Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In <i>International conference on machine learning</i> , pp. 5156–5165. PMLR, 2020.
611 612	Tobias Katsch. Gateloop: Fully data-controlled linear recurrence for sequence modeling. <i>arXiv</i> preprint arXiv:2311.01927, 2023.
613 614	Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In <i>International Conference on Learning Representations</i> , 2019.
615 616 617	A Krizhevsky. Learning multiple layers of features from tiny images. <i>Master's thesis, University of Tront</i> , 2009.
618 619	Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. In <i>International Conference on Learning Representations</i> , 2018.
620 621 622 623	Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long range language model- ing via gated state spaces. In <i>The Eleventh International Conference on Learning Representations</i> , 2023.
624 625	William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In <i>Forty-first International Conference on Machine Learning</i> , 2024.
626 627 628 629 630 631	Nikita Nangia and Samuel Bowman. ListOps: A diagnostic dataset for latent tree learning. In Silvio Ricardo Cordeiro, Shereen Oraby, Umashanthi Pavalanathan, and Kyeongmin Rim (eds.), <i>Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop</i> , pp. 92–99, New Orleans, Louisiana, USA, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-4013. URL https://aclanthology.org/N18-4013.
632 633 634	Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pas- canu, and Soham De. Resurrecting recurrent neural networks for long sequences. In <i>International</i> <i>Conference on Machine Learning</i> , pp. 26670–26698. PMLR, 2023.
636 637	Toshihiro Ota. Decision mamba: Reinforcement learning via sequence modeling with selective state spaces. <i>arXiv preprint arXiv:2403.19925</i> , 2024.
638 639 640	Badri Narayana Patro and Vijay Srinivas Agneeswaran. Mamba-360: Survey of state space models as transformer alternative for long sequence modelling: Methods, applications, and challenges. <i>arXiv preprint arXiv:2404.16112</i> , 2024.
642 643 644	Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. <i>arXiv preprint arXiv:2305.13048</i> , 2023.
645 646 647	Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In <i>International Conference on Machine Learning</i> , pp. 28043–28078. PMLR, 2023.

648 Zhen Qin, Dong Li, Weigao Sun, Weixuan Sun, Xuyang Shen, Xiaodong Han, Yunshen Wei, Bao-649 hong Lv, Fei Yuan, Xiao Luo, et al. Scaling transnormer to 175 billion parameters. arXiv preprint 650 arXiv:2307.14995, 2023. 651 Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. 652 Hgrn2: Gated linear rnns with state expansion. Conference on Language Modeling, 2024a. 653 654 Zhen Qin, Songlin Yang, and Yiran Zhong. Hierarchically gated recurrent neural network for se-655 quence modeling. Advances in Neural Information Processing Systems, 36, 2024b. 656 Haohao Qu, Liangbo Ning, Rui An, Wenqi Fan, Tyler Derr, Xin Xu, and Qing Li. A survey of 657 mamba. arXiv preprint arXiv:2408.01129, 2024. 658 659 Dragomir R. Radev, Pradeep Muthukrishnan, and Vahed Oazvinian. The ACL Anthology network 660 corpus. In Min-Yen Kan and Simone Teufel (eds.), Proceedings of the 2009 Workshop on Text and 661 Citation Analysis for Scholarly Digital Libraries (NLPIR4DL), pp. 54-61, Suntec City, Singapore, 662 August 2009. Association for Computational Linguistics. URL https://aclanthology. org/W09-3607. 663 664 Jimmy TH Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for se-665 quence modeling. In The Eleventh International Conference on Learning Representations, 2023. 666 667 Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. arXiv 668 preprint arXiv:2307.08621, 2023. 669 670 Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, 671 Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient 672 transformers. In International Conference on Learning Representations, 2021. 673 Matteo Tiezzi, Michele Casoni, Alessandro Betti, Marco Gori, and Stefano Melacci. State-space 674 modeling in long sequence processing: A survey on recurrence in the transformer era. arXiv 675 preprint arXiv:2406.09062, 2024. 676 677 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, 678 Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural informa-679 tion processing systems, 30(2017), 2017. 680 Chloe Wang, Oleksii Tsepa, Jun Ma, and Bo Wang. Graph-mamba: Towards long-range graph 681 sequence modeling with selective state spaces. arXiv preprint arXiv:2402.00789, 2024a. 682 683 Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention 684 with linear complexity. arXiv preprint arXiv:2006.04768, 2020. 685 Xiao Wang, Shiao Wang, Yuhe Ding, Yuehang Li, Wentao Wu, Yao Rong, Weizhe Kong, Ju Huang, 686 Shihao Li, Haoxiang Yang, et al. State space model for new-generation network alternative to 687 transformers: A survey. arXiv preprint arXiv:2404.09516, 2024b. 688 689 Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention 690 transformers with hardware-efficient training. In International Conference on Machine Learning, 2024. 691 692 Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vi-693 sion mamba: Efficient visual representation learning with bidirectional state space model. In 694 International Conference on Machine Learning, 2024a. 695 Rui-Jie Zhu, Yu Zhang, Ethan Sifferman, Tyler Sheaves, Yiqiao Wang, Dustin Richmond, Peng 696 Zhou, and Jason K Eshraghian. Scalable matmul-free language modeling. arXiv preprint 697 arXiv:2406.02528, 2024b. 698 699 700

702 A IMPLEMENTATION DETAILS: VANILLA VERSION

Algorithm 1 Sequential Mode: Minimal Version of GRU (minGRU)

Algorithm 2 Parallel Mode: Minimal Version of GRU (minGRU)

 $\boldsymbol{h}_{1:t} \leftarrow \text{ParallelScan}((\boldsymbol{1} - \boldsymbol{z}_{1:t}), [\boldsymbol{h}_0, \boldsymbol{z}_{1:t} \odot \boldsymbol{h}_{1:t}])$

A.1.2 MINLSTM: A MINIMAL LSTM

In this section, we provide the pseudocode and equivalent PyTorch code for minGRU and minL-STM. When performing repeated multiplications such as in many recurrent sequence models, numerical instabilities are common, especially during training. As such, we trained using a log-space implementation (see Section B) for improved numerical stability.

A.1 PSEUDOCODE: VANILLA VERSION

 $oldsymbol{h}_t \leftarrow (1-oldsymbol{z}_t) \odot oldsymbol{h}_{t-1} + oldsymbol{z}_t \odot oldsymbol{h}_t$

711 A.1.1 MINGRU: A MINIMAL GRU

 $\boldsymbol{z}_t \leftarrow \sigma(\operatorname{Linear}_{d_h}(\boldsymbol{x}_t))$

 $\boldsymbol{z}_{1:t} \leftarrow \sigma(\operatorname{Linear}_{d_h}(\boldsymbol{x}_{1:t}))$

 $\boldsymbol{h}_{1:t} \leftarrow \operatorname{Linear}_{d_h}(\boldsymbol{x}_{1:t})$

 $\begin{aligned} & \boldsymbol{f}_t \leftarrow \sigma(\operatorname{Linear}_{d_h}(\boldsymbol{x}_t)) \\ & \boldsymbol{i}_t \leftarrow \sigma(\operatorname{Linear}_{d_h}(\boldsymbol{x}_t)) \\ & \boldsymbol{f}_t', \boldsymbol{i}_t' \leftarrow \frac{\boldsymbol{f}_t}{\boldsymbol{f}_t + \boldsymbol{i}_t}, \frac{\boldsymbol{i}_t}{\boldsymbol{f}_t + \boldsymbol{i}_t} \end{aligned}$

 $\tilde{\boldsymbol{h}}_t \leftarrow \operatorname{Linear}_{d_h}(\boldsymbol{x}_t)$

 $oldsymbol{h}_t \leftarrow oldsymbol{f}_t' \odot oldsymbol{h}_{t-1} + oldsymbol{i}_t' \odot ilde{oldsymbol{h}}_t$

 $\hat{h}_t \leftarrow \operatorname{Linear}_{d_h}(\boldsymbol{x}_t)$

Input: x_t, h_{t-1}

Input: $\boldsymbol{x}_{1:t}, \boldsymbol{h}_0$

Output: $h_{1:t}$

scaling

Input: x_t, h_{t-1}

Output: h_t

Output: h_t

Algorithm 4 Parallel Mode: Minimal Version of LSTM (minLSTM) with length independence scaling

Algorithm 3 Sequential Mode: Minimal Version of LSTM (minLSTM) with length independence

 $\begin{array}{l} \text{Input: } x_{1:t}, h_0 \\ \text{Output: } h_{1:t} \\ f_{1:t} \leftarrow \sigma(\text{Linear}_{d_h}(\boldsymbol{x}_{1:t})) \\ i_{1:t} \leftarrow \sigma(\text{Linear}_{d_h}(\boldsymbol{x}_{1:t})) \\ f_{1:t}', i_{1:t}' \leftarrow \frac{f_{1:t}}{f_{1:t} + i_{1:t}}, \frac{i_{1:t}}{f_{1:t} + i_{1:t}} \\ \tilde{h}_{1:t} \leftarrow \text{Linear}_{d_h}(\boldsymbol{x}_{1:t}) \\ \end{array}$

 $\underline{\boldsymbol{h}_{1:t}} \leftarrow \text{ParallelScan}(\boldsymbol{f}_{1:t}', [\boldsymbol{h}_0, \boldsymbol{i}_{1:t}' \odot \tilde{\boldsymbol{h}}_{1:t}])$

756 A.2 PyTorch Code: Vanilla Version

A.2.1 MINGRU: A MINIMAL GRU

```
def forward(self, x_t, h_prev):
    # x_t: (batch_size, 1, input_size)
    # h_prev: (batch_size, 1, hidden_size)
    z_t = torch.sigmoid(self.linear_z(x_t))
    h_tilde = self.linear_h(x_t)
    h_t = (1 - z_t) * h_prev + z_t * h_tilde
    return h_t
```

Listing 1: Sequential Mode: Minimal Version of GRU (minGRU)

```
771
772
773
774
775
776
```

```
def forward(self, x, h_0):
    # x: (batch_size, seq_len, input_size)
    # h_0: (batch_size, 1, hidden_size)
    z = torch.sigmoid(self.linear_z(x))
    h_tilde = self.linear_h(x)
    h = parallel_scan((1 - z),
        torch.cat([h_0, z * tilde_h], dim=1))
    return h
```

Listing 2: Parallel Mode: Minimal Version of GRU (minGRU)

A.2.2 MINLSTM: A MINIMAL LSTM

```
def forward(self, x_t, h_prev):
    # x_t: (batch_size, 1, input_size)
    # h_prev: (batch_size, 1, hidden_size)
    f_t = torch.sigmoid(self.linear_f(x_t))
    i_t = torch.sigmoid(self.linear_i(x_t))
    tilde_h_t = self.linear_h(x_t)
    f_prime_t = f_t / (f_t + i_t)
    i_prime_t = i_t / (f_t + i_t)
    h_t = f_prime_t * h_prev + i_prime_t * tilde_h_t
```

Listing 3: Sequential Mode: Minimal Version of LSTM (minLSTM) with length independence scaling

```
def forward(self, x, h_0):
    # x: (batch_size, seq_len, input_size)
    # h_0: (batch_size, 1, hidden_size)
f = torch.sigmoid(self.linear_f(x))
    i = torch.sigmoid(self.linear_f(x))
    tilde_h = self.linear_h(x)
    f_prime = f / (f + i)
    i_prime = i / (f + i)
    h = parallel_scan(f_prime,
        torch.cat([h_0, i_prime * tilde_h], dim=1))
    return h
```

Listing 4: Parallel Mode: Minimal Version of LSTM (minLSTM) with length independence scaling

810 IMPLEMENTATION DETAILS: LOG-SPACE VERSION (ADDITIONAL В 811 NUMERICAL STABILITY) 812

In this section, we detail the log-space version of minLSTM and minGRU for improved numeri-814 cal stability. During training, the parallel modes are used to avoid backpropagation through time (BPTT), speeding up the training time significantly. At inference time, the sequential modes are 816 used. 817

B.1 PARALLEL SCAN: LOG-SPACE IMPLEMENTATION

Recall that, the parallel scan's objective is to compute $h_{1:t}$ where $h_k = a_k \odot h_{k-1} + b_k$. In code, the vanilla parallel scan function would take as input: coefficients $a_{1:t}$ and values $b_{0:t}$. The function then outputs $h_{1:t}$. For numerical stability, we consider a log-space implementation which takes as input $\log(a_{1:t})$ and $\log(b_{0:t})$ instead and outputs $h_{1:t}$. The code for the parallel scan in log-space is included below and is based on the code by Heinsen (2023).

824 825

813

815

818

819 820

821

822

823

```
826
827
```

```
829
830
```

```
831
832
```

833

834

835 836 837

838 839

840 841

842

```
def parallel_scan_log(log_coeffs, log_values):
    # log_coeffs: (batch_size, seq_len, input_size)
    # log_values: (batch_size, seq_len + 1, input_size)
   a_star = F.pad(torch.cumsum(log_coeffs, dim=1), (0, 0, 1, 0))
   log_h0_plus_b_star = torch.logcumsumexp(
                                log_values - a_star, dim=1)
    log_h = a_star + log_h0_plus_b_star
   return torch.exp(log_h)[:, 1:]
```

Listing 5: Parallel scan based on Heinsen (2023). This function computes $h_{1:t}$ given log coefficients $\log(\boldsymbol{a}_{1:t})$ and $\log \text{ values } \log(\boldsymbol{b}_{0:t})$.

B.2 PSEUDOCODE: LOG-SPACE VERSION

For maximal numerical stability, we rewrite the log-space versions of minGRU and minLSTM.

B.2.1 MINGRU: A MINIMAL GRU

Recall minGRU's recurrence is as follows $h_t \leftarrow (1-z_t) \odot h_{t-1} + z_t \odot h_t$. As such, $a_t \leftarrow (1-z_t)$ 843 and $b_t \leftarrow z_t \odot \tilde{h}_t$ where $z_t = \sigma(k_t)$ and $k_t = \text{Linear}_{d_h}(x_t)$. As a result, $\log(a_t) \leftarrow \log(1 - z_t)$ 844 and $\log(\mathbf{b}_t) \leftarrow \log(\mathbf{z}_t) + \log(\mathbf{h})_t$. We can break down these down as follows: 845

846
$$\log(\boldsymbol{z}_t) = \log(\sigma(\boldsymbol{k}_t))$$
848 $= \log\left(\frac{1}{1 + \exp(-\boldsymbol{k}_t)}\right)$ 849 $= -\operatorname{Softplus}(-\boldsymbol{k}_t)$ 850 $= -\operatorname{Softplus}(-\boldsymbol{k}_t)$ 851 $\log(1 - \boldsymbol{z}_t) = \log\left(\frac{\exp(-\boldsymbol{k}_t)}{1 + \exp(-\boldsymbol{k}_t)}\right)$ 853 $= \log\left(\frac{1}{1 + \exp(\boldsymbol{k}_t)}\right)$ 854 $= -\operatorname{Softplus}(\boldsymbol{k}_t)$

where $k_t = \text{Linear}_{d_h}(x_t)$. However, we need to compute $\log(\tilde{h})_t$ which is inconvenient if \tilde{h}_t 858 has some negative values. We could use complex numbers and a complex number version of the 859 parallel scan, but this would result in the parallel scan increasing in complexity. Instead, we propose 860 to ensure that $h_t > 0$. This be can done in a variety of ways. In our experiments, we added a 861 continuous activation function g replacing $h_t \leftarrow \text{Linear}_{d_h}(x_t)$ with $h_t \leftarrow g(\text{Linear}_{d_h}(x_t))$ where 862 $g(x) = \begin{cases} x + 0.5, & \text{if } x \ge 0\\ \sigma(x), & \text{otherwise} \end{cases} \text{ and its log: } \log(g(x)) = \begin{cases} \log(x + 0.5), & \text{if } x \ge 0\\ -\text{Softplus}(-x), & \text{otherwise} \end{cases}$ 863

At inference time, the sequential mode (Algorithm 5) is used. During training, the parallel mode (Algorithm 6) is used.

Algorithm 5 Sequential Mode: Minimal Version of GRU (minGRU) trained in log-space

Algorithm 6 Parallel Mode: Minimal Version of GRU (minGRU) for training in log-space

Input: $x_{1:t}, h_0$

 $\begin{array}{l} \textbf{Output: } \boldsymbol{h}_{1:t} \\ \text{linear_z} \leftarrow \text{Linear}_{d_h} \\ \text{log_}\boldsymbol{z}_{1:t} \leftarrow -\text{Softplus}(\text{linear_z}(-\boldsymbol{x}_{1:t})) \\ \text{log_coeffs} \leftarrow -\text{Softplus}(\text{linear_z}(\boldsymbol{x}_{1:t})) \\ \text{log_}\boldsymbol{h}_0 \leftarrow \log(h_0) \\ \text{log_}\boldsymbol{\tilde{h}}_{1:t} \leftarrow \text{log_g}(\text{Linear}_{d_h}(\boldsymbol{x}_{1:t})) \\ \boldsymbol{h}_{1:t} \leftarrow \text{ParallelScanLog}(\text{log_coeffs}, [\text{log_}\boldsymbol{h}_0, \text{log_}\boldsymbol{z}_{1:t} + \text{log_}\boldsymbol{\tilde{h}}_{1:t}) \end{array}$

B.2.2 MINLSTM: A MINIMAL LSTM

We also derive minLSTM's (with length independence scaling) log-space formulation as well. Recall minLSTM's (with length independence scaling) recurrence is as follows $h_t \leftarrow f'_t \odot h_{t-1} + i'_t \odot \tilde{h}_t$. As such, $a_t \leftarrow f'_t$ and $b_t \leftarrow i'_t \odot \tilde{h}_t$. As a result, $\log(a_t) \leftarrow \log(f'_t)$ and $\log(b_t) \leftarrow \log(i'_t) + \log(\tilde{h}_t)$.

 $\log(\boldsymbol{f}_t') = \log\left(rac{\boldsymbol{f}_t}{\boldsymbol{f}_t + \boldsymbol{i}_t}
ight)$

 $= \log\left(\frac{1}{1+\frac{i_t}{f}}\right)$

 $= -\log\left(1 + \frac{i_t}{f_t}\right)$

 $= -\log\left(1 + \exp\left(\log\left(rac{m{i}_t}{m{f}_t}
ight)
ight)
ight)$

 $= -\text{Softplus} \left(\log(\mathbf{i}_t) - \log(\mathbf{f}_t) \right)$

 $=-\text{Softplus}\left(\log\left(\frac{i_t}{f_t}\right)\right)$

P10 P11 P12 P13 Recall that i_t and f_t are computed via sigmoid. In other words, $i_t = \sigma(k_t)$ and $f_t = \sigma(p_t)$ where $k_t = \text{Linear}_{d_h}(x_t)$ and $p_t = \text{Linear}_{d_h}(x_t)$. Furthermore, recall in minGRU's derivation we showed that $\log(\sigma(k_t)) = -\text{Softplus}(-k_t)$ Using this, we can simplify the computation as follows:

 $\log(f'_t) = -\text{Softplus} \left(\log(\sigma(k_t)) - \log(\sigma(p_t))\right)$ $= -\text{Softplus} \left(\text{Softplus}(-p_t) - \text{Softplus}(-k_t)\right)$

918 Similarly, we also get that:

Input: x_t, h_{t-1}

 $f_t \leftarrow \sigma(\operatorname{Linear}_{d_h}(\boldsymbol{x}_t))$

 $i_t \leftarrow \sigma(\operatorname{Linear}_{d_h}(x_t))$

 $\tilde{\boldsymbol{h}}_t \leftarrow g(\operatorname{Linear}_{d_h}(\boldsymbol{x}_t))$

 $oldsymbol{h}_t \leftarrow oldsymbol{f}_t' \odot oldsymbol{h}_{t-1} + oldsymbol{i}_t' \odot oldsymbol{ ilde{h}}_t$

 $oldsymbol{f}_t', oldsymbol{i}_t' \leftarrow rac{oldsymbol{f}_t}{oldsymbol{f}_t + oldsymbol{i}_t}, rac{oldsymbol{i}_t}{oldsymbol{f}_t + oldsymbol{i}_t},$

Output: h_t

$$\log(\mathbf{i}_t') = -\text{Softplus}(\text{Softplus}(-\mathbf{k}_t) - \text{Softplus}(-\mathbf{p}_t)))$$

Combining these derivations, we get the parallel mode (Algorithm 8) for efficient training.

Algorithm 7 Sequential Mode: Minimal Version of LSTM (minLSTM) with length independence scaling trained in log-space

Algorithm 8 Parallel Mode: Minimal Version of LSTM (minLSTM) with length independence scaling for training in log-space

B.3 PYTORCH CODE: LOG-SPACE VERSION

Listing 6: The continuous function g ensures that $h_t \leftarrow g(\text{Linear}_{d_h}(x_t))$ is positive.

B.3.1 MINGRU: A MINIMAL GRU

return h_t

def forward(self, x_t, h_prev):

```
960
961
962
963
964
965
966
967
968
```

Listing 7: Sequential Mode: Minimal Version of GRU (minGRU) trained in log-space

def forward(self, x, h_0):
 # x: (batch_size, seq_len, input_size)

x_t: (batch_size, 1, input_size)

h_tilde = g(self.linear_h(x_t))

h_prev: (batch_size, 1, hidden_size)

z = torch.sigmoid(self.linear_z(x_t))

 $h_t = (1 - z) * h_prev + z * h_tilde$

```
972
                # h_0: (batch_size, 1, hidden_size)
973
     4
974
                k = self.linear_z(x)
     5
975
                \log_z = -F.softplus(-k)
     6
976
                log_coeffs = -F.softplus(k)
     7
                \log_h_0 = \log_g(h_0)
977
     8
                log_tilde_h = log_g(self.linear_h(x))
978
     9
                h = parallel_scan_log(log_coeffs,
    10
979
                         torch.cat([log_h_0, log_z + log_tilde_h], dim=1))
    11
980
                return h
    12
981
982
           Listing 8: Parallel Mode: Minimal Version of GRU (minGRU) for training in log-space
983
984
985
986
987
988
      B.3.2 MINLSTM: A MINIMAL LSTM
989
990
991
           def forward(self, x_t, h_prev):
992
                # x_t: (batch_size, 1, input_size)
993
                # h_prev: (batch_size, 1, hidden_size)
994
995
                f_t = torch.sigmoid(self.linear_f(x_t))
996
                i_t = torch.sigmoid(self.linear_i(x_t))
997
                tilde_h_t = g(self.linear_h(x_t))
998
                f_prime_t = f_t / (f_t + i_t)
999
                i\_prime\_t = i\_t / (f\_t + i\_t)
     ç
1000
                h_t = f_prime_t * h_prev + i_prime_t * tilde_h_t
    10
                return h_t
1001
    11
1002
      Listing 9: Sequential Mode: Minimal Version of LSTM (minLSTM) with length independence
1003
      scaling trained in log-space
1004
1005
1007
1008
           def forward(self, x, h_0):
1009
                # x: (batch_size, seq_len, input_size)
1010
     2
                # h_0: (batch_size, 1, hidden_size)
1011
     3
1012
                diff = F.softplus(-self.linear_f(x))
1013
                                   - F.softplus(-self.linear_i(x))
1014
                log_f = -F.softplus(diff)
1015
                log_i = -F.softplus(-diff)
1016
                log_h_0 = torch.log(h_0)
1017
                log_tilde_h = log_g(self.linear_h(x))
    10
1018
    11
                h = parallel_scan_log(log_f,
1019
    12
                         torch.cat([log_h_0, log_i + log_tilde_h], dim=1))
1020 13
                return h
1021
1022
      Listing 10: Parallel Mode: Minimal Version of LSTM (minLSTM) with length independence scaling
1023
      for training in log-space
```

1026 C DETAILED EXPERIMENT SETUP

1028 In this section, we describe the experiment setup in detail.

1030 1031 C.1 DATASETS

Selective Copying. In this task, the model learns to extract data tokens from a sequence while disregarding noise tokens. Following Gu & Dao (2024), we consider a vocabulary of 16 and sequences of length 4096. Each sequence includes 16 randomly placed data tokens. The remainder of the tokens are noise.

1036
1037
1038
1038
1039
1039
1039
1039
1039
1039
1039
1039
1040
1040
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041
1041</l

Long Range Arena. Our experiments on the Long Range Arena benchmark consist of three sequence modelling tasks with sequence lengths from 1024 to 4000, designed to evaluate architectures on long-range modelling:

- **Retrieval:** Based on the ACL Anthology Network (Radev et al., 2009), the task is to classify whether two citations, represented as integer token sequences, are equivalent. Sequences are of length 4000 with two possible classes.
- ListOps: An extended version of ListOps (Nangia & Bowman, 2018). The task is to compute the result of a nested mathematical expression in prefix notation. Sequences are of length 2048 with ten possible classes.
 - **G-Image:** Based on CIFAR-10 (Krizhevsky, 2009), the task is to predict the class of 32×32 grayscale images (converted from RGB). Sequences are of length 1024 with ten possible classes.

Reinforcement Learning. In this setting, we consider continuous control tasks from the D4RL benchmark (Fu et al., 2020). These tasks based on MuJoCo comprise of three environments with dense rewards: HalfCheetah, Hopper, and Walker. For each environment, three different datasets are considered that have varying level represent varying levels of data quality:

- Medium (M): One million timesteps generated by a policy scoring about one-third of an expert policy's score.
 - Medium-Replay (M-R): A replay buffer from an agent trained to perform like the Medium policy.
- Medium-Expert (M-E): One million timesteps from the Medium policy combined with one million from an expert policy.
- Following Fu et al. (2020), reported scores are normalized such that 100 represents an expert policy performance.

Language Modelling. In this setting, we consider the Shakespeare dataset, comprising a collection of text data derived from the works of William Shakespeare. The training and testing data consists of 1,003,854 and 111,540 tokens respectively.

1074

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1061

1062

1063

1064

1067 1068

1075 C.2 ARCHITECTURE

In our work, the primary goal was to demonstrate that simplified RNN architectures, such as minL STM and minGRU, can perform comparably to modern state-of-the-art sequence models. To achieve
 this, we stick with a minimalistic architecture, following standard practices such as residual connections, normalization, and a downprojection layer for the RNN's expanded hidden states. For more

complex tasks like language modeling and Long Range Arena, standard components (convolutional layer and MLP) are added⁴.

Selective Copying: No additional components.

Chomsky Hierarchy: (Conv4 \rightarrow minRNN), i.e., a convolutional layer with a kernel size of 4 is applied temporally before the minimal RNN.

Long Range Arena: (Conv4 \rightarrow minRNN \rightarrow MLP)

Language Modelling: (Conv4 \rightarrow minRNN \rightarrow MLP)

1091

1092

1093 C.3 Hyperparameters and general experimental details

1094 1095

Selective Copying. For this task, we closely follow the setup of Gu & Dao (2024), training the model for 400k steps with a batch size of 64 and an input dimension of 64. Due to GPU memory constraints, gradient accumulation is applied, where gradients for two batches of size 32 are accumulated before each gradient update and clipped to 1.0. The optimizer used is Adam with a learning rate of 3×10^{-4} alongside early stopping. Each model consists of 3 layers with a dropout rate of 0.1. The minLSTM and minGRU models have an expansion factor of 6. Baseline results are referenced from the Mamba paper.

Long Range Arena. For this benchmark, we closely follow the setup of Beck et al. (2024). For Retrieval, the models consisted of 6 blocks and an embedding dimension of 128 and were trained with a batch size of 64. For ListOps, the models consisted of 8 blocks and an embedding dimension of 128 and were trained with a batch size of 32. For G-Image, the models consisted of 6 blocks and an embedding dimension of 512 and were trained with a batch size of 64. All models were trained for 250k steps using AdamW optimizer with a learning rate of 0.001, weight decay of 0.05, 10% linear warm-up steps, and cosine annealing.

Chomsky Hierarchy. For this benchmark, we closely follow the setup of Beck et al. (2024), training models consisting of two blocks. The models were trained for 500k steps with a batch size of 256 and the AdamW optimizer with a learning rate of 3×10^{-4} and weight decay of 0.01.

Language Modelling. The models are optimized using AdamW with a learning rate of 1×10^{-3} . Each model consists of three layers, a dropout ratio of 0.2, and an embedding dimension of 384. Training is done with 5k steps using a batch size of 64 and evaluated every 25 steps. Gradients are clipped to 0.25. The Transformer is configured with 6 heads. Mamba uses an SSM state expansion factor of 16 and a block expansion factor of 2. Following Mamba, both minLSTM and minGRU utilize an expansion factor of 2 as well.

Reinforcement Learning. We follow the hyperparameter settings outlined by Ota (2024). For Hopper (Medium) and Hopper (Medium-Replay), an embedding dimension of 256 is used, while all other environments utilize an embedding dimension of 128. The learning rate is set to 1×10^{-4} for Hopper (Medium), Hopper (Medium-Replay), and Walker (Medium). For all other environments and datasets, the learning rate is 1×10^{-3} . The models are optimized using AdamW with a weight decay of 1×10^{-4} and a linear warmup for 10,000 steps. Each model consists of 3 layers and has a dropout ratio of 0.1. The models are trained for 100k steps with a batch size of 64. Results for the baselines are referenced from the Mamba and Aaren papers.

- 1127
- 1128
- 1129 1130

Reinforcement Learning: $(minRNN \rightarrow MLP)^5$

¹¹²⁶

⁴There is a trend in modern recurrent sequence models of prepending a convolutional layer (kernel size of
4) before their recurrent unit – for example, see Mamba (Gu & Dao, 2024) and xLSTM (Beck et al., 2024)
Empirically, we found that including this convolutional layer also helped minRNNs.

⁵Note this is equivalent to the standard Decision Transformer framework for (Offline) RL, replacing the self-attention module with minLSTM or minGRU.

1134 D ADDITIONAL EXPERIMENTS

1136

D.1 CHOMSKY HIERARCHY + LONG RANGE ARENA

1137 1138

In this section, we conduct experiments on both the Chomsky Hierarchy (Deletang et al., 2023) and Long Range Arena (Tay et al., 2021) benchmarks, which are well-established in the literature for evaluating sequence models. Together, these benchmarks provide a test of a model's ability to generalize and handle long-range dependencies, which are crucial for modern sequence modelling tasks.

We compare Minimal RNNs against other fully parallelizable models, such as RWKV, Mamba, and xLSTM[1:0] (using its parallelizable mLSTM module). Following Beck et al. (2024), we focus on tasks from the Chomsky Hierarchy where models have achieved at least 30% accuracy, indicating partial solvability. We closely followed the hyperparameter configurations outlined in the xLSTM paper and averaged results over 3 seeds for consistency. The baseline results (accuracy – higher is better) are taken from the xLSTM paper (Figure 4 for Chomsky Hierarchy and Table 6 for Long Range Arena).

Our experiments (Table 4 and extended Table 5) show that Minimal RNNs achieve competitive performance with state-of-the-art models (e.g., Mamba and xLSTM) across all tasks on these benchmarks, outperforming other models such as Retention, Hyena, RWKV, and Llama.

- 1154
- 1155 D.2 INFERENCE RUNTIME COMPARISON
- 1156

In these experiments, we compare the inference speeds of GRU, LSTM, minGRU, minLSTM, and
 Mamba (using the official implementation). It is important to note that inference speed may vary
 depending on the hardware and implementation used.

For this analysis, we tested different batch sizes (8, 16, 32, 64) and sequence lengths (up to 2048). In
Figure 3, we present the average inference speed across 50 runs, taking context tokens into account
before performing inference. Since GRU and LSTM models process context tokens sequentially,
their inference times are considerably slower than those of minGRU, minLSTM, and Mamba, all of
which benefit from parallel processing.

Overall, minLSTM and minGRU show inference speeds comparable to Mamba. Specifically, min-GRU was 6.6%, 4.1%, 4.9%, and 2.9% faster than Mamba for batch sizes of 8, 16, 32, and 64, respectively. On the other hand, minLSTM was 3.6%, 2.9%, 0%, and 1.3% slower than Mamba for the same batch sizes.

Since minLSTM and minGRU are simplifications of LSTM and GRU, we expect them to generally perform faster, including during inference. This is demonstrated in Figure 4, where we compare the inference speed of minLSTM and minGRU with traditional LSTM and GRU models across varying batch sizes. As expected, minGRU and minLSTM are 19.6% and 41.5% faster than GRU and LSTM for a batch size of 64, respectively.

1175

1176 D.3 ARCHITECTURE ABLATION

1177

In our work, the main objective was to demonstrate that simplified RNN architectures, such as minLSTM and minGRU, can perform on par with modern state-of-the-art sequence models. To achieve this, we adopted a minimalistic architectural design, incorporating standard practices such as residual connections, normalization, and a downprojection layer for the RNN's expanded hidden states. For more complex tasks, like language modeling and Long Range Arena, we introduced a convolutional layer and a multi-layer perceptron (MLP).

To better understand the impact of these architectural choices, we conducted an ablation study on the ListOps (Long Range Arena) dataset of these additional components. The results, averaged over 3 seeds, are shown in Table 6. The table highlights the effect of adding different layers to the minLSTM model. For ListOps, incorporating a convolutional layer (Conv) and an MLP resulted in improved performance.

Ν	lethod	Bucket	Missing	Cycle	Even	Majority
Transformers	Llama	0.92	0.08	0.04	1.0	0.37
	Mamba	0.69	0.15	0.86	1.0	0.69
Modern	RWKV-4	0.54	0.21	0.13	1.0	0.63
KININS	xLSTM	0.97	0.33	0.86	1.0	0.74
Minimal RNNs	minLSTM (Ours)	0.94	0.26	0.79	1.0	0.93
N	lethod	Majority Count	Retrieval	ListOps	G-Image	Average
Transformers	Llama	0.13	0.85	0.38	0.54	0.48
Madam	Mamba	0.45	0.90	0.33	0.69	0.64
DNN	RWKV-4	0.13	0.90	0.39	0.69	0.51
NININS	xLSTM	0.46	0.91	0.41	0.70	0.71
Minimal RNNs	minLSTM (Ours)	0.47	0.89	0.59	0.67	0.73

Table 4: Results for Chomsky Hierarchy and Long Range Arena Benchmarks. We compare minLSTM against other fully parallelizable models, including RWKV, Mamba, and xLSTM[1:0]
(using the mLSTM module). The baseline results (accuracy – higher is better) are taken from the xLSTM paper (Figure 4 for Chomsky Hierarchy and Table 6 for Long Range Arena). The results demonstrate that minLSTM achieves competitive performance with state-of-the-art models such as Mamba and xLSTM across all tasks on these benchmarks.

1212

1213 D.4 INITIALIZATION ANALYSES

1215 In this set of experiments, we examine the effect of initialization on the model's performance. De-1216 pending on the task at hand, it may be beneficial to encourage the model to retain information over 1217 time. One way to achieve this is by increasing the bias term in the forget gate of the minLSTM, 1218 which promotes information retention earlier in the training process. As a result, the forget gate f_t 1219 of the LSTM approaches a value of 1 due to this new initialization. As shown in Figure 5, increasing 1220 the forget gate bias in minLSTM enhances training efficiency, leading to faster convergence and 1221 greater stability during training.

1222 1223

1224

E ADDITIONAL RELATED WORK

Parallel Scan. Generalizing across the families of methods (including minLSTM and minGRU), 1225 these recent sequence models can be viewed as members of the same family of functions trainable 1226 via a parallel scan: $v_t = a_t \odot v_{t-1} + b_t$ (see Section 2.3) where a_t and b_t are functions of the input 1227 token x_t . Improving upon the parallel scan algorithm, several models (Yang et al., 2024; Gu & Dao, 1228 2024) such as Mamba have proposed specialized hardware-efficient methods that leverage GPU's 1229 memory hierarchy to reduce high I/O costs and speed up training. In our work, we implemented 1230 minLSTM and minGRU in plain PyTorch. However, due to the structural similarities in recurrences 1231 amongst the numerous methods that leverage parallel scan, many techniques such as chunking that 1232 apply to one work for speeding up training can also apply to others such as minGRU and minLSTM. 1233

Parameter Initializations. Unrolling the recurrences of these new recurrent sequence models over time often results in their outputs and gradients vanishing/exploding (Wang et al., 2024b) due to time dependency in their output's scale. To ensure model stability, the parameters of many models such as state-space models are initialized according to special distributions (Gu et al., 2020; 2022; Orvieto et al., 2023). In contrast, we found that minLSTM and minGRU are already stable using the default PyTorch initialization. Unlike SSMs, minLSTM and minGRU's outputs are time-independent in scale, avoiding potential instabilities.

1	2	4	5
1	2	4	6

Method		Context Sensitive		Regular		xLSTM	
		Bucket	Missing	Cycle	Even	Majority	Majority
		Sort	Duplicate	Nav.	Pairs	Wajonty	Count
Traditional	GRU	0.54	0.29	0.94	1.0	0.60	0.42
RNNs	LSTM	0.99	0.33	1.0	1.0	0.54	0.46
Transformers	Llama	0.92	0.08	0.04	1.0	0.37	0.13
	Mamba	0.69	0.15	0.86	1.0	0.69	0.45
	Retention	0.13	0.03	0.05	0.51	0.36	0.12
Modorn	Hyena	0.3	0.06	0.06	0.93	0.36	0.18
DNN	RWKV-4	0.54	0.21	0.13	1.0	0.63	0.13
KININS	RWKV-5	0.49	0.15	0.26	1.0	0.73	0.34
	RWKV-6	0.96	0.23	0.31	1.0	0.76	0.24
	xLSTM	0.97	0.33	0.86	1.0	0.74	0.46
Minimal RNNs	minLSTM (Ours)	0.94	0.26	0.79	1.0	0.93	0.47

Table 5: Extended Results for Chomsky Hierarchy Benchmark. The baseline results (accuracy — higher is better) are taken from the xLSTM paper (Figure 4). We compare minLSTM against other fully parallelizable models, including RWKV, Mamba, and xLSTM[1:0] (using the mLSTM module). The results demonstrate that minLSTM achieves competitive performance with state-of-the-art models such as Mamba and xLSTM, while outperforming other models like Retention, Hyena, RWKV, and Llama across all tasks in the Chomsky Hierarchy benchmark.

1279		
1280	Model	Accuracy
1281	minLSTM	0.46
1202	minLSTM (+ Conv)	0.45
1284	minLSTM (+ MLP)	0.52
1285	minLSTM (+ Conv + MLP)	0.59

Table 6: Architecture Ablation on the ListOps (Long Range Arena) Dataset. Results (accuracy - higher is better) are averaged over 3 seeds. The table shows the impact of adding different layers to the minLSTM model. For more complex tasks like language modeling and Long Range Arena, we incorporate a convolutional layer (Conv) and a multi-layer perceptron (MLP). The performance increases when these components are added.



Traditional Counterparts (LSTM and GRU). As simplified versions of LSTM and GRU, minL-STM and minGRU generally exhibit faster inference times, particularly with larger batch sizes, as shown in the plots.



Figure 5: **Impact of Forget Gate Bias Initialization on Training Efficiency.** The plot illustrates how increasing the bias of the forget gate in minLSTM enhances training efficiency by promoting earlier retention of information, leading to faster convergence and a more stable training process.