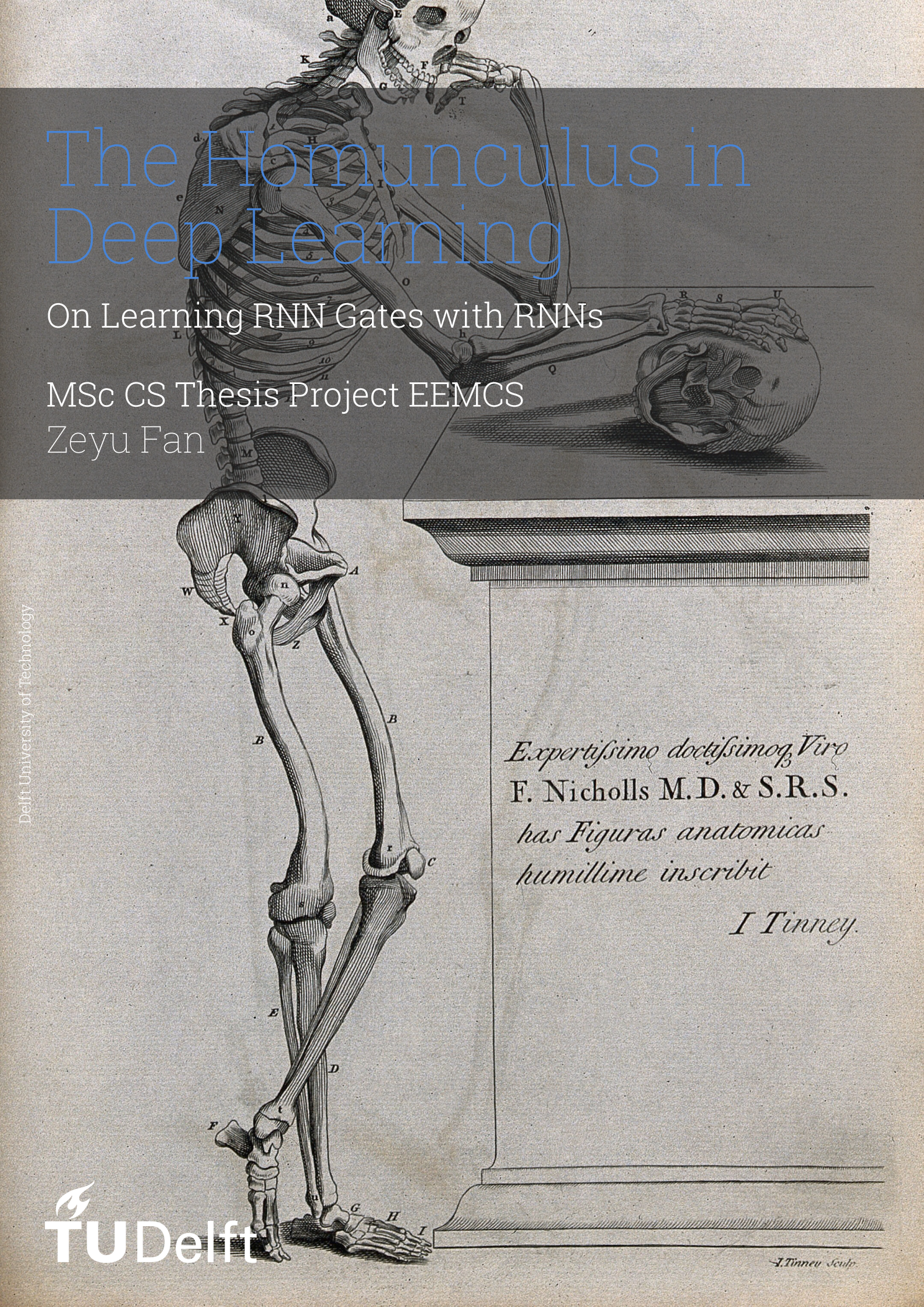
The Homunculus in Deep Learning

On Learning RNN Gates with RNNs

MSc CS Thesis Project EEMCS

Zeyu Fan



Delft University of Technology

TU Delft

The Homunculus in Deep Learning

On Learning RNN Gates with RNNs

by

Zeyu Fan

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday June 29, 2026.

Student number:	6179487	
Project duration:	September 2025 – June 2026	
Thesis committee:	Prof. dr. J.C. van Gemert	TU Delft, Advisor
	ir. A. D. Manolache	TU Delft, Daily Supervisor
	Asst. prof. dr. P. Kellnhofer	TU Delft, External Committee Member

Cover: A skeleton thinking about a skull. Original image from Wellcome Collection, via Rawpixel.
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

This thesis began with a question from my first quarter at TU Delft. In the course *Machine and Deep Learning*, Prof. Jan van Gemert was introducing recurrent neural networks (RNNs), the gradient problem, and gated models as a way to address it. He asked the class whether the gate computation looked familiar, and then showed that it is very similar to standard RNN computation. This made a question cross my mind, and I asked: To solve the gradient problem of RNNs, we use gates, but the gate computation is like another RNN, are we not in a loop?

I still remember how the question was received. Jan repeated it to the class and explained that, in related papers, gates are shown to work both mathematically and empirically. Yet none of them seemed to answer the conceptual question: why should the gate computation be easier to learn than the very similar recurrent computation it is meant to help? Jan's intuition was that the gates may be solving a simpler problem, but that this intuition was not yet a complete explanation. At the time, this was only a short exchange in class. It later became the starting point of this thesis.

Near the end of my first year, I looked through the available thesis topics and found *The Homunculus in Deep Learning*. The topic description explicitly mentioned GRUs and the conceptual loop. It felt like a natural continuation of the earlier classroom curiosity. When I discussed the topic with Jan, he still remembered that question, and the project began from there.

The research process then taught me how large the distance can be between an intuition and a research claim. The original question was easy to ask in a classroom, but making it precise required a long process of formulation, controlled data design, failed attempts, and repeated reinterpretation. The final experiments in this thesis may look simple, but their simplicity lies in how directly they reveal and answer the question, not in how easily they were designed. We had to decide what should remain fixed, what should be varied, and what kind of evidence would actually support or challenge the original intuition. Along the way, the project branched into directions I had not anticipated at the beginning. I also became more sensitive and noticed similar loops appearing in other deep learning architectures, and even in domains beyond computer science, many of which remain largely unaddressed. The more I worked on the project, the more I realized that even the most familiar mechanisms hide deeper complexities than we typically assume.

I am satisfied with the clearer answer I can now give to the question that started this thesis, and with the broader perspective the project has given me. At one point, Jan joked that if a future student asks a similar question in the same course, this work might help him answer it more concretely. I hope this thesis can serve that purpose. For me, it also closes a small yet meaningful circle: a question from the beginning of my Master's study returned at the end as a systematic research project.

Contents

1

Introduction

Scientific explanations often become persuasive by decomposing a difficult problem into smaller parts. Yet some explanations do not truly resolve the original difficulty; instead, they merely move it to another place inside the system. The *homunculus argument* captures this failure of explanation. In its classical form in vision, a theory of perception explains seeing by assuming that light from the outside world forms an image on the retinas in the eyes, and an internal observer in the brain looks at these inner images like watching a movie on a screen, as illustrated in Figure 1.1. However, this immediately raises a new question: how does the internal observer itself see? If the answer again points to another internal observer, the explanation generates an infinite regress rather than a genuine account of perception [17, 21]. The homunculus argument is therefore not only a philosophical curiosity, but also a useful diagnostic tool: it reveals when an explanation silently presupposes the very problem it is supposed to explain.

The idea of a small observer inside the head is compelling because it turns perception into a familiar scene: someone is looking at a screen. Yet the nested observers in Figure 1.1 reveal the problem. Each internal observer inherits the same explanatory burden as the whole system, so the explanation is repeated rather than completed. This thesis uses the homunculus idea as a conceptual lens. The question is not only whether a smaller internal component can help a larger system, but also whether that component genuinely simplifies the problem or merely pushes the difficulty one level inward [7].

A similar diagnostic question arises in deep learning when one learned component is introduced to help a bigger one. This thesis focuses on a concrete example: recurrent neural networks (RNNs) with learned gates. A recurrent neural network processes a sequence one element at a time while maintaining a hidden state that summarizes information from previous steps. When relevant information must be kept in that state over many steps, a standard RNN can struggle to preserve that information [3, 13, 18]. Long short-term memory (LSTM) and gated recurrent units (GRU) were proposed to address this difficulty by introducing learned gates [4, 14]. A gate controls how much of the previous steps should be preserved and how much should be replaced by the current step. As illustrated in Figure 1.2, the gate-controlled additive path (green arrow) can allow information to travel better through steps.

The gates, however, are not predefined rules. At each step, their values must also be computed from the current input and the previous state, using parameters shared across steps: a recurrent structure identical to the bigger network the gates are added to. Gated recurrent networks therefore raise a homunculus-like question. If the main recurrent network struggles to preserve the information, why can the smaller but similar computations that control the gates learn when to keep and when to update that information? The gates may empirically help the RNN learning problem, but explaining by saying that ”the gates decide what to keep” is simply moving the difficulty from the recurrent state to the learned gate controller. This thesis investigates when gated recurrent networks learn useful gate decisions, and how the resulting gate-controlled paths can make long-range learning easier. The importance of this particular problem becomes clearer when considering why recurrent models remain attractive for sequence modeling.

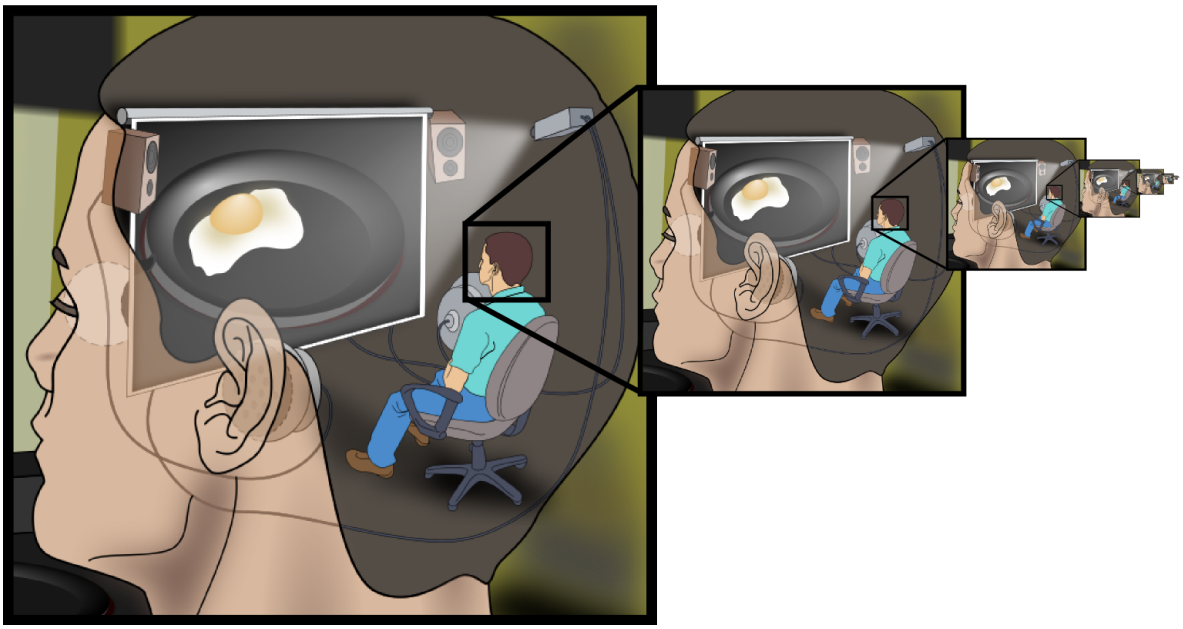


Figure 1.1: Illustration of the homunculus argument. The figure should be read from left to right. Visual perception is explained by positing an internal observer that views an image inside the head. However, the same explanation can then be applied to the internal observer itself, producing a nested sequence of observers. This explanation generates an infinite regress, it postpones the problem of perception instead of resolving it. *Source: Jennifer Garcia, with derivative work by Pbroks13 and Was a bee, Wikimedia Commons, licensed under CC BY-SA 2.5.*

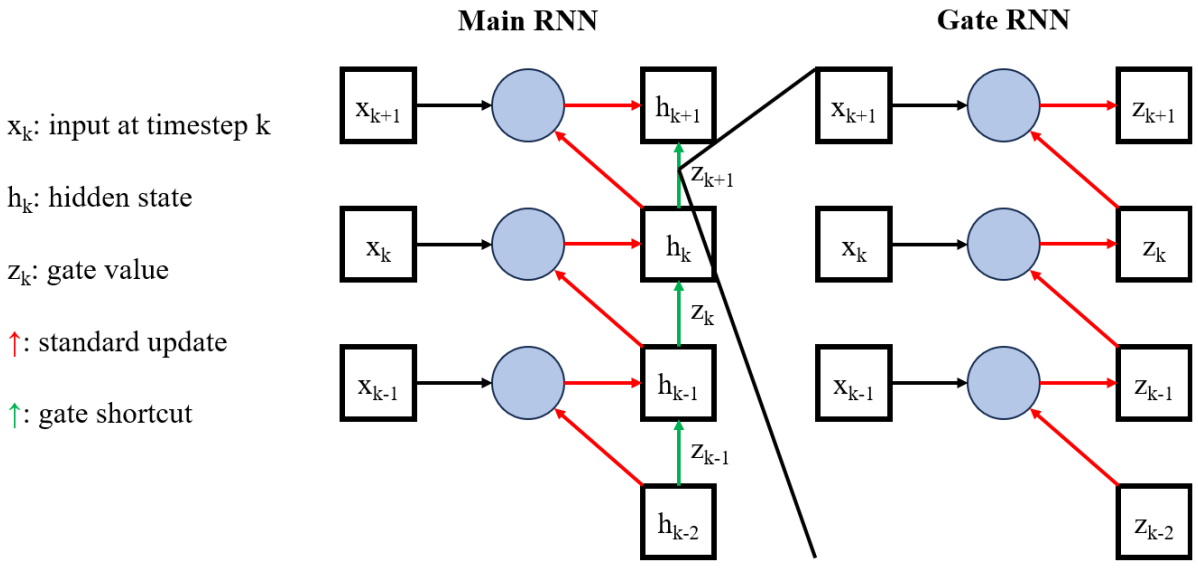


Figure 1.2: The homunculus-like question in gated recurrent networks. The left panel schematically illustrates a recurrent network. Red arrows denote standard recurrent updates, while green arrows denote direct paths controlled by learned gate values. These paths can preserve information through steps. The right panel abstracts the learned computation that produces the gate values. Because the gate decisions must themselves be learned from sequentially propagated information, the explanation raises a homunculus-like question: why is learning the controller easier than learning the original recurrent dynamics? The diagram is a conceptual illustration rather than the exact computational graph of a particular GRU or LSTM.

Sequence modeling is needed whenever the meaning of an input depends on information distributed across time or position. Examples include language, audio, and other time-series data. Transformers have become a dominant architecture for such problems because self-attention allows each position to directly access information from other positions [24]. Direct access is useful for modeling long-range dependencies, but it also comes with a cost: standard self-attention compares pairs of positions, so its computational and memory requirements grow quadratically with the sequence length.

Recurrent models offer a complementary approach. Rather than comparing all pairs of positions, a recurrent model processes a sequence step by step and compresses information into a hidden state. This gives recurrent processing a linear dependence on sequence length and a natural bias toward reusable state updates. Recent work argues that this inductive bias can make recurrent models more data-efficient than Transformers because the same update mechanism can be shared across different sequence lengths [8]. The continued interest in recurrent-style sequence modeling is also visible in modern architectures such as Mamba, which uses selective state spaces for linear-time sequence modeling [6, 12], and in related architectures such as xLSTM [1], RWKV [19], and RetNet [23]. Memory-caching methods [2] further extend recurrent models with a growing set of cached hidden states, providing a middle ground between fixed-size memory and quadratic attention. Recurrent-style models have also become useful in scientific applications: Caduceus [22] builds on Mamba for long-range DNA sequence modeling, where genomic interactions can span very long contexts. These developments show that understanding recurrent networks remains relevant even in an era dominated by Transformers.

This thesis does not propose another recurrent architecture. Instead, it takes the continued relevance of recurrent models as a reason to revisit a more basic question: why do gates help recurrent networks learn long-range dependencies in the first place? The usual explanations [4, 10, 14, 28] describe what useful gates can do after they have been learned, but they do not yet explain how the model learns these useful gate decisions. This thesis studies that missing step in two ways. First, we analyze the learning signal received by gates and show that gate learning can also depend on gradients propagated through long recurrent chains. In other words, the gate that is supposed to protect long-range information must itself be learned from signals that may be attenuated by the same recurrent depth. Second, we empirically test whether the gate learning difficulty appears in controlled sequence tasks. The tasks are designed so that the label depends on only a small number of tokens in the input sequence. The rest are distractors, a useful recurrent state should filter them out while keeping the label-relevant information. When all useful information appears far from the final decision, both vanilla (without gates) and gated recurrent networks can fail. When the training data also contains easier samples with short dependencies, where the informative tokens occur closer to the final decision, gated models can first learn a simpler selective update behavior from shorter gradient paths. Once this behavior is learned, the same model can apply it to harder long-range samples. The experiments then ask whether this learned behavior transfers beyond the specific tokens that received easy samples, and whether the observed improvements are reflected in the model's gradients, gates, and hidden-state changes.

Gated recurrent networks provide a concrete case study of a broader design question in deep learning. Many architectures use a smaller learned component to guide larger decisions. A spatial transformer network [15] uses a localization network to predict how an image or feature map should be spatially transformed before further processing. A task-driven fixation network [25] uses a learned fixation-point generator to select local high-resolution regions for closer inspection. Recent gated-attention models [20] use learned gates to modulate attention outputs in large language models. These architectures differ in their goals and mechanisms, and we do not claim that they share a common explanation. The examples nevertheless expose a recurring question put forward by this thesis: under which conditions does delegating a decision to a smaller learned component make the full problem easier to learn?

The remainder of this thesis is organized into two parts. Part 2 introduces the preliminary material needed to understand the technical side of the study. It explains sequence modeling with recurrent neural networks, backpropagation through time, vanishing and exploding gradients, gated recurrent networks, and the controlled sequence data. Part 3 presents the scientific article. The article formalizes the research questions, describes the experiments, reports the empirical results, and discusses what the controlled findings reveal about learned gates. The preliminary material is intended to make the scientific article accessible to readers with a general machine learning background, while the article itself is written in a publication-style format for readers familiar with deep learning and recurrent models.

2

Preliminary Materials

We assume that readers are familiar with basic neural network terminology, including multi-layer perceptrons, feed-forward networks, activation functions, losses, and gradient-based optimization. A comprehensive introduction to modern deep learning can be found in Goodfellow, Bengio, and Courville (2016). This chapter therefore does not review deep learning from first principles. Instead, it introduces the recurrent models, gradient-flow issues, gated architectures, and controlled sequence tasks that are needed to understand the scientific article in Part 3.

2.1. Sequence Modeling and Recurrent Neural Networks

Sequence modeling concerns data whose elements are ordered. A sequence can be a sentence, an audio signal, a time series, or any list of observations where the meaning of one element depends on previous or later elements. In a standard feed-forward network, an input is usually treated as a fixed-size vector. A sequence can be forced into such a format by padding or truncating it to a fixed length, but this does not solve the main modeling problem. A feed-forward network with position-specific weights would have to learn separately how the same pattern behaves at different positions, and it would not naturally share computations across different sequence lengths. A sequence model should instead use the same rule as it moves through the sequence.

Recurrent neural networks (RNNs) implement this idea by processing a sequence one element at a time while maintaining an internal state. Classical recurrent models were studied by Williams and Zipser [27], and the Elman network introduced a simple and influential form of recurrent hidden-state dynamics [9]. Modern presentations usually describe an RNN as a neural network with a hidden state that is repeatedly updated through time [10, 28]. The hidden state acts as a compressed representation of the sequence prefix that has been observed so far.

Let $x_{1:T} = (x_1, x_2, \ldots, x_T)$ denote a sequence of length T, where $x_t \in \mathbb{R}^{d_x}$ is the input vector at time step t. A vanilla RNN maintains a hidden state $h_t \in \mathbb{R}^{d_h}$. Starting from an initial state h_0, often chosen as a zero vector, the hidden state is updated by

$$h_t = \phi(W_x x_t + W_h h_{t-1} + b_h), \tag{2.1}$$

where $W_x \in \mathbb{R}^{d_h \times d_x}$ maps the current input to the hidden state, $W_h \in \mathbb{R}^{d_h \times d_h}$ maps the previous hidden state to the current hidden state, $b_h \in \mathbb{R}^{d_h}$ is a bias vector, and ϕ is a nonlinear activation function such as $\tanh$. The same parameters W_x, W_h, and b_h are used at every time step. This parameter sharing is what makes the computation recurrent: the model repeatedly applies the same transition rule while the hidden state changes over time.

The hidden state can be used to produce outputs in different ways. In a sequence prediction task, the model may produce an output at every time step,

$$o_t = W_o h_t + b_o, \tag{2.2}$$

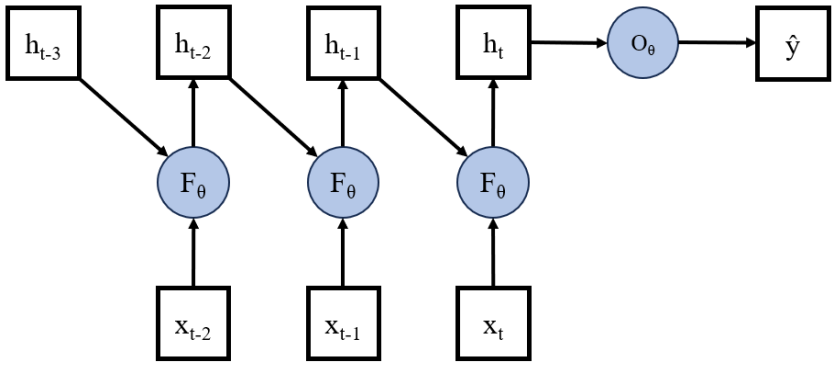


Figure 2.1: Unrolled recurrent computation for many-to-one sequence classification. The figure shows the last few steps of a recurrent neural network. At each time step, the same recurrent transition F_θ (Equation 2.1) receives the current input x_t and the previous hidden state, and produces the next hidden state. The repeated label F_θ indicates parameter sharing across time steps. In a many-to-one classification setting, the final hidden state is passed to an output module O_θ to produce the prediction $\hat{y}$. Thus, any information from earlier inputs that is needed for the prediction must be preserved through the chain of hidden states.

where o_t is the output vector at time step t. In a sequence classification task, the model may instead read the entire sequence and use only the final hidden state,

$$\hat{y} = \mathrm{softmax}(W_y h_T + b_y), \tag{2.3}$$

where $\hat{y}$ is the predicted class distribution. The experiments in this thesis use this many-to-one setting: the recurrent model processes all tokens, and the classifier receives only the final hidden state. This design forces the recurrent state to carry any task-relevant information until the end of the sequence.

The recurrence in Equation 2.1 can be understood by unrolling the model through time, as illustrated in Figure 2.1. After unrolling, an RNN resembles a deep feed-forward network with one layer per time step, except that all recurrent layers share the same parameters. This view explains both the appeal and the difficulty of RNNs. The appeal is that the model can process sequences of different lengths with a fixed set of parameters. The difficulty is that the hidden state is a bottleneck: all information from earlier time steps that remains useful later must be preserved through repeated state updates.

This bottleneck is central to long-range sequence modeling. If a label depends on an early input, then the influence of that input must remain present in the hidden state until the loss is computed. In short sequences, this requirement may be easy to satisfy. In long sequences, however, the same information must survive many recurrent updates. The next section explains why learning such long-range dependencies is difficult for vanilla RNNs: the learning signal itself must be propagated backward through the same long recurrent chain.

2.2. Backpropagation Through Time and Gradient Problem

Training an RNN requires computing how the loss depends on parameters that are reused at every time step. Once an RNN is unrolled through time, this training procedure is ordinary backpropagation applied to the unrolled computation graph. This procedure is called backpropagation through time (BPTT) [10, 26, 28]. The special feature of BPTT is parameter sharing: the same recurrent parameters appear at every time step, so the gradient with respect to these parameters is the sum of their contributions at all time steps.

This thesis mainly considers many-to-one sequence classification, where the loss is computed from the final hidden state h_T. Let L denote the loss for one sequence. For the vanilla RNN update in Equation 2.1, the recurrent parameters affect L through all intermediate hidden states. A parameter change at an early time step can influence the loss only if its effect on the hidden state survives until the final time step. The gradient therefore has to assign credit backward through the recurrent chain.

To see the source of the difficulty, define the recurrent Jacobian

$$J_t = \frac{\partial h_t}{\partial h_{t-1}}. \tag{2.4}$$

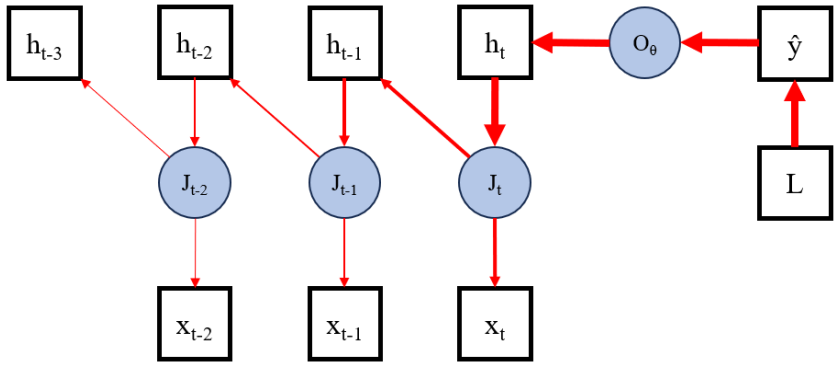


Figure 2.2: Backpropagation through the unrolled recurrent chain. The figure shows the same many-to-one recurrent computation as in Figure 2.1, but from the perspective of the backward pass. The loss L produces a learning signal at the output, which is propagated backward from the final hidden state to earlier hidden states. Each blue node J_i denotes the local recurrent Jacobian $J_i = \partial h_i / \partial h_{i-1}$ of the corresponding recurrent transition. In the backward pass, the learning signal is multiplied by the transposed Jacobians $J_i^\top$ along the red arrows. Thus, the gradient reaching an early state is the result of repeatedly multiplying local Jacobian factors, which explains why gradients can vanish or explode over long temporal distances.

For the vanilla RNN, with pre-activation $a_t = W_x x_t + W_h h_{t-1} + b_h$, this Jacobian is

$$J_t = \operatorname{diag}(\phi'(a_t))\, W_h. \tag{2.5}$$

The gradient of the loss with respect to an earlier hidden state h_k is obtained by repeatedly applying the chain rule:

$$\frac{\partial L}{\partial h_k} = \left(\prod_{i=k+1}^{T} J_i^\top \right) \frac{\partial L}{\partial h_T}. \tag{2.6}$$

Equation 2.6 is visualized in Figure 2.2. The figure shows that the learning signal at time k is not directly connected to the loss. Instead, it is transported backward through a product of Jacobian matrices, one for each recurrent step between k and T.

This product can shrink or grow rapidly as the temporal distance $T - k$ increases. The vanishing case can be seen from a simple norm bound:

$$\left\| \frac{\partial L}{\partial h_k} \right\| \leq \left\| \frac{\partial L}{\partial h_T} \right\| \prod_{i=k+1}^{T} \|J_i\|. \tag{2.7}$$

When the factors in this product are consistently smaller than one, the gradient to early hidden states decays approximately exponentially with distance $T - k$. This is the vanishing gradient problem. In that case when the gradient vanishes, early inputs may be relevant for the target, but the loss provides only a small learning signal for how those early inputs should change the hidden state. The model is then biased toward learning short-range dependencies, because recent states receive much stronger gradient signals than distant states [3, 13, 18].

The same Jacobian product can also cause gradients to grow rather than decay. If the recurrent Jacobian product expands strongly in some direction, the gradient norm can grow very large as it is propagated backward through time. This is the exploding gradient problem. Exploding gradients can make optimization unstable because a single update may move the parameters too far. In practice, gradient clipping is often used to limit the norm of such updates [18]. Clipping can reduce instability caused by large gradients, but it does not solve the vanishing gradient problem, where the signal needed to learn a long-range dependency has already become too small.

The gradient problem is therefore a credit-assignment problem for sequence learning. Suppose the correct label depends on a token near the beginning of the sequence. The forward computation must preserve information about that token until the final hidden state. During training, the backward computation must also send a useful learning signal from the final loss back to the early time step. Vanilla

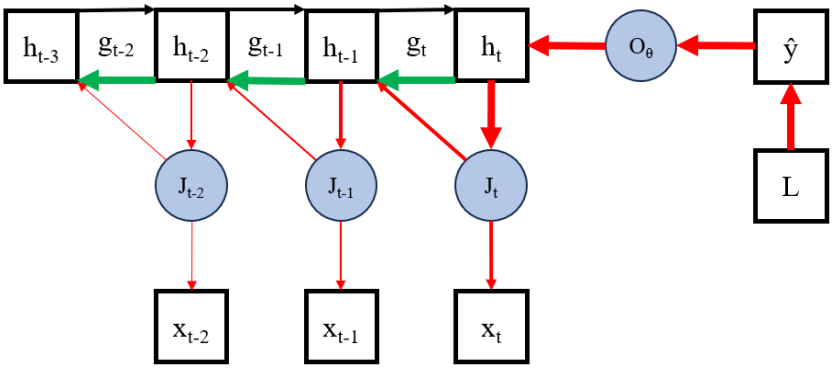


Figure 2.3: Gated direct paths in recurrent networks. The figure extends the previous unrolled views of recurrent computation. Black arrows denote the forward gate-controlled copy path between consecutive recurrent states. Red arrows denote the standard backward learning signal passing through the output layer and the local recurrent Jacobians J_i. Green arrows denote backward gradient routes along the gate-controlled copy path. When the gate values g_i favor copying, part of the recurrent state can be preserved more directly across multiple time steps. This makes it easier to retain information in the forward pass and provides a more direct route for gradient propagation in the backward pass. The figure uses h_t as a generic recurrent state; in an LSTM, the analogous copy path is carried by the cell state c_t.

RNNs struggle with long-range dependencies because both requirements involve a long chain of multiplicative transformations.

This observation motivates gated recurrent networks. If a model can create a path through time that is close to an identity mapping, information can be preserved more easily in the forward pass and gradients can propagate more easily in the backward pass. The next section introduces GRUs and LSTMs as architectures that implement such gated paths. The key question for this thesis remains whether the gates that create these paths can themselves be learned reliably.

2.3. Gated Recurrent Networks

The previous section showed that vanilla RNNs have difficulty learning long-range dependencies because both forward information and backward learning signals must pass through repeated recurrent transformations. Gated recurrent networks address this issue by changing how the hidden state is updated. Instead of replacing the old state with a newly computed state, a gated model can choose to preserve part of the old state and update only part of it. This creates a direct path through time.

A useful abstract form of a gated recurrent update is

$$s_t = g_t \odot s_{t-1} + (1 - g_t) \odot \tilde{s}_t, \tag{2.8}$$

where s_t is a state vector, $\tilde{s}_t$ is a candidate update, $g_t \in (0,1)^{d_s}$ is a gate vector, and $\odot$ denotes element-wise multiplication. When an entry of g_t is close to one, the corresponding state dimension is copied from s_{t-1} to s_t. When it is close to zero, the old value is replaced by the candidate value. Thus, Equation 2.8 can behave either like memory or like update, depending on the gate value.

This form is important for gradient propagation. If the state is copied through an additive path, then the backward signal does not need to pass only through a full nonlinear recurrent transformation at every step. In the idealized case where g_t is fixed and close to one, the derivative of s_t with respect to s_{t-1} contains a near-identity component. This provides a path along which information and gradients can be preserved more easily than in a vanilla RNN.

Figure 2.3 illustrates the role of the additive copy path in a gated recurrent update. Compared with the vanilla recurrent chain, the gate introduces an additional route between consecutive states. This route can carry information forward when the gate favors copying, and it can carry gradients backward through a more direct path. GRUs and LSTMs implement this general idea in slightly different ways.

Gated recurrent units. A gated recurrent unit (GRU) implements this idea with two gates: an update gate and a reset gate [4, 5]. In this thesis, we use the convention that a larger update gate means more

copying of the previous hidden state. Given the current input x_t and the previous hidden state h_{t-1}, the GRU computes

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z), \tag{2.9}$$
$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r), \tag{2.10}$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h), \tag{2.11}$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t. \tag{2.12}$$

The reset gate r_t controls how much of the previous hidden state is used when computing the candidate state $\tilde{h}_t$. If r_t is close to zero, the candidate is computed mostly from the current input, allowing the unit to ignore previous state information. The update gate z_t then controls how much of the previous hidden state is preserved in the final state h_t. If z_t is close to one, the hidden state is mostly copied forward. If z_t is close to zero, the hidden state is mostly replaced by the candidate state.

Equation 2.12 is the main reason why GRUs can help with long-range dependencies. The term $z_t \odot h_{t-1}$ creates a direct additive path from h_{t-1} to h_t. If this path remains open over many time steps, relevant information can be carried forward without being repeatedly overwritten by noisy or irrelevant inputs. In the backward pass, the same path allows gradients to move through time without being forced through a full nonlinear transformation at every step. This makes the GRU update gate directly relevant to the write-or-copy behavior studied in this thesis.

Long short-term memory networks. Long short-term memory (LSTM) networks introduce a separate cell state c_t in addition to the hidden state h_t [14]. Modern LSTM variants are commonly described with an input gate, a forget gate, and an output gate. In the formulation used here, without peephole connections, the gates and candidate cell update are

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i), \tag{2.13}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f), \tag{2.14}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o), \tag{2.15}$$
$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c). \tag{2.16}$$

The cell state and hidden state are then updated by

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{2.17}$$
$$h_t = o_t \odot \tanh(c_t). \tag{2.18}$$

The forget gate f_t controls how much of the previous cell state is retained. The input gate i_t controls how much new candidate content is written into the cell. The output gate o_t controls how much of the cell state is exposed as the hidden state. Compared with a vanilla RNN, the LSTM separates memory storage in c_t from the exposed hidden representation h_t.

The key operation is the additive cell update in Equation 2.17. If f_t is close to one and i_t is close to zero, then c_t is approximately equal to c_{t-1}, so the cell state is copied forward. If f_t is close to zero, the old memory is erased. If i_t is large, new information is written into the cell. In this way, the LSTM can learn when to keep old information, when to forget it, and when to write new information. The direct path through the cell state is often described as the mechanism that helps LSTMs preserve long-range information and mitigate vanishing gradients.

The cell-state path also clarifies the role of the forget gate. Along the direct path from c_{t-1} to c_t, the derivative contains the factor f_t when other dependencies are held fixed. If f_t remains close to one, gradients can propagate along the cell state with little attenuation. If f_t is consistently much smaller than one, this path can still decay over time. This is why practical LSTM implementations often initialize the forget-gate bias positively, encouraging the model to retain information at the beginning of training [16].

What gates do and what they do not explain. GRUs and LSTMs therefore mitigate the gradient problem by changing the geometry of the recurrent update. Vanilla RNNs repeatedly transform the entire hidden state through a nonlinear map. Gated recurrent networks instead include additive copy

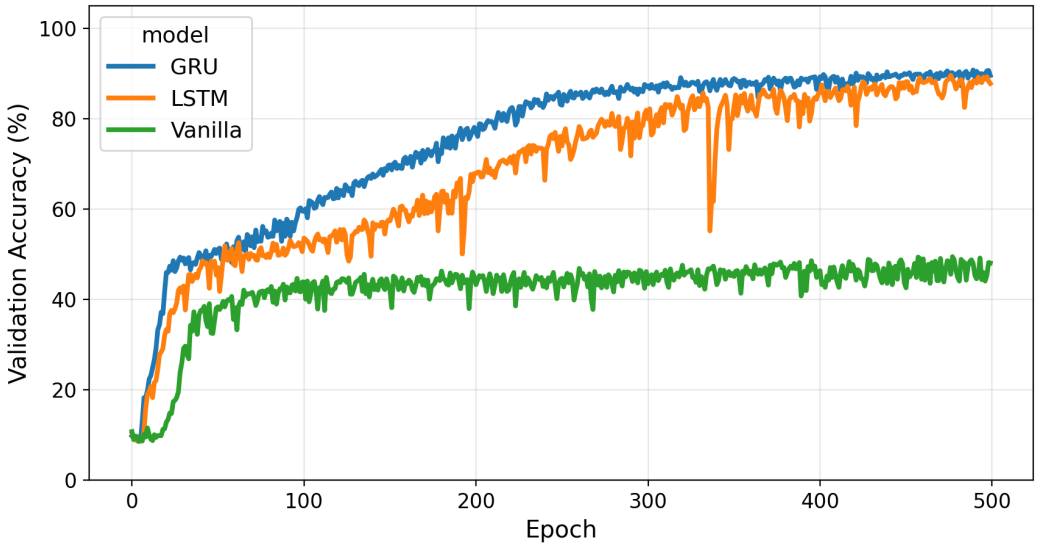


Figure 2.4: Implementation sanity check on MNIST-1D. The figure shows validation accuracy over training epochs for the vanilla RNN, GRU, and LSTM implementations used in this thesis. MNIST-1D is used as the external sanity check dataset. The GRU and LSTM reach roughly the accuracy range reported for GRU models on MNIST-1D in the original benchmark, while the vanilla RNN remains much lower. This supports the interpretation that the later failures of recurrent baselines in controlled long-range tasks are not caused simply by broken implementations.

paths. These paths make it possible to preserve selected parts of the state and to propagate gradients through time more easily. This explains why suitable gate values can help long-range learning.

However, this explanation is incomplete for the purpose of this thesis. The gate values are not given in advance. In both GRUs and LSTMs, gates are computed from the current input and the previous hidden state, using learned parameters. Therefore, the model must learn not only what information to store, but also when to open or close the gates. If the only useful learning signal comes from a distant final loss, then learning the gate policy may itself require long-range credit assignment. This is the homunculus-like issue studied in this thesis: gated recurrent networks provide useful direct paths through time, but the model still has to learn when those paths should be used.

2.4. Controlled Sequential Data

Before using controlled tasks to study a mechanism, it is important to check that the compared recurrent models are trainable and implemented correctly. If a baseline fails on the main experiments, the failure should reflect the hypothesized limitation of the architecture rather than a coding error, a broken implementation, or an obviously unsuitable training setup. We therefore use an external sequence benchmark as an implementation sanity check before moving to the controlled experiments.

The sanity check uses MNIST-1D, a procedural one-dimensional version of MNIST designed for low-compute experiments on architectural inductive biases [11]. Each example is a short one-dimensional signal, and the task is to classify it into one of ten digit classes. MNIST-1D is not the main task of this thesis, but it is useful here because it is independent from the synthetic tasks used in Part 3. The original MNIST-1D paper reports that a GRU reaches about 91% test accuracy on this benchmark [11]. As shown in Figure 2.4, the recurrent implementations used in this thesis show the expected behavior: the GRU and LSTM learn the task and reach approximately the same accuracy range, while the vanilla RNN remains substantially lower. This result does not prove that the implementations are correct in every detail, but it gives evidence that the recurrent baselines are capable of learning a nontrivial sequence classification task and that the gated models behave consistently with an external benchmark.

MNIST-1D is useful for validation, but it is not sufficient for answering the research question of this thesis. If a gated model performs better than a vanilla RNN on MNIST-1D, the result does not reveal why. The task contains several possible sources of difficulty at the same time, including local shape

recognition, positional variation, noise, class-specific patterns, and optimization effects. A performance difference on such a benchmark can show that one architecture works better, but it does not isolate whether the benefit comes from learning to preserve information, learning to ignore irrelevant inputs, receiving stronger gradients, or exploiting some other structure in the data.

The main experiments therefore use controlled sequential data. The purpose of controlled data is to isolate the mechanism under study. In the controlled tasks used in Part 3, only a small number of tokens are informative, while the remaining tokens are deliberately non-informative. The sequence length controls how far information must be carried through the recurrent state. Easier training examples can shorten this dependency while preserving the same underlying rule. This makes it possible to ask whether gated models first learn a simpler selective update behavior and then reuse that behavior on harder long-range examples.

This controlled setting is essential for the homunculus-like question studied in this thesis. The MNIST-1D sanity check validates that the recurrent implementations can learn a standard external sequence task. The controlled tasks then investigate a more specific mechanism: when gates help, do they help because the model has learned a reusable rule for retaining informative inputs and ignoring non-informative ones? The next part of the thesis introduces these tasks formally and uses them to test the learning behavior of vanilla RNNs, GRUs, and LSTMs.

3

Scientific Article

THE RNN HOMUNCULUS: ON LEARNING RNN GATES WITH RNNS

ABSTRACT

Gated recurrent neural networks are commonly explained by their ability to create additive copy paths through time, which can preserve information and gradients over long sequences. This explanation is correct, but incomplete: useful gate values must themselves be learned, and this gate-learning process is also performed through recurrent computation. We study this missing learning step with controlled sequence classification tasks. We show that gated architectures do not solve long-range dependencies by architecture alone: when all training samples require long-range memory from the start, gated and non-gated recurrent models both fail. However, when training also contains short-dependency samples in which the same label relation can be learned over shorter temporal gaps, gated models can first learn a selective update behavior and then apply it to long-range samples. Diagnostic probes show that during successful training, larger gradients reach early recurrent states, and state updates depend more clearly on the input. A multi-class extension further shows that the learned behavior transfers partially beyond the subset of informative inputs that receives short-dependency samples. Overall, our results suggest that gates help not because they automatically solve long-range dependencies, but because they provide a mechanism that can be learned once the data makes the gate-learning problem simple enough. This reframes gated recurrence from an automatic solution to a learnable scaffold, and suggests that training data should be designed to expose gate-learning signals before relying on gates for long-range memory.

1 INTRODUCTION

Long-range sequence modeling is of great significance in a wide variety of applications such as Large Language Models (LLMs), Chatbots, and Agentic AI (Brown et al., 2020; Wang et al., 2024). However, modern sequence models face a tension between long-range reasoning and inference cost. Self-Attention has become the dominant mechanism for modeling long sequences, but its inference cost grows quadratically with sequence length (Vaswani et al., 2017). Recurrent Neural Networks (RNNs) may offer a complementary trade-off: they update fixed-size states recurrently, leading to linear-time sequence processing during inference (Elman, 1990; Williams & Zipser, 1989). The obstacle is that vanilla RNNs struggle to learn long-range dependencies. Backpropagation through time multiplies many Jacobians, which can cause the gradients to vanish as the sequence length grows (Bengio et al., 1994; Hochreiter, 1998; Pascanu et al., 2013). Understanding when and how recurrent models can overcome the gradient problem and long-range credit assignment remains relevant because state-space architectures continue to be attractive for efficient long-range modeling (Gu et al., 2022; Gu & Dao, 2023).

Gated recurrent networks, such as Long Short-Term Memory (LSTMs) and Gated Recurrent Unit (GRUs), are the seminal answer to the gradient problem in RNNs (Cho et al., 2014; Chung et al., 2014; Hochreiter & Schmidhuber, 1997). Their hidden state updates contain additive paths controlled by gates, which can create near-identity routes for information and gradients. For example, an LSTM cell can preserve information when the forget gate remains close to one, and a GRU can preserve information when its update gate keeps the previous hidden state (Noh, 2021). This mathematical explanation is correct, but incomplete. There is a *homunculus* in it. A homunculus explanation explains a difficult behavior by placing a smaller agent or mechanism inside the system that performs the same behavior. Such an explanation becomes problematic when the internal mechanism requires the same kind of explanation again, producing a regress rather than resolving

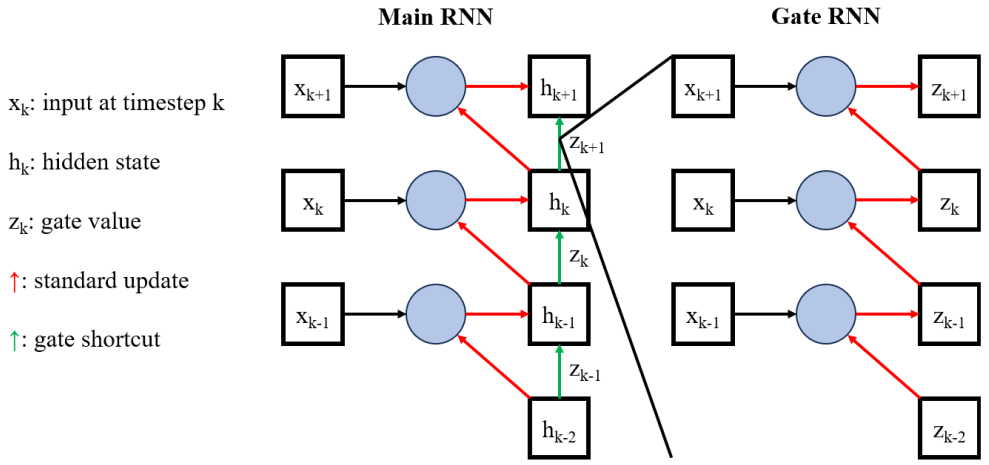


Figure 1: **The homunculus problem in gated RNNs.** The left side shows a gated RNN: the standard recurrent computation, shown in red, updates the hidden state h_k with the previous hidden state and the current input x_k. This path forms the long chain through which gradients can vanish. Gate values z_k create direct paths, shown in green, directly from h_{k-1} to h_k. The zoom-in on the right shows the missing step in this explanation: the gate values z_k are themselves computed from inputs x_k and previous states by a recurrent process, a "gate RNN" with identical structure to the main gated RNN.

the original difficulty (Kenny, 1971; Ryle, 2000; Dennett, 1991). The standard explanation of gated RNNs assumes useful gate values, but the gates are themselves learned functions of the current input and previous hidden state. Therefore, the gate values that protect long-range gradients must also be learned through the same recurrent computation that suffers from long-range gradient decay. Figure 1 illustrates this regress of recurrent structures.

This paper studies the missing learning problem of gates. The central question is not whether certain gate configurations can create good gradient paths, but how gated RNNs learn these configurations in the first place. We hypothesize that gates do not inherently solve the gradient problem and long-range dependencies. Instead, gated RNNs succeed when the data contains a short-range sub-task that can be learned before the full long-range task is solved. In the settings studied in this paper, this simpler sub-task is to identify which inputs should be remembered and which inputs should be ignored. Once the gates have learned this selective-update rule, the main recurrent state can use the induced memory path to solve harder long-range tasks.

We test the hypotheses with controlled sequence classification tasks. We synthesized data where in each sample, two informative tokens are placed in a sequence of non-informative tokens. The binary label depends on the relation between the two informative tokens. When the informative tokens appear only at long range, vanilla RNNs, GRUs, and LSTMs all fail as the sequence length increases. When a small fraction of easy, short-dependency samples, where the informative tokens are placed near the end of the sequence, is added to the training set, gated RNNs learn to solve the original long-range task, while vanilla RNNs still fail. We further extend the task to a multi-class setting, where short-dependency samples are provided for only one group of informative tokens. The improvement transfers partly to another token group, suggesting that the benefit is not merely caused by memorizing the informative tokens, but by learning a reusable gating policy within the observed token distribution.

Our contributions can be summarized as follows:

- We identify a missing step in the standard explanation of gated RNNs: gates can create good gradient paths only after useful gate values have been learned, but gate learning itself is also subject to recurrent gradient decay.
- We introduce controlled experiments that demonstrate when gated RNNs fail and when they succeed. The experiments show that a small fraction of short-dependency samples can enable gates to learn a selective-update rule that transfers to harder long-range samples.

- We provide diagnostic evidence for the learned mechanism and test whether it transfers beyond the examples that make it easy to learn.

2 Related Work

2.1 The Vanishing Gradient Problem

Vanilla RNNs can represent temporal information through recurrent hidden states, but learning long-range dependencies with gradient descent is difficult. The classical analysis shows that the problem is not only representational, but also about optimization: information from an early input must influence a later loss through many recurrent transitions (Bengio et al., 1994; Hochreiter, 1998; Pascanu et al., 2013).

For a recurrent state update, let $h_t \in \mathbb{R}^{d_h}$ denote the hidden state at time step t, let x_t denote the input at time step t, and let θ denote the recurrent parameters. The hidden state is computed by a differentiable transition function F as $h_t = F(h_{t-1}, x_t; \theta)$. For two time steps $k < t$, the sensitivity of the later hidden state h_t to the earlier hidden state h_k is obtained by applying the chain rule through all intermediate recurrent transitions:

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}. \tag{1}$$

Here, each factor $\partial h_i / \partial h_{i-1} \in \mathbb{R}^{d_h \times d_h}$ is the recurrent Jacobian at time step i, and the product is ordered from $i = k + 1$ to $i = t$ along the unrolled computation graph. This product is the central quantity in the vanishing and exploding gradient problem. If the recurrent Jacobians contract the error signal on average, then the norm of the product in Equation 1 decays exponentially with the temporal distance $t - k$; if they expand it, the same product can grow exponentially (Pascanu et al., 2013). As a result, long-range temporal contributions to the gradient can become negligible compared with short-range contributions, making gradient descent biased toward short-term dependencies (Bengio et al., 1994; Hochreiter, 1998).

This classical analysis explains why vanilla RNNs often fail on tasks where the target depends on inputs far in the past. It also motivates gated architectures: if a model can create an additive path whose Jacobian is close to the identity, then information and gradients can in principle travel over longer temporal gaps.

2.2 Gated Recurrent Networks

A major architectural response to the gradient problem is to replace the fully overwritten hidden state of vanilla RNNs with gated state updates. The Long Short-Term Memory network (LSTM) was introduced to maintain constant error flow through memory cells, with multiplicative gates controlling access to that flow (Hochreiter & Schmidhuber, 1997). The Gated Recurrent Unit (GRU) later simplified this idea by using reset and update gates to control how much each unit remembers and forgets (Cho et al., 2014). Empirical comparisons show that gated units usually outperform traditional recurrent units on sequence modeling tasks, although neither architecture uniformly dominates the other (Chung et al., 2014; Jozefowicz et al., 2015).

We can describe the common gating mechanism without committing to a specific LSTM or GRU. Let $h_t \in \mathbb{R}^{d_h}$ denote the recurrent state at time step t, let x_t denote the input, and let $\tilde{h}_t \in \mathbb{R}^{d_h}$ denote a candidate state computed from x_t and the previous state h_{t-1}. A generic gated recurrent update interpolates between keeping the old state and writing a new state:

$$h_t = g_t \odot h_{t-1} + (1 - g_t) \odot \tilde{h}_t, \tag{2}$$

where $g_t \in [0, 1]^{d_h}$ is a learned gate vector and $\odot$ denotes element-wise multiplication. When an element of g_t is close to one, the corresponding dimension of the previous state is copied forward; when it is close to zero, that dimension is overwritten by the candidate state. Thus, the gates make data-dependent decisions about what to keep and what to update.

This additive update changes the gradient path through time. Equation 2 contains a direct state-copying term with local derivative

$$\frac{\partial h_t}{\partial h_{t-1}} \supset \mathrm{diag}(g_t), \tag{3}$$

where $\mathrm{diag}(g_t)$ is the diagonal matrix whose diagonal entries are the elements of g_t. $\supset$ indicates that $\mathrm{diag}(g_t)$ is the direct copy-path contribution to the full recurrent Jacobian, not necessarily the full Jacobian itself. If these diagonal entries of g_t remain close to one over many time steps, this direct path can transmit information and gradients without repeatedly passing them through full recurrent matrices and bounded nonlinearities. This is the reason gated recurrent networks are viewed as mitigating the vanishing gradient problem.

2.3 What Gated Recurrent Networks Learn

A complementary line of work studies what gated recurrent networks learn after training, especially whether their recurrent states encode structured long-range dependencies. In subject–verb agreement, LSTMs perform well under targeted supervision, but become less reliable when linear cues conflict with syntactic structure (Linzen et al., 2016). LSTM language models can also predict long-distance agreement across multiple languages and even in nonce sentences, suggesting that they learn more than lexical or semantic shortcuts (Gulordava et al., 2018). Similar probing methods show that LSTMs can track filler–gap dependencies across positions and constructions (Kobzeva et al., 2020). These results suggest that gated RNNs can learn useful long-range structure, but they do not explain when such structure is learnable.

Mechanistic studies provide evidence that trained recurrent networks can implement simple internal tracking mechanisms. Character-level LSTMs contain interpretable cells that track long-range properties such as quotes, brackets, and line length (Karpathy et al., 2016). In number agreement, a small number of LSTM units can carry long-range number information, while other units track syntactic structure and regulate when this information is stored or released (Lakretz et al., 2019).

The standard mathematical analyses focus on gradient paths created after gates take useful values, but not on the long-chain gradient problem faced by the gate functions themselves. The empirical studies show what useful gated mechanisms can look like after training, but not how they are learned. Our work studies the missing steps: we analyze why gate learning can also suffer from long-range gradient problem, and use controlled experiments to test when and how gates can first acquire the write-or-copy behavior: writing task-relevant inputs into the recurrent state while copying the previous state across non-informative inputs, observed in trained recurrent networks.

3 From Gate Gradients to Controlled Questions

3.1 The Gate Learning Problem

Here we derive mathematically the gradient problem faced by gates themselves.

Consider a GRU-style gated update (Cho et al., 2014),

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t, \tag{4}$$

where $h_t \in \mathbb{R}^{d_h}$ is the hidden state at time step t, $\tilde{h}_t \in \mathbb{R}^{d_h}$ is a candidate state, $z_t \in [0,1]^{d_h}$ is the update gate, and $\odot$ denotes element-wise multiplication. With this convention, a gate value close to one copies the previous state, while a value close to zero writes the candidate state.

Let L be a loss that depends on the final hidden state h_T, and let $\delta_t = \partial L / \partial h_t$ denote the gradient arriving at hidden state h_t. Differentiating Equation 4 with respect to the gate activation gives

$$\frac{\partial L}{\partial z_t} = \delta_t \odot (h_{t-1} - \tilde{h}_t). \tag{5}$$

This gradient tells the model whether increasing the copy path or the write path would reduce the loss at time step t. However, the strength of this signal depends directly on δ_t.

For a final-step loss, δ_t is obtained by backpropagating through all future recurrent transitions:

$$\delta_t = J_{t+1}^\top J_{t+2}^\top \cdots J_T^\top \delta_T, \qquad J_i = \frac{\partial h_i}{\partial h_{i-1}}. \tag{6}$$

Thus, the learning signal for an early gate also contains a long product of recurrent Jacobians. If this product contracts, then δ_t becomes small, and Equation 5 shows that the gate activation also becomes small.

Finally, the gate activation is produced by a learned function,

$$z_t = \sigma(a_t^z), \qquad a_t^z = W_z x_t + U_z h_{t-1} + b_z, \tag{7}$$

where x_t is the input, W_z, U_z, and b_z are gate parameters, and σ is the logistic sigmoid. The parameter gradients are therefore proportional to

$$\frac{\partial L}{\partial a_t^z} = \frac{\partial L}{\partial z_t} \odot z_t(1 - z_t). \tag{8}$$

Combining Equations 5–8, the gate parameters receive a useful update only when the loss signal reaches h_t and when the sigmoid is not saturated.

The same issue appears in LSTMs. For the cell update (Hochreiter & Schmidhuber, 1997)

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t,$$

the forget gate receives

$$\frac{\partial L}{\partial f_t} = \frac{\partial L}{\partial c_t} \odot c_{t-1}.$$

Thus, although a forget gate close to one can preserve cell-state gradients, the signal that teaches the forget gate to stay close to one still depends on the gradient arriving at c_t.

3.2 Do Gated RNNs Also Fail at Long-Range?

The derivation above shows that, before a useful gating policy has been learned, the gradient reaching an early gate can be attenuated by the long product of future recurrent Jacobians. Our first question is therefore whether gated RNNs can solve a long-range dependency task when the training data provides no short-range gate-learning signal.

To test this, we use a controlled temporal XOR task. Each input is an integer sequence $x_{0:T-1}$ with vocabulary size V. Two tokens in the sequence are informative, drawn from $\{0, 1\}$, and all other tokens are non-informative, drawn from $\{2, \ldots, V-1\}$.

In this setting, the informative tokens are always placed at the first and last positions:

$$x_0 = a, \qquad x_{T-1} = b, \qquad a, b \in \{0, 1\}. \tag{9}$$

All intermediate positions $x_1, \ldots, x_{T-2}$ are non-informative tokens.

The label is the XOR of the two informative tokens:

$$y = a \oplus b = \mathbb{I}[a \neq b]. \tag{10}$$

For each sequence length T, we generate a separate training set and test set where all sequences have the same length T and the same first–last dependency structure. We train vanilla RNNs, GRUs, and LSTMs independently on each dataset, and evaluate test accuracy. This setting contains no short-range samples: to make a correct classification, information from the first token must be carried to the last hidden state through the recurrent chain.

If gates solve long-range dependencies by architecture alone, GRUs and LSTMs should maintain accuracy as T increases. If not, then gated RNNs should eventually fail together with vanilla RNNs. Figure 2 shows examples of sequences in the temporal XOR setting.

3.3 When and How Do Gates Learn?

The second question is whether gated RNNs succeed once the gate-learning problem is made easier. Section 3.1 suggests that the gate parameters receive useful updates only when the loss signal reaches the relevant gate activations. Shorter dependencies could provide such signals earlier in training.

We therefore introduce a mixed temporal XOR setting that keeps the test task unchanged but modifies the training distribution. The long-dependency samples are identical to the previous setting: the two

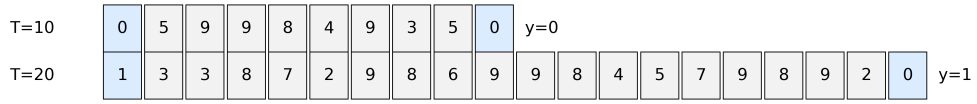


Figure 2: **Examples from the temporal XOR task.** The highlighted first and last tokens are the only informative tokens and determine the binary XOR label. All intermediate tokens are non-informative. This setting forces the model to connect information across the full sequence length.

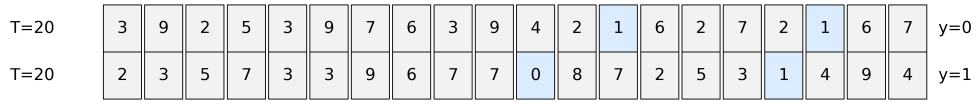


Figure 3: **Examples from the short-dependency temporal XOR training samples.** The highlighted tokens determine the XOR label, but they are placed near the end instead of at the boundaries. These samples provide short-range cases from which gates can learn which tokens to remember or ignore.

informative tokens occur at positions 0 and $T-1$. We also refer to these as *hard samples*, because solving them requires carrying information across the full recurrent chain. For a fraction p_{easy} of the training samples, however, the two informative tokens are placed inside a short suffix window near the end of the sequence. We refer to these as *short-dependency samples*. These samples have the same label rule and the same token vocabulary as the hard samples, but the temporal distance from the informative tokens to the final loss is shorter. At test time, we evaluate only on hard samples with informative tokens at the sequence boundaries.

Our prediction is that gated RNNs can use these short-dependency samples to learn a write-or-copy policy, and that this policy then transfers to the original long-range samples. A vanilla RNN has no explicit gate-controlled copy path, so the same short-dependency samples should be less effective at helping preserve information across long temporal gaps. Figure 3 shows examples of the short-dependency samples used in the mixed training setting.

3.4 Does the Learned Gating Policy Transfer Beyond Specific Labels?

The mixed temporal XOR experiment tests whether short-dependency samples help gates solve harder samples of the same binary task. Our third question is whether a gate-learning benefit can transfer across informative token groups.

We construct a multi-class XOR task with disjoint informative token groups. The first group uses tokens $\{0,1\}$ and the second group uses tokens $\{2,3\}$. Non-informative tokens are sampled from $\{4,\dots,V-1\}$. The task now has four balanced classes:

$$(0,1),(1,0)\mapsto 0, \qquad (0,0),(1,1)\mapsto 1, \tag{11}$$

$$(2,3),(3,2)\mapsto 2, \qquad (2,2),(3,3)\mapsto 3. \tag{12}$$

To test transfer, short-dependency samples are added only for one informative group, for example the $\{2,3\}$ group. Samples from the $\{0,1\}$ group remain hard-only throughout training. If the model only memorizes the treated labels and corresponding informative tokens, improvement should be restricted to the group that receives short-dependency samples. If the model learns a more reusable gating policy, such as ignoring non-informative tokens, then some benefit should transfer to the hard-only group. Figure 4 shows examples of sequences in the multi-class XOR setting.

4 Experiments

4.1 Experimental Setup

In the controlled experiments, we evaluate three recurrent architectures: vanilla RNNs, GRUs, and LSTMs. All models receive sequences of discrete tokens. Each token is first mapped to a learnable embedding, then processed by the recurrent model, and the final hidden state is used for classification.

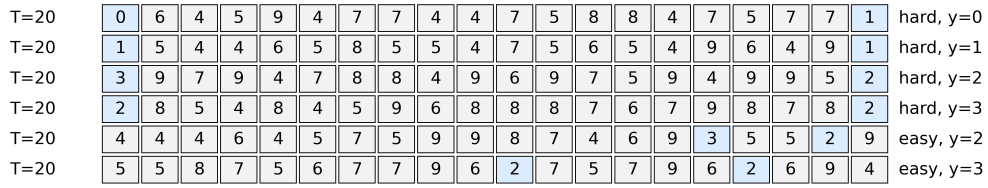


Figure 4: **Examples from the multi-class XOR setting.** The first four rows show hard samples for all four classes. The last two rows show short-dependency samples added only for the $\{2, 3\}$ group.

The last state classifier setting forces the model to preserve task relevant information through the recurrent updates rather than reading it directly from intermediate hidden states.

The binary temporal XOR experiments use vocabulary size $V = 10$. The informative tokens are $\{0, 1\}$ and the non-informative tokens are sampled from $\{2, \ldots, 9\}$. The multi-class XOR experiments use vocabulary size $V = 12$. The informative tokens are $\{0, 1, 2, 3\}$ and the non-informative tokens are sampled from $\{4, \ldots, 11\}$. Thus, both settings contain eight non-informative token types. For each sequence length T, we generate separate training, validation, and test sets, train each model from scratch, select a checkpoint using the validation set, and report test accuracy on the corresponding test set. For the mixed setting, only the training distribution is modified by adding short-dependency samples, the validation and test distribution always contain hard samples only, where the informative tokens are placed at the first and last positions.

We repeat each experiment over multiple random seeds and report the mean and standard deviation of the selected checkpoint test accuracy. For the multi-class XOR experiments, we also report group-wise accuracies for the $\{0, 1\}$ and $\{2, 3\}$ informative-token groups.

Implementation details. We use a token embedding dimension of 8. In hard-only experiments, all models use hidden size 10. In mixed-training experiments, GRUs and LSTMs still use hidden size 10, while the vanilla RNN uses hidden size 24. This gives the vanilla RNN a larger recurrent state and a comparable number of parameters, making the comparison conservative: if the vanilla RNN still underperforms, the advantage of gated models is unlikely to be explained only by parameter count. All models are trained with cross-entropy loss and optimizer AdamW, using learning rate 5×10^{-3} for binary temporal XOR and 10^{-2} for multi-class, weight decay 10^{-4}, and betas $(0.9, 0.95)$. The batch size is 500. Models are trained for up to 500 epochs. During training, we keep the checkpoint with the best validation accuracy, using validation loss as a tie-breaker, and evaluate this selected checkpoint on the test set. Training is stopped early only if validation accuracy reaches 100% for 20 consecutive epochs.

The binary temporal XOR setting uses $2{,}000$ training samples, 400 validation samples, and 400 test samples. The multi-class XOR setting uses $4{,}000$ training samples, 800 validation samples, and 800 test samples. Samples are generated uniquely across the training, validation, and test splits, so that test accuracy does not depend on exact sequence memorization. Unless stated otherwise, the proportion of short-dependency samples in mixed training is $p_{\text{easy}} = 40\%$. For asymmetric multi-class transfer experiments, this proportion is applied only within the specified informative-token group, corresponding to an expected short-dependency sample proportion of 20% over the full training set. For each model and sequence length, we run all combinations of five model-initialization seeds $[11, 22, 33, 44, 55]$ and five data-generation seeds $[11, 22, 33, 44, 55]$, resulting in 25 runs.

4.2 Exp 1: Gated RNNs Also Fail

The first experiment tests whether gated RNNs solve long-range dependencies by architecture alone.

Figure 5 shows that gated RNNs do not automatically solve the hard temporal XOR task. For short sequences, all models can learn the task. As the sequence length increases, the vanilla RNN quickly collapses to chance-level accuracy. The GRU and LSTM are more robust at intermediate lengths, especially the LSTM, but they also eventually fail as T grows.

This result supports our derived gate-learning problem. Gated architectures provide a possible copy path for information and gradients, but this path is only useful once the model has learned when to

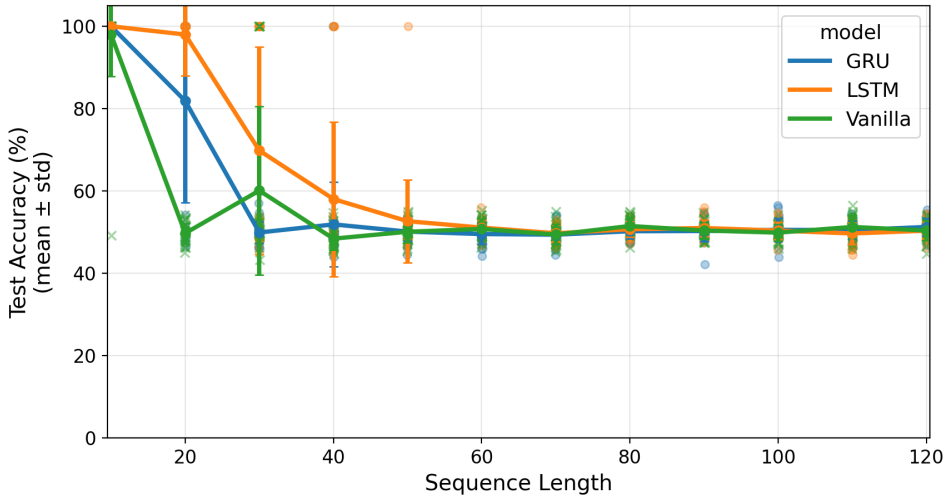


Figure 5: **Hard-only training on binary temporal XOR.** For every sequence, the two informative tokens are fixed at the first and last positions, so the model must preserve the first token across the full sequence before comparing it with the last token. Lines show mean test accuracy, error bars show standard deviation, and faint markers show individual runs. Vanilla RNNs collapse to chance accuracy at short sequence lengths, while GRUs and LSTMs remain accurate for longer but also collapse as the sequence length increases. Thus, gated RNNs delay the long-range failure, but do not solve the long-range temporal XOR task by architecture alone.

open and close the gates. In the hard-only setting, there are no short-range samples that make this gate-learning problem easier. The failure of GRUs and LSTMs at long sequence lengths therefore suggests that gating alone is not sufficient: the useful gating policy itself must be learned.

4.3 Exp 2: Short-Dependency Samples Teach Selective Memory

The second experiment tests whether gated RNNs succeed when the gate-learning problem is easier.

Figure 6 shows different behaviors from the hard-only setting. With the same test distribution, GRUs and LSTMs remain accurate over much longer sequences once short-dependency samples are added to training. In contrast, the vanilla RNN still drops to near chance-level accuracy for long sequences.

This result supports the hypothesis that short-dependency samples help gated RNNs learn when to write and when to copy. The short-dependency samples contain the same token identities and label rule as the hard samples, but the relevant tokens are closer to the classification unit which backpropagates the loss gradients. They therefore provide a stronger learning signal for the gates. Once the gates learn to preserve informative tokens and suppress non-informative tokens, the learned policy can be applied to hard samples where the temporal distance is longer.

The contrast between Figures 5 and 6 is the key controlled comparison: the test task and gated architectures stay the same, but the training distribution changes. In the hard-only setting, gated RNNs eventually fail together with vanilla RNNs. In the mixed setting, the same gated architectures succeed on the original hard test task. The intervention is not a new architecture, but a change in the learnability of the gating policy.

To inspect what changes during successful mixed training, we analyze a pair of GRU runs, a failed hard-only setting and a successful mixed training setting, at sequence length $T = 100$. We visualize the model's learning process and probe internal behaviors.

Figure 7 shows a two-stage learning process in the successful mixed run. In the hard-only setting, training accuracy gradually increases, but validation accuracy stays near chance. This gap suggests that the model is over-fitting, it memorizes parts of the training distribution without learning a rule

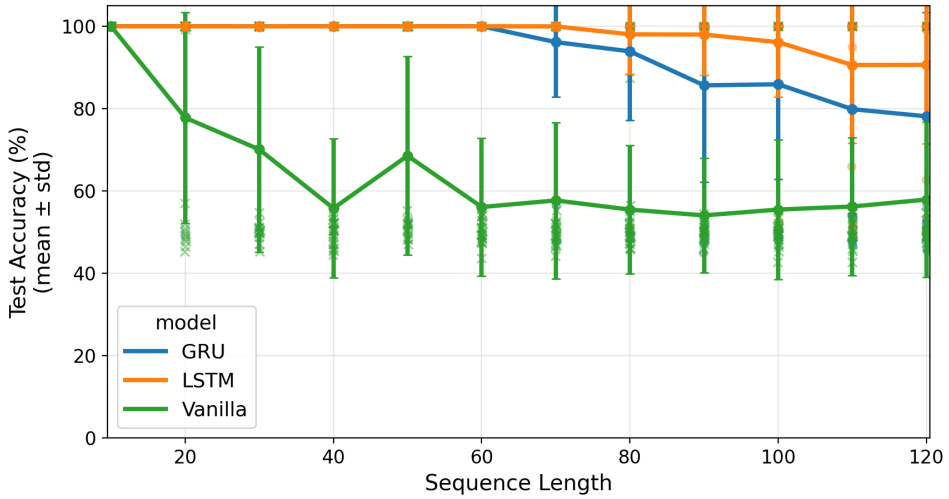


Figure 6: **Mixed training on binary temporal XOR.** All validation and test sequences are hard samples, with the two informative tokens fixed at the first and last positions. During training, 40% of the samples are short-dependency samples, where the informative tokens follow the same XOR label rule but appear inside a short suffix window near the end of the sequence. Lines show mean test accuracy, error bars show standard deviation, and faint markers show individual runs. Compared with Figure 5, GRUs and LSTMs maintain high test accuracy over much longer sequences, while vanilla RNNs remain close to chance for long sequences. Thus, short-dependency samples help gated RNNs solve the hard long-range task, suggesting that they make the gating policy easier to learn.

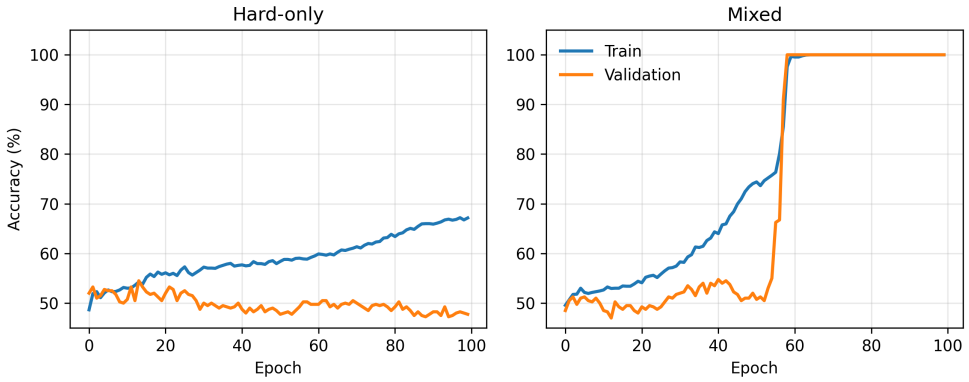


Figure 7: **Training dynamics of one representative GRU run.** The left panel shows hard-only training. The right panel shows mixed training. Both panels show training accuracy curves; the validation curve is always measured on hard samples. In the hard-only setting, training accuracy improves while validation accuracy remains near chance, which is consistent with overfitting or memorization rather than learning a reusable long-range rule. In the mixed setting, training accuracy and hard-validation accuracy rise together, indicating that the rule learned from short-dependency samples transfers to hard samples.

that generalizes to long range. In the mixed setting, the early increase in training accuracy does not immediately improve hard-validation accuracy. This first phase is consistent with the model learning the short-dependency samples, which share the same label rule but do not yet require long-range memory. Around the transition point, training accuracy rises from the intermediate range to nearly perfect accuracy, and validation accuracy rises at the same time. This second phase suggests that the information learned from short-dependency samples has become useful for hard samples as well.

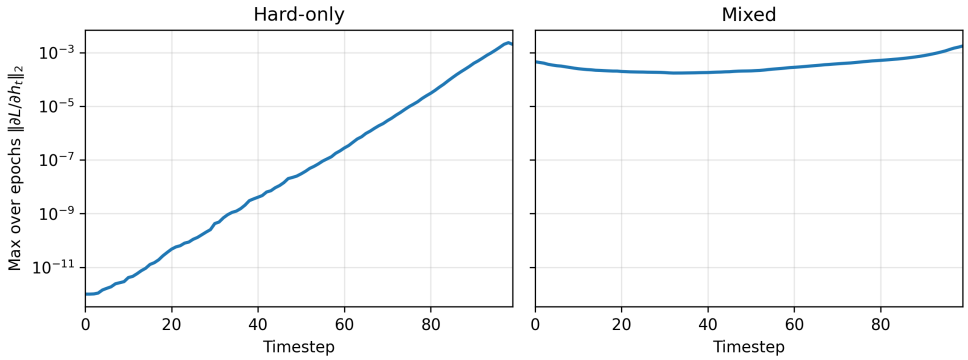


Figure 8: **Maximum gradient received at each timestep during one representative GRU run.** A recurrent timestep refers to a position t in the input sequence and in the unrolled recurrent chain. For every timestep t, the figure reports the maximum value of $\|\partial L/\partial h_t\|_2$ observed over training epochs. The y-axis is logarithmic. In the hard-only setting, the largest gradient signal decays by many orders of magnitude from the final timestep to the first timestep, so early hidden states receive almost no useful learning signal. In the mixed setting, large gradients reach all timesteps at some point during training. Thus, short-dependency samples change the optimization problem: they allow learning signals to reach early recurrent states.

Figure 8 connects this difference in learning behavior to the availability of gradient signals along the recurrent chain. For each timestep, we plot the largest hidden-state gradient norm that reaches that timestep at any point during training. In the hard-only setting, the maximum gradient decreases exponentially as the timestep moves away from the final loss. The early hidden states therefore receive almost no learning signal. This explains why the hard-only run can improve training accuracy without learning a rule that generalizes to the validation samples: the early recurrent states are difficult to update from the final loss. The mixed setting shows a different gradient profile. Large gradients reach not only the final timesteps but also the early timesteps. This observation does not yet show what the GRU has learned internally. It shows a more basic prerequisite: mixed training makes early recurrent computations reachable by useful learning signals. The short-dependency samples provide shorter dependency paths during training, and these paths can create gradient signals before the full first-to-last dependency has been solved. We next inspect whether the GRU uses this improved learning signal to develop token-dependent gate and state-update behavior.

Figure 9 asks whether the improved gradient availability in mixed training is accompanied by a change in the gate behavior itself. The hard-only run shows no useful separation between token types: the update gate values do not develop a clear distinction between informative and non-informative inputs. This is consistent with the earlier observation that the hard-only run improves training accuracy without discovering a rule that transfers to validation. The mixed run shows a different pattern. The update gate moves in opposite directions for different token types: it increases for non-informative tokens and decreases for the two informative tokens: token 0 and token 1. Under the GRU update convention used here, this means that the model increasingly copies the previous hidden state across non-informative tokens, while writing new candidate information when an informative token appears. The gate probe therefore gives internal evidence for a token-dependent write-or-copy behavior. The next probe checks whether this gate-level separation is also reflected in the actual hidden-state updates.

Figure 10 checks whether the gate-level separation in Figure 9 is reflected in the recurrent state dynamics. The hard-only run does not develop selective state updates: non-informative tokens continue to perturb the hidden state throughout training. In the mixed run, the hidden state becomes increasingly stable on non-informative tokens, while token 0 and token 1 trigger much larger updates. This shows that the learned gate behavior is not only visible in the average update gate, but also changes how the GRU uses its recurrent state. The successful mixed run therefore implements the expected write-or-copy behavior at the level of hidden-state dynamics.

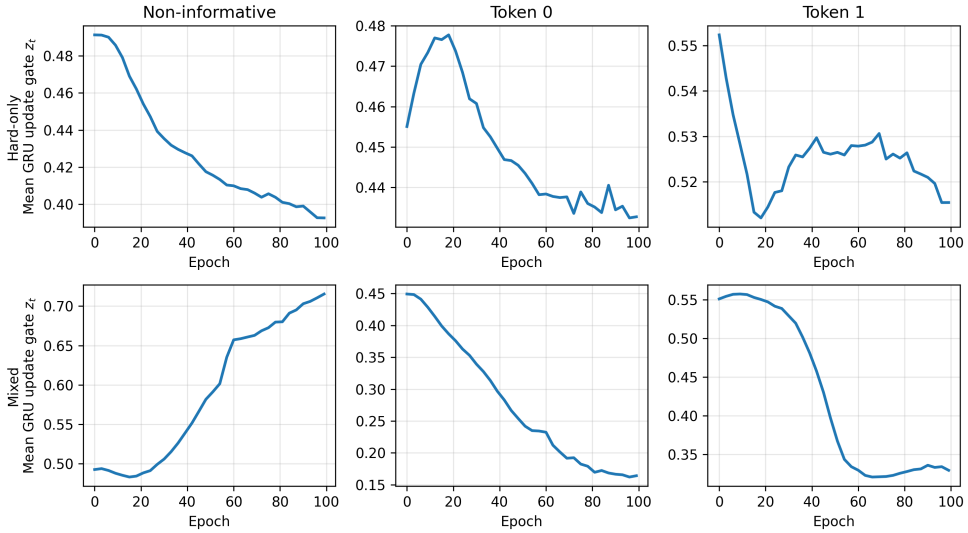


Figure 9: **Token-conditioned GRU update gate values during one representative GRU run.** Columns correspond to non-informative tokens, token 0, and token 1. In the binary temporal XOR task, token 0 and token 1 are the informative tokens that determine the XOR label. The top row shows the hard-only setting, and the bottom row shows the mixed setting. Each curve shows the mean update gate z_t over training epochs, conditioned on the current token type. In our GRU convention, the hidden-state update is $h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1}$, so a larger z_t copies more of the previous hidden state, while a smaller z_t writes more of the candidate state. In the hard-only setting, the gate values for informative and non-informative tokens are not clearly separated. In the mixed setting, the update gate becomes larger for non-informative tokens and smaller for informative tokens. Thus, the successful mixed run learns a token-dependent gate policy that copies across non-informative tokens and writes when an informative token appears.

Together, these probes illustrate the mechanism suggested by the accuracy results in one successful GRU run. Short-dependency samples do not merely improve the final test accuracy. In this run, they change what the GRU learns during training. Early hidden states receive a usable gradient signal during the learning transition, the update gate becomes more token-dependent, and the hidden state changes selectively: informative tokens produce large updates, while uninformative tokens are largely suppressed. This behavior matches the write-or-copy sub-task proposed above. Thus, mixed training can help by making the gate-learning problem easier first. Once this selective update policy is learned, the same recurrent model can apply it to hard long-range samples.

4.4 Exp 3: Partial Knowledge Transfer

The third experiment tests whether the learned gating policy transfers beyond the token group that receives short-dependency samples. The binary XOR experiment shows that short-dependency samples can help gated RNNs learn a write-or-copy policy. However, it does not distinguish whether the model learns a reusable policy or simply exploits the same informative token identities that appear in the short-dependency samples.

In this task, the informative tokens come from two disjoint groups, $\{0, 1\}$ and $\{2, 3\}$. The test distribution always contains hard samples for both groups. During mixed training, we add short-dependency samples only for the $\{2, 3\}$ group. Within this group, 40% of the training samples are short-dependency samples. Since the two groups are balanced, this corresponds to about 20% short-dependency samples in the full training set. Performance on the $\{2, 3\}$ group measures whether the model learns from the treated group, while performance on the untreated $\{0, 1\}$ group measures whether the benefit transfers beyond the specific group that received short-dependency samples.

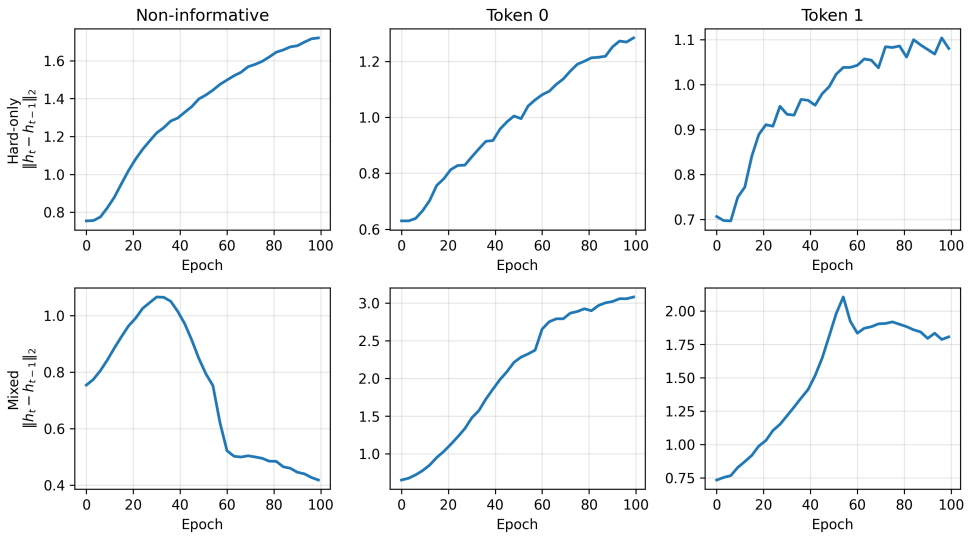


Figure 10: **Token-conditioned hidden-state changes during one representative GRU run.** Columns correspond to non-informative tokens, token 0, and token 1. The top row shows the hard-only setting, and the bottom row shows the mixed setting. Each curve shows the effective hidden-state update $\|h_t - h_{t-1}\|_2$ over training epochs, conditioned on the current token type. This quantity measures how much the recurrent state changes when a token is processed. In the hard-only setting, non-informative tokens change the hidden state as much as, or more than, the informative tokens. In the mixed setting, hidden-state changes become strongly token-selective: non-informative tokens produce much smaller updates, while token 0 and token 1 produce much larger updates. Thus, the learned gate pattern in mixed training is reflected in the recurrent state itself: informative tokens are written into memory, while non-informative tokens are increasingly suppressed.

Figures 11 and 12 show the hard-only baseline. As in the binary XOR experiment, long sequences are difficult when all samples require long-range memory. The overall accuracy approaches about 50% at long sequence lengths, which is above four-class chance accuracy but consistent with a model that can use the final informative token to identify the token group without reliably remembering the first informative token. The group-wise curves confirm this interpretation: within each token group, accuracy approaches the 50% baseline for guessing between the two group-specific labels. Thus, the multi-class task preserves the same long-range learning difficulty as binary temporal XOR, while also providing separate token groups for testing transfer.

Figures 13 and 14 provide a fully treated reference condition. Here, both informative-token groups receive short-dependency training samples, while validation and test samples remain hard for all classes. GRUs and LSTMs maintain high overall accuracy over much longer sequence lengths than in the hard-only baseline, and the group-wise curves show that both groups benefit. This result establishes an upper bound for the transfer experiment: high accuracy on both groups is achievable in the multi-class setting when both groups receive direct short-range support during training. The vanilla RNN does not show the same improvement, indicating that the benefit still depends on gated recurrence rather than on the changed training distribution alone.

Figures 15 and 16 show the asymmetric transfer setting, where only the $\{2, 3\}$ group receives short-dependency samples during training. For GRUs and LSTMs, the treated $\{2, 3\}$ group remains highly accurate across long sequence lengths, confirming that the direct benefit of short-dependency samples is preserved in the multi-class task. More importantly, the untreated $\{0, 1\}$ group also improves substantially compared with the hard-only baseline. This shows that the benefit is not restricted to the specific token identities that appear in the short-dependency samples. At the same time, the untreated $\{0, 1\}$ group does not reach the accuracy observed in the fully treated reference condition, where both groups receive short-dependency samples. The transfer is therefore incomplete.

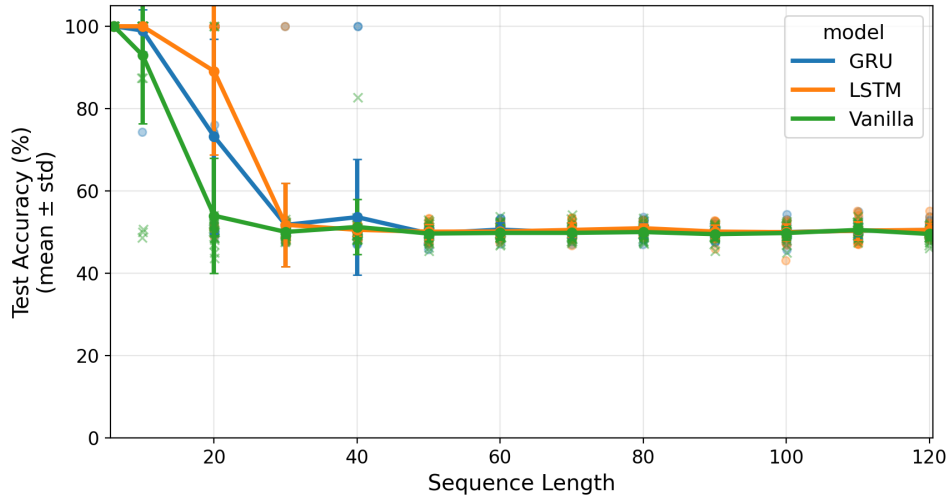


Figure 11: **Hard-only training on multi-class XOR.** All training, validation, and test samples are hard samples, with the two informative tokens fixed at the first and last positions. The task has four balanced classes, defined by two disjoint informative-token groups, $\{0, 1\}$ and $\{2, 3\}$. Lines show mean test accuracy, error bars show standard deviation, and faint markers show individual runs. As sequence length increases, all models collapse to about 50% accuracy. Thus, without short-dependency samples, all recurrent architectures fail to reliably combine the first and last informative tokens in the multi-class setting.

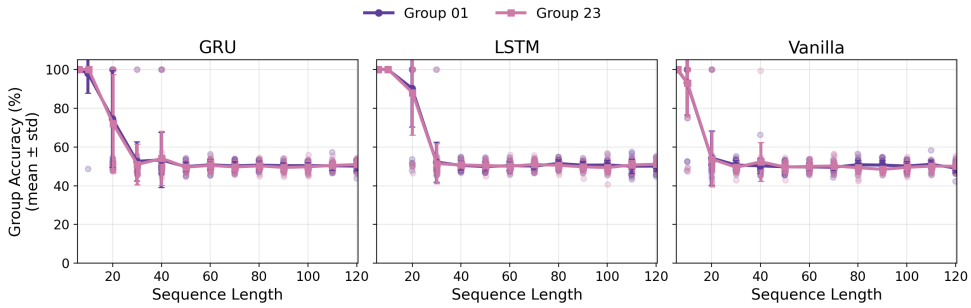


Figure 12: **Group-wise hard-only accuracy on multi-class XOR.** The $\{0, 1\}$ and $\{2, 3\}$ informative-token groups are evaluated separately for each model. Lines show mean group accuracy, error bars show standard deviation, and faint markers show individual runs. Both groups collapse to approximately 50% accuracy at long sequence lengths for all models, which corresponds to guessing between the two labels within that group. Thus, the hard-only failure is not restricted to one token group; both disjoint groups preserve the same long-range learning difficulty.

A natural interpretation is that the write-or-copy policy has both shared and group-specific parts. The non-informative tokens are shared across the two groups, so learning to copy hidden state across non-informative tokens in the treated $\{2, 3\}$ group can also help the untreated $\{0, 1\}$ group. However, writing useful information into memory still depends on recognizing the informative tokens themselves. Since token 0 and token 1 never appear in short-dependency samples, the model receives no direct short-range supervision for learning when to write these untreated informative tokens.

Together, the multi-class XOR results refine the conclusion from the binary task. Short-dependency samples do not only help the specific token group that receives them. When both groups receive short-dependency samples, both groups are highly learnable, showing that the task admits a strong long-range solution. When only one group receives short-dependency samples, the untreated group

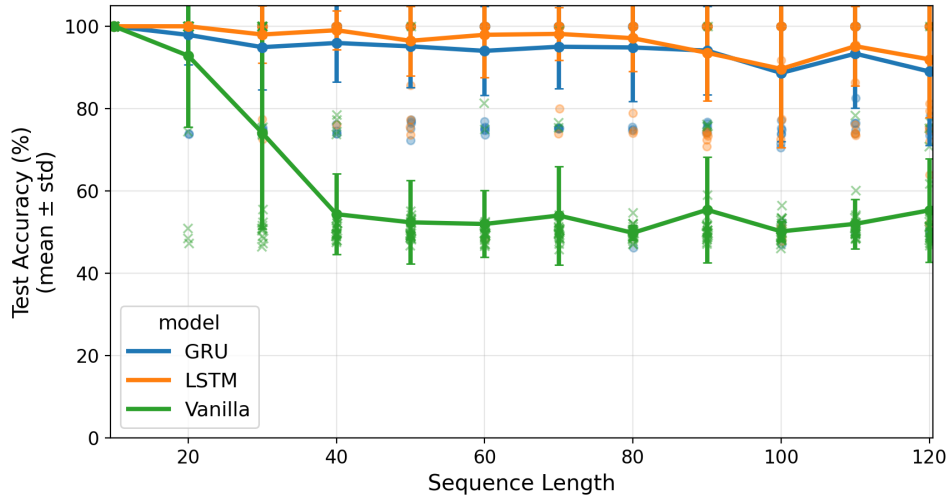


Figure 13: **Multi-class XOR with short-dependency samples added to both informative-token groups.** During training, both the $\{0, 1\}$ and $\{2, 3\}$ group receive short-dependency samples, corresponding to an expected proportion of 20% for each group. Validation and test samples remain hard samples for all four classes. Lines show mean test accuracy, error bars show standard deviation, and faint markers show individual runs. GRUs and LSTMs maintain high overall test accuracy over much longer sequence lengths, while the vanilla RNN remains near its hard-only baseline. Thus, when both groups receive short-dependency samples, gated RNNs can solve the multi-class long-range task for both groups.

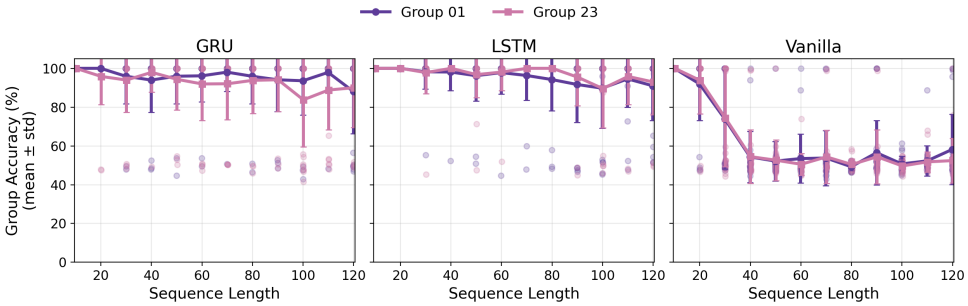


Figure 14: **Group-wise accuracy when short-dependency samples are added to both informative-token groups.** The $\{0, 1\}$ and $\{2, 3\}$ groups are evaluated separately for each model. Validation and test samples remain hard for both groups. Lines show mean group accuracy, error bars show standard deviation, and faint markers show individual runs. For GRUs and LSTMs, both groups remain highly accurate over long sequence lengths. Thus, the high performance in this setting is not restricted to one token group: when both groups receive short-dependency samples, both groups are learnable.

still improves over the hard-only baseline, showing transfer beyond the treated tokens. However, this transfer is incomplete, which suggests that mixed training induces a partially reusable recurrent strategy: copying across shared non-informative tokens can transfer, while writing group-specific informative tokens remains harder without direct short-dependency examples.

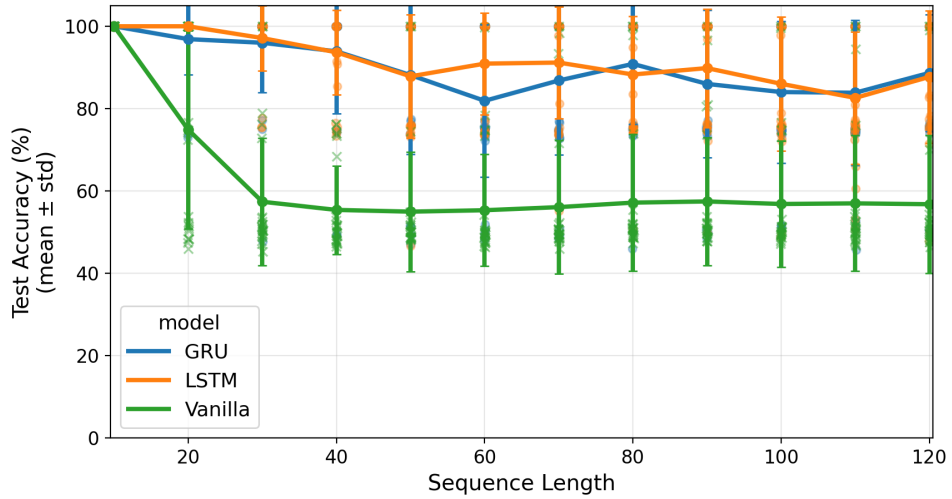


Figure 15: **Multi-class XOR with short-dependency samples added only to the $\{2, 3\}$ group.** During training, only the $\{2, 3\}$ informative-token group receives short-dependency samples, corresponding to an expected short-dependency-sample proportion of 20% over the full training set. The $\{0, 1\}$ group remains hard-only throughout training, and all validation and test samples remain hard. Lines show mean test accuracy, error bars show standard deviation, and faint markers show individual runs. If the short-dependency samples helped only the treated $\{2, 3\}$ group while the untreated $\{0, 1\}$ group stayed at its within-group chance level, the expected overall accuracy would be about 75%. GRUs and LSTMs remain above this no-transfer reference level over long sequence lengths. Thus, the overall accuracy already suggests transfer beyond the treated group, which is tested directly with the group-wise accuracies in Figure 16.

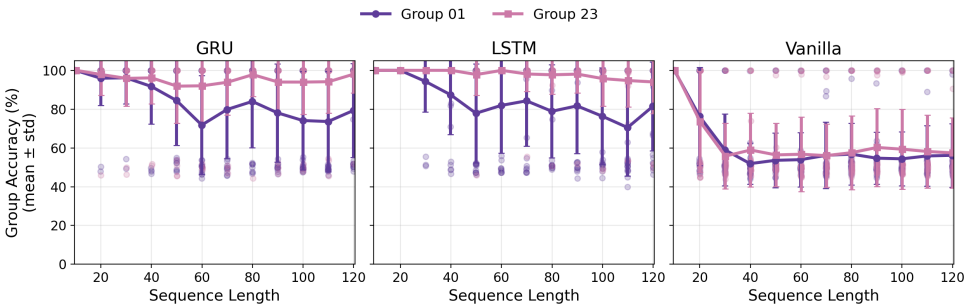


Figure 16: **Group-wise accuracy when short-dependency samples are added only to the $\{2, 3\}$ group.** The purple shows the untreated $\{0, 1\}$ group, and the pink shows the treated $\{2, 3\}$ group. Lines show mean group accuracy, error bars show standard deviation, and faint markers show individual runs. For GRUs and LSTMs, the treated $\{2, 3\}$ group remains highly accurate, while the untreated $\{0, 1\}$ group also improves substantially over the hard-only baseline. However, the untreated group does not reach the accuracy obtained when both groups receive short-dependency samples. Thus, the benefit transfers beyond the treated token group, but the transfer is only partial.

5 Limitations and Future Work

This work studies gate learning through controlled synthetic tasks. The advantage of this setting is that the source of difficulty can be isolated: long and short-dependency samples use the same token identities and the same label rule, but differ in the temporal distance between informative tokens and the final loss. This makes the experiments suitable for testing whether shorter dependency paths

can help gated RNNs first learn a write-or-copy behavior. The limitation is that temporal XOR is intentionally simple. The task does not contain the semantic, visual, or linguistic structure present in real sequence problems. The results therefore demonstrate a mechanism in a controlled setting, rather than showing that the same mechanism is already sufficient for real-world long-range modeling.

In a broader sense, the limitation concerns the relation between this work and curriculum learning (Bengio et al., 2009; Cirik et al., 2017; Sadasivan & Dasgupta, 2021). The short-dependency samples used here resemble a curriculum because they are easier for the model to learn first. However, the intended control is more specific than general curriculum learning: the short-dependency samples shorten the gradient path while preserving the same label rule and token vocabulary. In less controlled tasks, these factors are difficult to separate. For example, in preliminary experiments with image classification as a pixel sequence, training first on shorter subsampled sequences and then on longer sequences improved gated models more than vanilla RNNs. However, subsampling also changes the visual information available to the model. A shorter pixel sequence may have a shorter recurrent gradient path, but it may also be visually less informative or semantically harder. This makes it unclear whether the benefit comes from easier optimization, from a conventional curriculum effect, or from a change in task content.

Future work should therefore separate gradient difficulty from task difficulty more carefully. The current approach designs tasks where the semantic information is held fixed while only the temporal placement of relevant evidence changes. A future direction is to construct paired curricula where one curriculum shortens dependency length but does not simplify the input, while another simplifies the input without shortening the dependency length. Such factorial designs would make it possible to test whether gated RNNs benefit specifically from shorter gradient paths, from easier examples in the usual curriculum-learning sense, or from both.

Finally, the controlled insight should be tested on broader long-range sequence benchmarks only after this separation is clear. Candidate settings include Long Range Arena (Tay et al., 2021), bAbI-style reasoning tasks (Weston et al., 2016), denoising or selective-copying tasks (Jing et al., 2019; Gu & Dao, 2023), and sequence versions of vision datasets (Le et al., 2015; Arjovsky et al., 2016; Wisdom et al., 2016). For these tasks, the main challenge is not only to improve accuracy, but to define easy and hard variants that change the gate-learning difficulty without also changing the target task in uncontrolled ways. The controlled experiments in this paper suggest what to look for: easy variants should help gated models first learn when to write task-relevant information and when to copy state across non-informative inputs. A real-world extension would be convincing only if it preserves this distinction and shows that the same mechanism explains the improvement.

6 Conclusion

Gated recurrent networks are usually explained by their ability to create copy paths through time. This explanation is correct, but it assumes that useful gates have already been learned. This paper studied the missing learning step: how gated RNNs learn when to write new information and when to copy previous state before the long-range task has already been solved.

Through controlled temporal XOR experiments, we showed that gated models do not solve long-range dependencies by architecture alone. When all training samples require a first-to-last dependency, vanilla RNNs, GRUs, and LSTMs all eventually fail as sequence length increases. When the training set also contains short-dependency samples with the same label rule but shorter dependency paths, GRUs and LSTMs solve the original hard test task over much longer sequences, while vanilla RNNs remain much less effective. This contrast shows that the advantage of gated RNNs depends not only on having a possible copy path, but also on making the gate-learning problem learnable.

The probe results support this interpretation. In a successful mixed-training GRU run, early recurrent states receive larger gradient signals than in the hard-only setting. The update gate then becomes token-dependent: non-informative tokens increasingly copy the previous hidden state, while informative tokens increasingly write new information. The corresponding hidden-state changes show the same pattern, indicating that the learned gate behavior affects the recurrent state dynamics.

The multi-class XOR experiments further show that this learned behavior can transfer partially beyond the token group that receives short-dependency samples. When both informative-token groups receive short-dependency samples, gated models solve both groups. When only one group

receives short-dependency samples, the untreated group still improves over the hard-only baseline, but remains below the fully treated reference condition. This suggests that part of the learned recurrent strategy is reusable across token groups, especially copying across shared non-informative tokens.

Overall, the results refine the standard story of gated RNNs. Gates can provide long-range memory paths, but those paths are useful only after the model has learned how to use them. Short-dependency samples can serve as a learning scaffold for this gate behavior, allowing gated RNNs to acquire a selective update policy before applying it to harder long-range samples. The main conclusion is therefore not that gates automatically solve long-range dependencies, but that gates help when the data first makes the gate-learning problem simple enough to learn.

REFERENCES

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1120–1128, 2016.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 41–48, 2009.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

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*, pp. 1724–1734, 2014.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS Deep Learning Workshop*, 2014.

Volkan Cirik, Eduard Hovy, and Louis-Philippe Morency. Visualizing and understanding curriculum learning for long short-term memory networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

Daniel C. Dennett. *Consciousness Explained*. Little, Brown and Company, Boston, 1991.

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

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. Colorless green recurrent networks dream hierarchically. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 1195–1205, 2018.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2): 107–116, 1998.

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

Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljačić, and Yoshua Bengio. Gated orthogonal recurrent units: On learning to forget. *Neural Computation*, 31(4): 765–783, 2019. doi: 10.1162/neco_a_01174.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2342–2350, 2015.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. In *International Conference on Learning Representations Workshop*, 2016.

Anthony Kenny. The homunculus fallacy. In Marjorie Grene and I. Prigogine (eds.), *Interpretations of Life and Mind: Essays around the Problem of Reduction*, pp. 155–165. Humanities Press, New York, 1971.

Anastasia Kobzeva, Suhas Arehalli, Tal Linzen, and Dave Kush. LSTMs can learn basic wh- and relative clause dependencies in norwegian. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 2020.

Yair Lakretz, German Kruszewski, Theo Desbordes, Dieuwke Hupkes, Stanislas Dehaene, and Marco Baroni. The emergence of number and syntax units in LSTM language models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 11–20, 2019.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4: 521–535, 2016.

Seol-Hyun Noh. Analysis of gradient vanishing of RNNs and performance comparison. *Information*, 12(11):442, 2021. doi: 10.3390/info12110442.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

Gilbert Ryle. *The Concept of Mind*. The University of Chicago Press, Chicago, 2000. Originally published in 1949.

Vinu Sankar Sadasivan and Anirban Dasgupta. Statistical measures for defining curriculum scoring function. In *Workshop on SubSetML: Subset Selection in Machine Learning: From Theory to Practice at the 38th International Conference on Machine Learning*, 2021.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qVyeW-grC2k`.

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*, 2017.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. In *International Conference on Learning Representations*, 2016.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Scott Wisdom, Thomas Powers, John R. Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

A Additional Experimental Results

A.1 Sensitivity to the Short-Dependency Window Size

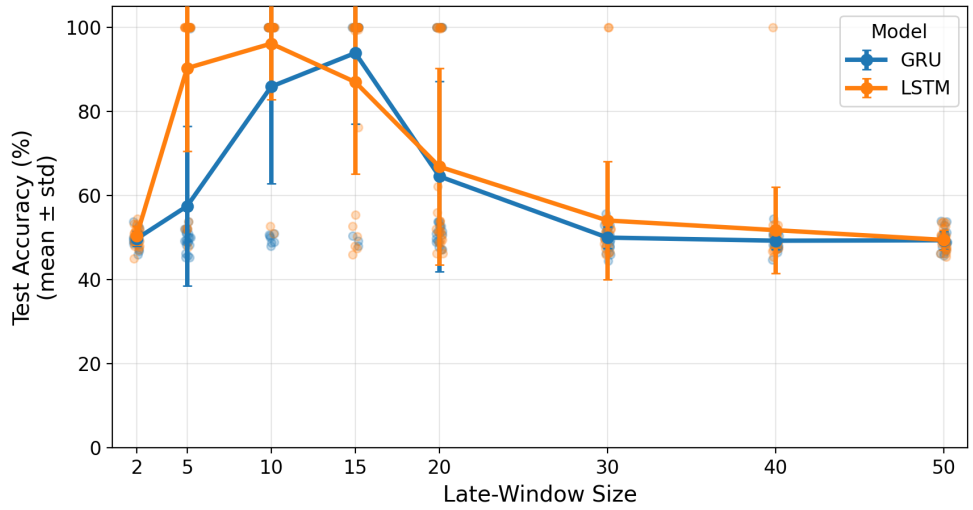


Figure A.1: **Sensitivity to the short-dependency suffix-window size.** Sequence length is fixed to $T = 100$, the short-dependency-sample proportion is fixed to $p_{\text{easy}} = 40\%$, and test samples are always hard. Very small windows and very large windows both reduce performance. Intermediate windows work best: they are short enough to provide usable gradient signals, but large enough to include non-informative tokens from which the model can learn when to copy state. The main experiments use suffix-window size 10, which lies in this high-performing regime.

In the main experiments, short-dependency samples place the informative tokens inside a suffix window of size 10. To test whether this choice is special, we repeat the mixed binary temporal XOR experiment at fixed sequence length $T = 100$ while varying the suffix-window size. Figure A.1 shows an inverse-U pattern. Very small windows provide limited non-informative context, while very large windows make the short-dependency samples less easy by increasing the dependency length again. The best performance is obtained for intermediate window sizes, including the window size used in the main experiments.

A.2 Additional Probe Results

The main text analyzes one successful GRU run to illustrate how mixed training changes training dynamics, gradient availability, gate behavior, and hidden-state dynamics. Here we include additional probes for the vanilla RNN and LSTM at the same sequence length.

Vanilla RNN. The vanilla RNN provides a useful comparison because it receives the same mixed training distribution but has no explicit gate-controlled copy path. Figure A.2 shows that mixed training does not lead to successful hard-validation generalization. Figure A.3 shows that the hidden-state gradient profile remains strongly biased toward later timesteps. Finally, Figure A.4 shows that hidden-state changes do not become selective by token type: non-informative tokens continue to perturb the recurrent state as much as informative tokens. Together, these probes support the interpretation that the benefit of short-dependency samples depends on the gated model's ability to convert short-range learning signals into a selective recurrent update policy.

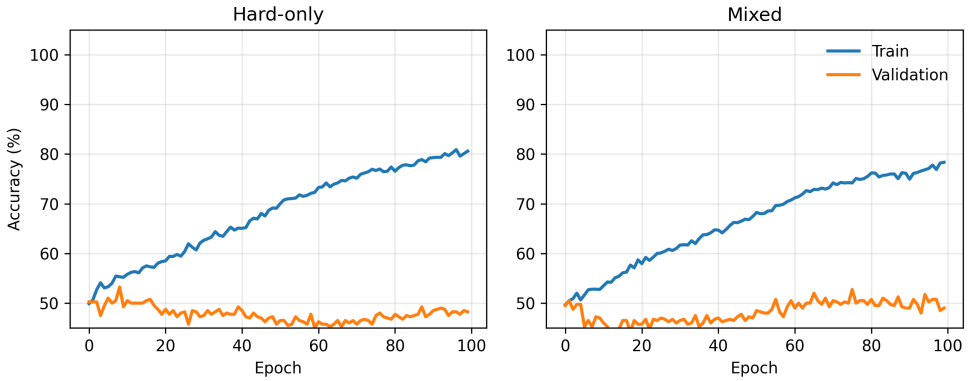


Figure A.2: **Training dynamics of one representative vanilla RNN run.** The left panel shows hard-only training. The right panel shows mixed training. Both panels show training accuracy curves; the validation curve is always measured on hard samples, with informative tokens fixed at the first and last positions. In both settings, training accuracy increases while hard-validation accuracy remains near chance. Thus, unlike the successful GRU run in the main text, the vanilla RNN does not turn mixed training into a reusable long-range rule.

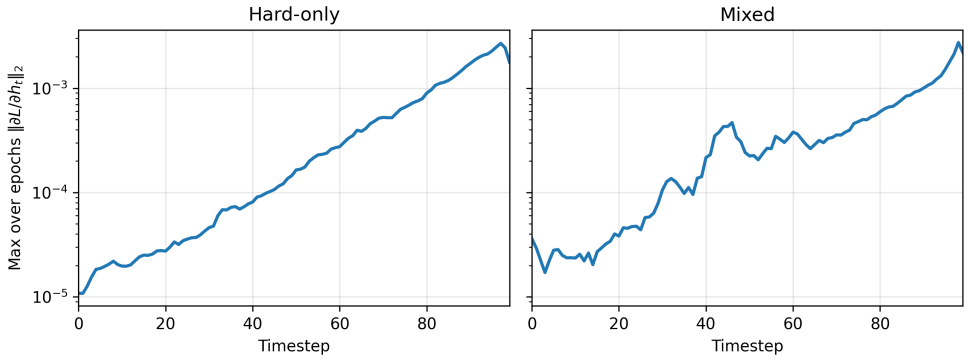


Figure A.3: **Maximum gradient received at each timestep during one representative vanilla RNN run.** For every timestep t, the figure reports the maximum value of $\|\partial L/\partial h_t\|_2$ observed over training epochs. The y-axis is logarithmic. In both hard-only and mixed training, the largest gradients are concentrated near later timesteps and become much smaller toward the beginning of the sequence. Thus, mixed training does not create the same broad early-timestep gradient availability observed for the successful GRU run.

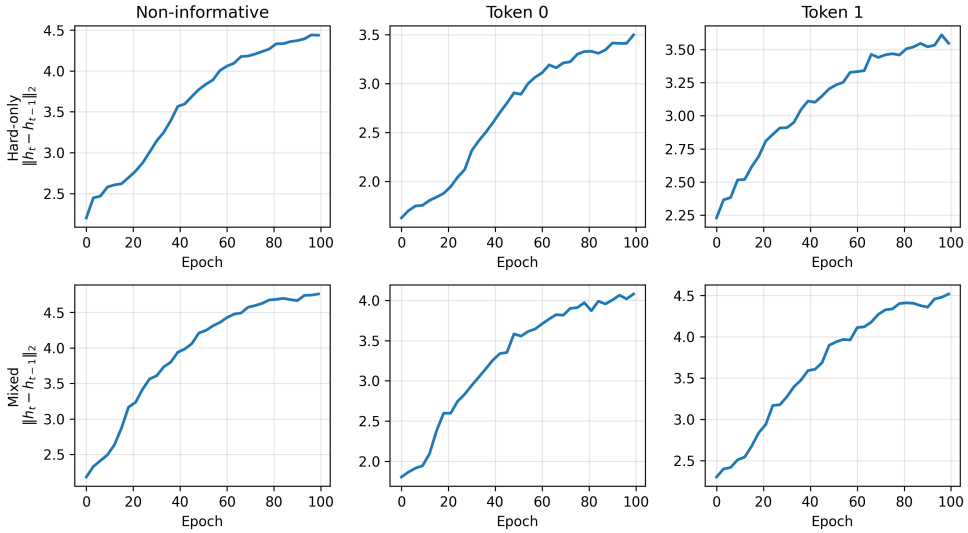


Figure A.4: **Token-conditioned hidden-state changes during one representative vanilla RNN run.** Columns correspond to non-informative tokens, token 0, and token 1. The top row shows the hard-only setting, and the bottom row shows the mixed setting. Each curve shows the effective hidden-state update $\|h_t - h_{t-1}\|_2$ over training epochs, conditioned on the current token type. Unlike the gated models, the vanilla RNN does not develop a selective update pattern: non-informative tokens continue to change the hidden state strongly, and their behavior is not clearly suppressed relative to informative tokens.

LSTM. We also probe a representative LSTM run at $T = 100$. The LSTM provides a second gated architecture for comparison with the GRU analysis in the main text. Figures A.5 and A.6 show the same qualitative pattern as the GRU: mixed training leads to a sharp transition in hard-validation accuracy, and large gradients reach early timesteps during training. Figures A.7 and A.8 inspect the internal dynamics. The forget-gate averages are less directly interpretable than the GRU update gate, because an LSTM update depends jointly on the forget gate, input gate, candidate cell update, and output gate. However, the hidden-state change probe shows a clearer selective-update pattern: in the mixed setting, non-informative tokens produce smaller state changes, while informative tokens produce larger changes. For this reason, the main text uses the GRU as the clearest gate-level visualization, while the LSTM probes here support the broader conclusion that mixed training also changes the learning dynamics of another gated recurrent architecture.

A.3 Symmetry Check for the Multi-Class XOR Setting

In the main multi-class XOR transfer experiment, short-dependency samples are added only to the $\{2, 3\}$ token group. Here we repeat the asymmetric experiment in the opposite direction: short-dependency samples are added only to the $\{0, 1\}$ group, while the $\{2, 3\}$ group remains hard-only during training. All validation and test samples remain hard samples for both groups.

Figures A.9 and A.10 show a qualitatively symmetric result to the setting in the main text. GRUs and LSTMs again remain above the 75% no-transfer level in overall accuracy. The group-wise curves show that the treated $\{0, 1\}$ group is learned reliably, while the untreated $\{2, 3\}$ group also improves compared with the hard-only baseline. However, the untreated group remains less stable than the treated group and does not match the fully treated reference condition from the main text. This supports the interpretation that the benefit of short-dependency samples is not specific to the particular $\{2, 3\}$ group used in the main experiment. Instead, mixed training induces a partially reusable recurrent strategy: part of the behavior transfers across token groups, but direct short-dependency samples are still useful for learning how to write group-specific informative tokens.

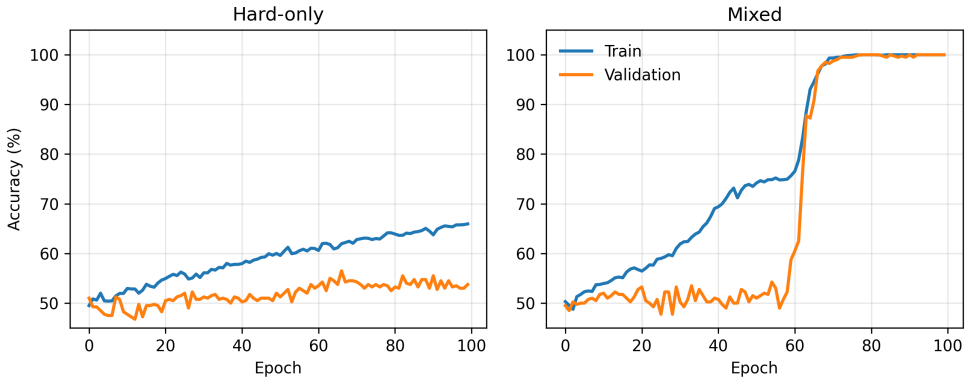


Figure A.5: **Training dynamics of one representative LSTM run.** The left panel shows hard-only training. The right panel shows mixed training. Both panels show training accuracy curves; the validation curve is always measured on hard samples, with informative tokens fixed at the first and last positions. In the hard-only setting, training accuracy increases slowly while validation accuracy remains close to chance. In the mixed setting, training accuracy first improves on the mixed training distribution and then undergoes a sharp transition together with hard-validation accuracy. Thus, as for the GRU in the main text, short-dependency samples help the LSTM discover a rule that transfers to hard long-range samples.

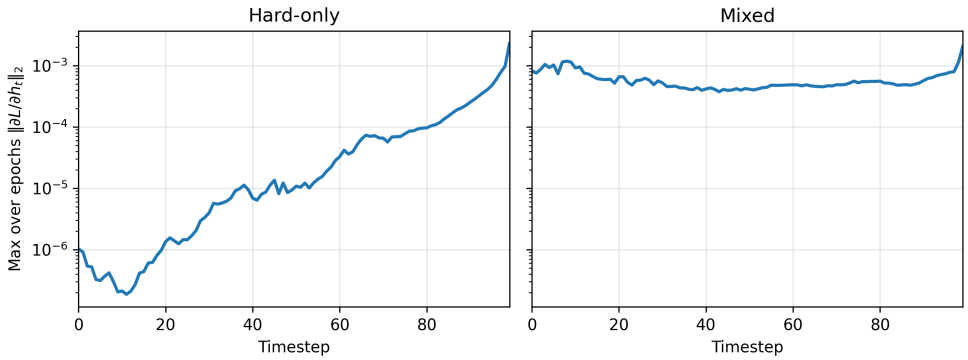


Figure A.6: **Maximum gradient received at each timestep during one representative LSTM run.** For every timestep t, the figure reports the maximum value of $\|\partial L/\partial h_t\|_2$ observed over training epochs. The y-axis is logarithmic. In the hard-only setting, the largest gradient signal decays strongly toward early timesteps. In the mixed setting, large gradients reach all timesteps at some point during training. Thus, mixed training changes the optimization conditions for the LSTM in the same qualitative way as for the GRU.

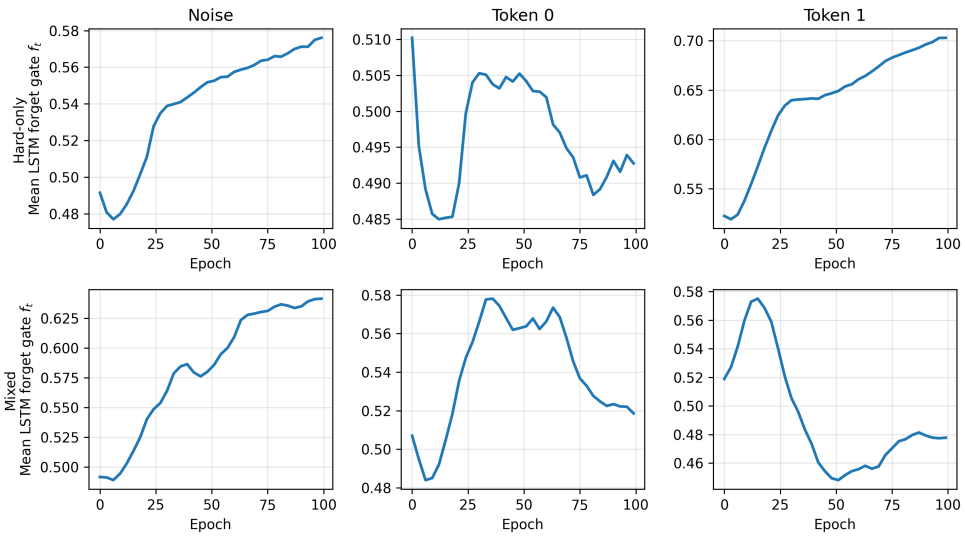


Figure A.7: **Token-conditioned LSTM forget gate values during one representative LSTM run.** Columns correspond to non-informative tokens, token 0, and token 1. The top row shows the hard-only setting, and the bottom row shows the mixed setting. Each curve shows the mean forget gate f_t over training epochs, conditioned on the current token type. Larger forget-gate values preserve more of the previous cell state. In the mixed setting, the forget gate tends to increase for non-informative tokens, which is consistent with copying state across noise. The informative-token curves are less cleanly interpretable, because LSTM state updates depend jointly on the forget gate, input gate, candidate cell update, and output gate. Therefore, this probe should be read as a diagnostic view of one gate, rather than as a complete explanation of the LSTM mechanism.

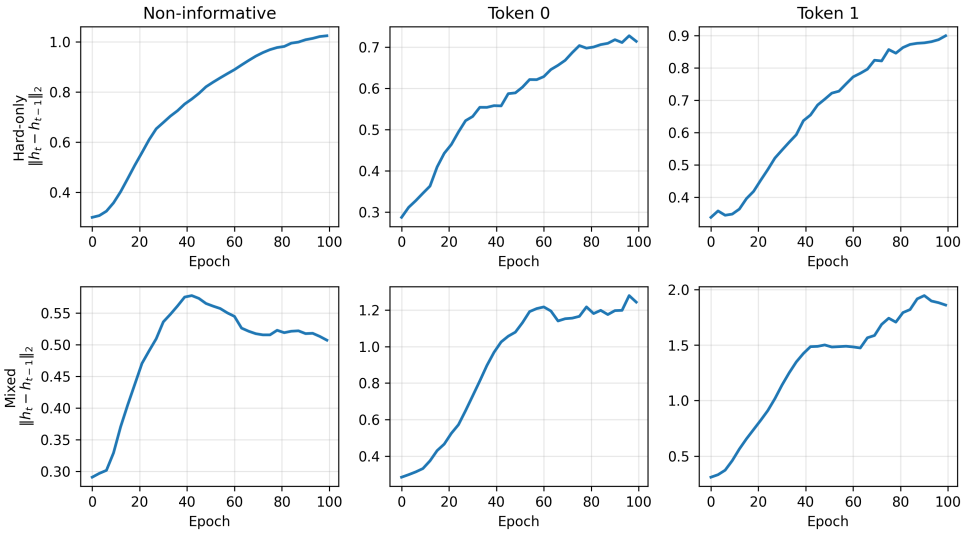


Figure A.8: **Token-conditioned hidden-state changes during one representative LSTM run.** Columns correspond to non-informative tokens, token 0, and token 1. The top row shows the hard-only setting, and the bottom row shows the mixed setting. Each curve shows the effective hidden-state update $\|h_t - h_{t-1}\|_2$ over training epochs, conditioned on the current token type. In the hard-only setting, non-informative tokens and informative tokens all produce increasing hidden-state changes. In the mixed setting, hidden-state changes become more token-selective: non-informative tokens are increasingly suppressed, while token 0 and token 1 produce larger updates. Thus, although the forget gate alone is not fully explanatory, the LSTM state dynamics are consistent with the same write-or-copy behavior observed for the GRU.

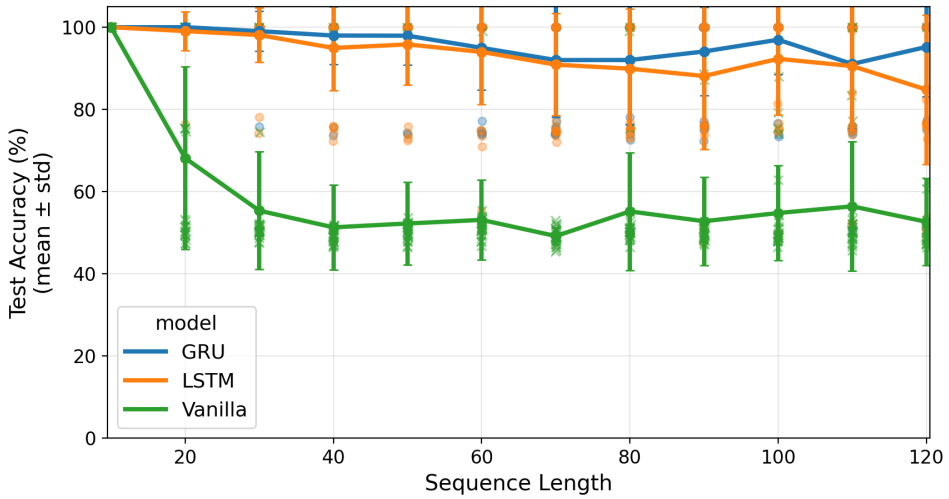


Figure A.9: **Multi-class XOR with short-dependency samples added only to the $\{0, 1\}$ group.** This is the symmetric counterpart of the transfer experiment in the main text, where short-dependency samples are added only to the $\{2, 3\}$ group. During training, only the $\{0, 1\}$ informative-token group receives short-dependency samples, corresponding to an expected short-dependency-sample proportion of 20% over the full training set. The $\{2, 3\}$ group remains hard-only throughout training, and all validation and test samples remain hard. Lines show mean test accuracy, error bars show standard deviation, and faint markers show individual runs. GRUs and LSTMs remain well above the 75% no-transfer reference accuracy over long sequence lengths, while the vanilla RNN remains close to its baseline.

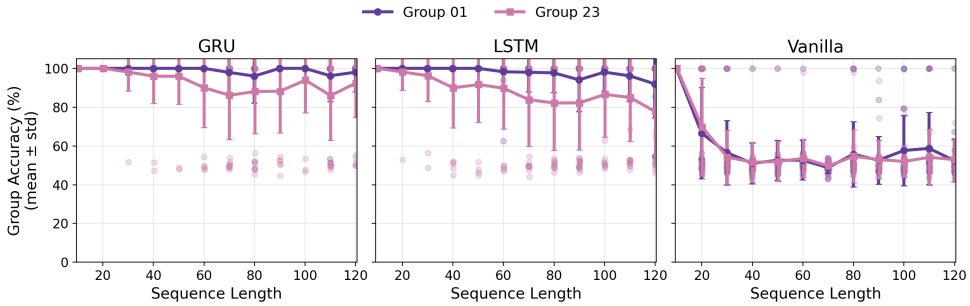


Figure A.10: **Group-wise accuracy when short-dependency samples are added only to the $\{0, 1\}$ group.** The purple curve shows the treated $\{0, 1\}$ group, and the pink curve shows the untreated $\{2, 3\}$ group. Lines show mean group accuracy, error bars show standard deviation, and faint markers show individual runs. For GRUs and LSTMs, the treated group remains highly accurate, while the untreated group also improves over the hard-only baseline. The untreated group is less stable than the treated group and remains below the fully treated reference condition from the main text. Thus, the reverse asymmetric experiment supports the same conclusion: the learned behavior transfers beyond the treated token group, but the transfer is partial rather than complete.

Acknowledgements

The Devil is in the details, as the saying goes. An MSc thesis like this has a lot of devils, any one of which will bite you if you don't watch out. Fortunately, I know a lot of angels.

First, I would like to extend my deepest gratitude to my advisor, Prof. Jan van Gemert. This thesis grew from a question I first asked in his *Machine and Deep Learning* course, and I am grateful that he treated that question not as a small interruption, but as a genuine research direction. Throughout the project, Jan helped me keep sight of the larger story: what hypothesis we were really testing, what a controlled experiment should isolate, and what kind of conceptual insight would make the work meaningful. His guidance shaped not only the direction of this thesis, but also my understanding of what good research can look like.

I am also deeply grateful to my daily supervisor, Alex Manolache, for his patient and detailed supervision throughout the project. While Jan often helped me think about the high-level research question, Alex helped me turn the question into concrete implementations, experiments and results. In our weekly meetings, he discussed experimental settings, debugging, result interpretation, and writing with great care. Many details that now look natural in the final thesis became clear only through these discussions.

I would also like to thank the PhD candidates and fellow master's students in the weekly group meetings. Their questions and comments often came from perspectives outside the immediate project, which made them especially valuable. Some suggestions helped me notice weaknesses in the setup, others helped me interpret results more carefully, and many simply reminded me that research is improved by generous discussion.

I am grateful to my family and friends for their support throughout my master's study. Their encouragement, patience, and trust gave me strength during a period that was both exciting and demanding. Even when they were far away from the technical details of this thesis, they helped make it possible for me to continue working on it.

Finally, I would like to thank my love, Shuang Liu. During the whole process of this thesis, she stayed beside me with warmth, patience, and quiet support. She listened to my worries, celebrated small progress with me, and reminded me that the value of this journey was not only in the final result, but also in the people who walked with me and in the person I became along the way. For that, I am sincerely grateful.

References

[1] Maximilian Beck et al. “xLSTM: Extended Long Short-Term Memory”. In: *Advances in Neural Information Processing Systems*. Vol. 37. arXiv:2405.04517. 2024.

[2] Ali Behrouz et al. “Memory Caching: RNNs with Growing Memory”. In: *arXiv preprint arXiv:2602.24281* (2026).

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning Long-Term Dependencies with Gradient Descent is Difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[4] Kyunghyun Cho et al. “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)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[5] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *arXiv preprint arXiv:1412.3555* (2014). arXiv: 1412.3555 [cs.NE].

[6] Tri Dao and Albert Gu. “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality”. In: *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. Proceedings of Machine Learning Research. arXiv:2405.21060. 2024.

[7] Daniel C. Dennett. *Consciousness Explained*. Boston: Little, Brown and Company, 1991.

[8] M. Reza Ebrahimi et al. “On the “Induction Bias” in Sequence Models”. In: *Proceedings of the 43rd International Conference on Machine Learning*. Vol. 306. Proceedings of Machine Learning Research. arXiv:2602.18333v2. 2026.

[9] Jeffrey L. Elman. “Finding Structure in Time”. In: *Cognitive Science* 14.2 (1990), pp. 179–211. DOI: 10.1207/s15516709cog1402_1.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. https://www.deeplearningbook.org. MIT Press, 2016.

[11] Samuel James Greydanus and Dmitry Kobak. “Scaling Down Deep Learning with MNIST-1D”. In: *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. Proceedings of Machine Learning Research. 2024, pp. 16404–16415.

[12] Albert Gu and Tri Dao. “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”. In: *arXiv preprint arXiv:2312.00752* (2023). DOI: 10.48550/arXiv.2312.00752.

[13] Sepp Hochreiter. “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.2 (1998), pp. 107–116.

[14] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[15] Max Jaderberg et al. “Spatial Transformer Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015.

[16] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. “An Empirical Exploration of Recurrent Network Architectures”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. JMLR Workshop and Conference Proceedings. Lille, France, 2015, pp. 2342–2350.

[17] Anthony Kenny. “The Homunculus Fallacy”. In: *Interpretations of Life and Mind: Essays around the Problem of Reduction*. Ed. by Marjorie Grene and I. Prigogine. New York: Humanities Press, 1971, pp. 155–165.

[18] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the Difficulty of Training Recurrent Neural Networks”. In: *International Conference on Machine Learning*. 2013, pp. 1310–1318.

[19] Bo Peng et al. “RWKV: Reinventing RNNs for the Transformer Era”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 2023, pp. 14048–14077.

[20] Zihan Qiu et al. “Gated Attention for Large Language Models: Non-Linearity, Sparsity, and Attention-Sink-Free”. In: *Advances in Neural Information Processing Systems*. Oral presentation; Best Paper Award; arXiv:2505.06708. 2025.

[21] Gilbert Ryle. *The Concept of Mind*. Originally published in 1949. Chicago: The University of Chicago Press, 2000.

[22] Yair Schiff et al. “Caduceus: Bi-Directional Equivariant Long-Range DNA Sequence Modeling”. In: *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. Proceedings of Machine Learning Research. 2024.

[23] Yutao Sun et al. “Retentive Network: A Successor to Transformer for Large Language Models”. In: *arXiv preprint arXiv:2307.08621* (2023). DOI: 10.48550/arXiv.2307.08621.

[24] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[25] Shuguang Wang and Yuanjing Wang. “Task-Driven Fixation Network: An Efficient Architecture with Fixation Selection”. In: *arXiv preprint arXiv:2501.01548* (2025). DOI: 10.48550/arXiv.2501.01548.

[26] Paul J. Werbos. “Backpropagation Through Time: What It Does and How to Do It”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[27] Ronald J. Williams and David Zipser. “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks”. In: *Neural Computation* 1.2 (1989), pp. 270–280. DOI: 10.1162/neco.1989.1.2.270.

[28] Aston Zhang et al. *Dive into Deep Learning*. https://d2l.ai. Cambridge University Press, 2023.

A

Declaration of AI Usage

Following the university's academic honesty and transparency guidelines, this appendix chapter lists the generative AI tools used during this project. As the author, I keep full creative and intellectual responsibility for all the code, text, and final results in this thesis.

A.1. Coding Support

AI tools were used as assistants to speed up coding tasks and help with data processing, without replacing core research contributions.

- **Tools Used:** GitHub Copilot and OpenAI Codex.
- **Scope of Work:** The tools helped write boilerplate code, fix bugs, and make the code more readable. They were also used to create scripts for data analysis and visualization. All generated code was manually verified and tested before use.

A.2. Writing Support

AI tools were used to assist with the writing process of this thesis.

- **Tools Used:** OpenAI ChatGPT and Google Gemini.
- **Scope of Work:** The tools were used to improve language fluency, word choice, clarity of expression, and the organization of selected passages. They were also used to discuss possible ways of presenting and refining the research narrative.

A.3. AI Usage Statement

All AI-assisted content was carefully reviewed, edited, and verified by the author. The research questions, experimental design, implementation, results, analysis, and conclusions remain the author's own responsibility. The author ensured that the use of generative AI did not introduce conceptual or factual deviations from the original research content. Additionally, no private, sensitive, or confidential data was uploaded to any external AI tools during this project.