PRETRAINING RECURRENT NETWORKS WITHOUT RECURRENCE

Anonymous authors
Paper under double-blind review

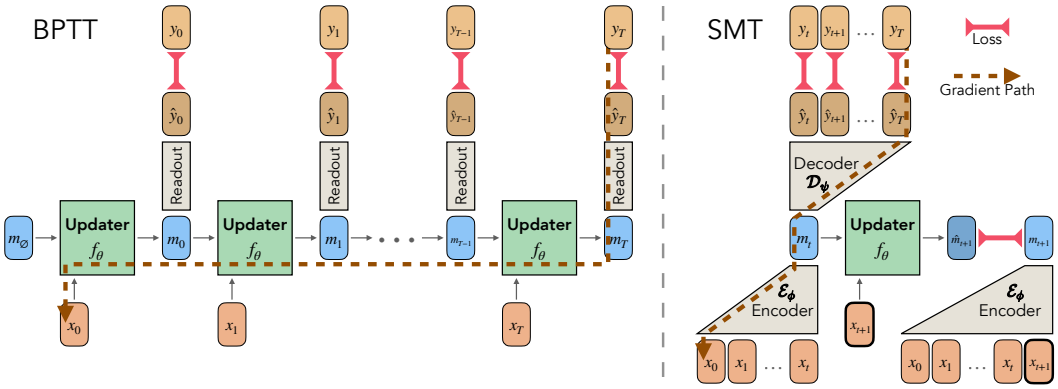

Figure 1: **BPTT vs SMT. Left:** BPTT trains an RNN by recurrently unrolling the "updater" network in time, and backpropagating gradients through the entire graph. **Right:** Supervised Memory Training (SMT) trains an RNN with supervised learning on one-step memory transition labels, which are generated by a Transformer encoder-decoder model pair trained to produce *predictive states*. SMT is fully time-parallel. In SMT, the longest gradient path between tokens is $\mathcal{O}(1)$ (compared to $\mathcal{O}(T)$ in BPTT), which stabilizes gradients, making learning long-range dependencies qualitatively easier.

ABSTRACT

Training recurrent neural networks (RNNs) requires assigning credit across long sequences of computations. Standard backpropagation through time (BPTT) addresses this problem poorly: it is sequential in time, limiting parallelism, and suffers from vanishing or exploding gradients, making long-range associations difficult to learn. We propose *Supervised Memory Training* (SMT), a method for training nonlinear RNNs that sidesteps recurrent credit propagation entirely by reducing RNN training to supervised learning on one-step memory transition labels $(m_t, x_{t+1}) \to m_{t+1}$. SMT acquires these memory labels by training a Transformer-based encoder on a predictive state objective—retaining only information from the past necessary to predict the future. By decoupling *what to remember* from *how to update memory*, SMT enables time-parallel RNN training with a stable $\mathcal{O}(1)$ length gradient path between any two tokens—without ever unrolling the RNN. We find that SMT outperforms BPTT when pretraining various RNN architectures on tasks like language modeling and pixel sequence modeling. SMT enables nonlinear RNNs to better capture long-range dependencies and train in parallel, potentially unlocking the scaling of models that build temporal abstractions of past experience.

1 INTRODUCTION

Recurrent neural networks (RNNs) store information about the past that will only become useful in the future. The core training challenge is that the utility of a memory may be delayed: many intermediate computations intervene between writing information and eventually using it. These intervening steps confound learning the correct associations, a problem known as *credit assignment* (Minsky, 1961).

The standard approach, backpropagation through time (BPTT), assigns credit across a sequence by unrolling the RNN in time and propagating gradients backward through the resulting computation graph (Rumelhart et al., 1986; Werbos, 1990). Although conceptually well-motivated, BPTT is

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

sequential in time and suffers from unstable high variance gradients that may vanish or explode (Pascanu et al., 2013b). The lack of time-parallelism makes BPTT scale poorly, while its gradient instability makes learning long-range associations difficult, as credit must propagate across up to $\mathcal{O}(T)$ steps (Bengio et al., 1994). Is recurrent credit propagation unavoidable?

In this paper, we propose *Supervised Memory Training* (SMT), a method to train nonlinear RNNs that sidesteps recurrent credit propagation by reducing the problem to supervised learning. Suppose we had access to the optimal memory state at each timestep, m_t^*. Then, RNN training reduces to learning the one-step update $(m_t^*, x_{t+1}) \to m_{t+1}^*$ using standard supervised objectives.

The challenge, of course, is how to actually obtain such memory labels. In this paper, we assert that an effective memory is a sufficient statistic of the past for predicting the future, i.e., a predictive state (Littman & Sutton, 2001). The past is typically viewed as a sequence, suggesting that memory must be computed sequentially over time. Our key insight is that, by augmenting each observation with its timestamp, the past can instead be losslessly represented as a *set* of timestamped events, rather than a sequence. Under this reparameterization, the optimal memory becomes a permutation-invariant function of this set, and can therefore be estimated using models that operate in parallel over time. This reframing allows us to train memory representations *without recurrently propagating credit through time*.

In practice, we train a Transformer encoder model to embed the past context into a memory that a separate decoder can use to predict the future. This objective operationalizes the notion of a predictive state: a representation of the past that retains only the information needed to predict the future and nothing more. Once this teacher encoder has learned to construct such memory representations, the RNN can then focus on learning the now much simpler task of updating that memory over time.

In essence, SMT decouples learning *what to remember* (memory representation), which is a non-sequential problem, from learning *how to update memory* (memory dynamics), which is a sequential process but can be supervised one-step at a time. This decoupling enables time-parallel training of nonlinear RNNs *without unrolling*, and creates a stable $\mathcal{O}(1)$ gradient path for long-range associations.

Indeed, Transformers solved time-parallelism and credit assignment in the same way (Vaswani et al., 2017), and have since revolutionized sequence modeling (Brown et al., 2020). However, Transformers do not possess a compressed memory of the past in the way RNNs or human brains do (Gu, 2025; Kandel, 2000). Instead, Transformers store the *entire history* of past token representations and attend to *all of them* when processing each new token. As a result, their memory size grows with sequence length, leading to prohibitive computational costs for unbounded sequences, such as a human lifetime of experience (Tay et al., 2022). Sliding-window transformers mitigate this issue by storing only the most recent tokens, but have the severe drawback that they lose access to information before the context window (Dai et al., 2019). In contrast, no known biological intelligence operates in this manner—accessing its entire experiential history for every new decision—but instead constructs a temporally compressed abstraction of past experience, like an RNN (Bennett, 2023).

Linear attention RNN models also exhibit time-parallel training and relatively stable credit assignment, while maintaining a fixed memory size (Katharopoulos et al., 2020; Gu et al., 2021; Gu & Dao, 2023; Dao & Gu, 2024). But, because their transition function is linear, the class of functions they can represent is fundamentally constrained (Merrill et al., 2026), which can lead to failure on important sequential tasks such as state tracking (Merrill et al., 2024; Liu et al., 2025).

SMT aims to combine the best of all worlds: time-parallel training, stable $\mathcal{O}(1)$ long-range credit assignment, fixed-memory inference, and maximal expressivity via nonlinear dynamics. Our results confirm that, on language modeling and pixel sequence modeling tasks, SMT outperforms BPTT in learning long-range dependencies while requiring less sequential computation. SMT should primarily be used for pretraining RNNs, followed by some lightweight post-training to mitigate drift from the teacher memory trajectories and adapt to specific downstream tasks. In fact, post-training is necessary to go beyond the limitations of the teacher encoder (Merrill & Sabharwal, 2023). Beyond its role as a training approach for RNNs, SMT can also be seen as a new method for learning representations (mappings from data to latent variables) and for learning world models (transitions from state at time t to state at time $t + 1$).

2 METHODS

2.1 BACKGROUND

Causal Conditional Sequence Modeling Let $\mathbf{x} = [x_0, \ldots, x_T]$ and $\mathbf{y} = [y_0, \ldots, y_T]$ denote input and output sequences. The objective is to learn a model of the conditional distribution $p(\mathbf{y} \mid \mathbf{x})$. We assume each output y_t depends only on $x_0, \ldots, x_t$. Formally we model this distribution with $\prod_{t=0}^{T} p_\theta(y_t \mid \mathbf{x}_{\leq t})$. Autoregressive sequence modeling is a special case when $x_t = y_{t-1}$.

Recurrent Neural Networks (RNNs) An RNN models this problem using a fixed-size latent state, m_t, that summarizes past inputs. At each timestep, this state is updated according to:

$$m_{t+1} = f_\theta(m_t, x_{t+1}) \tag{1}$$

where f_θ is the transition function. The predicted output token distribution is then $p_\theta(y_t \mid \mathbf{x}_{\leq t}) = \text{softmax}(g_\theta(m_t))$, where g_θ is the readout function. Ideally, m_t "remembers" important information from the past and intentionally "forgets" unimportant information, i.e., m_t is a *memory*.

Backpropagation Through Time (BPTT) Traditionally, RNNs are trained with BPTT (Rumelhart et al., 1986; Werbos, 1990). In the forward pass, f_θ is recurrently unrolled over the sequence. The input sequence x_t is provided via teacher forcing, while the memory sequence m_t is generated by the RNN's transition and is used to compute the output predictions. Conceptually, the computation graph takes the form:

$$m_t = f_\theta(\ldots f_\theta(f_\theta(m_\emptyset, x_0), x_1), \ldots, x_t) \quad \text{with} \quad m_\emptyset = \mathbf{0}$$

Gradients are then computed end-to-end on this unrolled computation graph, propagating from the output prediction losses backward through the trajectory of the nonlinear dynamical system. Thus, this gradient credit assignment signal may have to travel for a path length of up to $\mathcal{O}(T)$ steps. Depending on the singular values of the Jacobian of f_θ, gradients may vanish or explode in time.

BPTT has two well-known limitations:

1. Equation 1 is usually implemented with a recurrent for-loop, preventing parallelization (Pascanu et al., 2013a).
2. BPTT often produces unstable high variance gradients (Bengio et al., 1994). When gradients vanish, the RNN experiences a recency bias, hindering the learning of long-range associations (Ravfogel et al., 2019). When gradients explode, the induced dynamical system is chaotic, causing training instability (Pascanu et al., 2013b).

2.2 SUPERVISED MEMORY TRAINING (SMT)

We propose *Supervised Memory Training* (SMT) for pretraining nonlinear RNNs without BPTT. The core idea is to decouple the learning of memory representation from memory dynamics.

Motivation Consider a hypothetical oracle memory-encoding model $\mathcal{Q}$, that takes as input the sequence of tokens up to timestep t, $\mathbf{x}_t^{\text{ctx}} = [x_0, x_1, \ldots, x_t]$, and outputs an effective compressed memory for that timestep $m_t^* = \mathcal{Q}(\mathbf{x}_t^{\text{ctx}})$. This memory retains all information from the past input that is relevant for predicting the future output, $\mathbf{y}_t^{\text{fut}} = [y_t, \ldots, y_T]$, while deliberately discarding unimportant details. For example, the oracle would remember the personalities of characters in a story, but discard details of what they were wearing on a specific day, just as humans do. Running $\mathcal{Q}$ at different points along the sequence produces a corresponding sequence of memory labels $[m_0^*, m_1^*, \ldots, m_T^*]$. With $\mathcal{Q}$, the RNN's problem of learning a temporal update collapses to standard supervised learning on oracle memory transitions labels $(m_t^*, x_{t+1}) \to m_{t+1}^*$. Our key insight is that $\mathcal{Q}$ does *not* have to be a recurrent function over $[x_0, x_1, \ldots, x_t]$, but can instead be represented as a permutation-invariant function over the *set* $\{(x_0, 0), (x_1, 1), \ldots, (x_t, t)\}$ (details in Appendix E).

In practice, SMT approximates $\mathcal{Q}$ by training a time-parallel model (e.g. a Transformer) to compress the past input into a memory representation that a separate decoder model can use to predict the future output. This future-predicting objective operationalizes the notion of a predictive state (Littman & Sutton, 2001).

Formulation Formally, we have the RNN f_θ, bidirectional encoder $\mathcal{E}_\phi$, and causal decoder $\mathcal{D}_\psi$. $\mathcal{E}_\phi$ and $\mathcal{D}_\psi$ are time-parallel Transformer architectures. Given $\mathbf{x} = [x_0, x_1, \ldots, x_T]$ and $\mathbf{y} = [y_0, y_1, \ldots, y_T]$, we consider, for each timestep t, a decomposition into the past and future:

$$\mathbf{x}_t^{\text{ctx}} = [x_0, \ldots, x_t] \qquad \mathbf{x}_t^{\text{fut}} = [x_{t+1}, \ldots, x_T] \qquad \mathbf{y}_t^{\text{fut}} = [y_t, \ldots, y_T]$$

The encoder maps each context to a memory state with $m_t = \mathcal{E}_\phi(\mathbf{x}_t^{\text{ctx}})$. Then, the decoder predicts the future output distribution using the memory of the past and teacher forced *future inputs*:

$$p_{\phi,\psi}(\mathbf{y}_t^{\text{fut}} \mid \mathbf{x}_t^{\text{ctx}}, \mathbf{x}_t^{\text{fut}}) = \prod_{\tau=t}^{T} p_\psi(y_\tau \mid m_t, \mathbf{x}_{t+1:\tau}) = \mathcal{D}_\psi(m_t, \mathbf{x}_t^{\text{fut}})$$

The future decoding loss for timestep t is (CE denotes the sequence level cross-entropy loss):

$$\mathcal{L}_t^{\text{dec}} = \text{CE}\left(\mathbf{y}_t^{\text{fut}}, p_{\phi,\psi}(\mathbf{y}_t^{\text{fut}} \mid \mathbf{x}_t^{\text{ctx}}, \mathbf{x}_t^{\text{fut}})\right) \tag{2}$$

We have the RNN predict the next memory given the current memory and the next input with $\hat{m}_{t+1} = f_\theta(m_t, x_{t+1})$. This prediction is supervised with the next timestep's memory:

$$\mathcal{L}_t^{\text{dyn}} = \text{MSE}(\hat{m}_{t+1}, m_{t+1}) \tag{3}$$

This dynamics loss has two distinct purposes: 1) to train the RNN and 2) to explicitly shape the encoder memory representations to be *Markovian* (i.e. m_{t+1} is predictable solely from (m_t, x_{t+1})).

We add a uniformity loss (Wang & Isola, 2020) to prevent the memory space from collapsing:

$$\mathcal{L}^{\text{unif}} = \log \mathbb{E}_{t_a, t_b \sim [0,\ldots,T]} \exp(-2\|m_{t_a} - m_{t_b}\|_2^2) \tag{4}$$

The full objective is a weighted sum of all three losses:

$$\mathcal{L}^{\text{smt}} = \lambda_{\text{dec}} \mathbb{E}_t\left[\mathcal{L}_t^{\text{dec}}\right] + \lambda_{\text{dyn}} \mathbb{E}_t\left[\mathcal{L}_t^{\text{dyn}}\right] + \lambda_{\text{unif}} \mathcal{L}^{\text{unif}} \tag{5}$$

where the λ terms control the trade-off between memory representation, dynamics, and collapse.

Practice Theoretically, it should be enough to train $\mathcal{E}_\phi$ and $\mathcal{D}_\psi$ with only $\mathcal{L}^{\text{dec}}$, and separately train f_θ with only $\mathcal{L}^{\text{dyn}}$ (proof in Appendix F). However, in practice we find it beneficial to jointly train all models in one stage with $\mathcal{L}^{\text{smt}}$, since that explicitly optimizes m_t to be Markovian, and provides additional temporal credit propagation benefits described in Section 3.5.

For experiments, we truncate $\mathbf{x}_t^{\text{ctx}}$ to a context length T_c and $\mathbf{y}_t^{\text{fut}}$ to a future length T_f. For computational efficiency, we estimate the expectation in $\mathcal{L}^{\text{smt}}$ by randomly sampling a single timestep t, rather than computing all timesteps in the sequence. This yields SMT a smaller training memory footprint than BPTT: $\mathcal{O}(M + T)$ instead of $\mathcal{O}(MT)$, where M is the memory size.

Properties of SMT In SMT, the encoder model constructs appropriate memory representations of the past, while the RNN is responsible for learning the now much simpler task of updating that memory in one-step, thereby decoupling memory representation from memory dynamics. In contrast, under BPTT training, the RNN must learn both tasks simultaneously. Since the memory labels are acquired with a "teacher" encoder-decoder pair, SMT inherits all of its properties, such as time-parallelism, $\mathcal{O}(1)$ credit path for long-range associations, and gradient stability.

2.3 DAgger Memory Training (DMT)

After SMT, the RNN achieves low one-step error in predicting $(m_t, x_{t+1}) \to m_{t+1}$ when m_t comes from the encoder. However, at evaluation time, the model is unrolled autoregressively, using its own predicted memories rather than the encoder memories as input. This train–test mismatch causes small prediction errors to accumulate over time, leading to a growing drift between the RNN-generated memory trajectory $[\hat{m}_0, \ldots, \hat{m}_T]$ and the encoder trajectory $[m_0, \ldots, m_T]$, even with teacher forced input tokens. This drift is quantified as $\delta_t = \text{MSE}(\hat{m}_t, m_t)$.

We introduce *DAgger Memory Training* (DMT), a finetuning phase that corrects this drift via *on-policy imitation learning* (Ross et al., 2011). By exposing

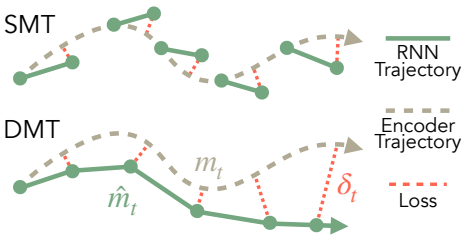

Figure 2: **SMT vs DMT.** SMT trains the RNN with behavior cloning on the encoder-generated memory states (off-policy imitation learning). DMT unrolls the RNN with its own memory states and then imitates the encoder trajectory (on-policy imitation learning).

the RNN to its own induced memory state distribution, DMT trains the RNN to autocorrect its errors to stay aligned with the encoder trajectory (Figure 2).

Concretely, given $\mathbf{x}$, we first compute the encoder trajectory $[m_0, \ldots, m_T]$ using only $\mathcal{E}_\phi$ and then the RNN trajectory $[\hat{m}_0, \ldots, \hat{m}_T]$ using f_θ. Instead of training on SMT labels $(m_t, x_{t+1}) \to m_{t+1}$, we train on DMT labels $(\hat{m}_t, x_{t+1}) \to m_{t+1}$. Equivalently, the training loss is:

$$\mathcal{L}^{\text{dmt}} = \mathbb{E}_t \left[\text{MSE}(\hat{m}_t, m_t) \right] \tag{6}$$

During DMT, we freeze the encoder and decoder and only train the RNN with a small learning rate. Note that DMT unrolls the RNN memories, but still uses teacher forced x_t inputs. Although DMT unrolls the RNN and gradients may optionally propagate through time, its objective is fundamentally different than standard BPTT, since long-range credit is already assigned in the encoder memory labels, m_t. DMT is not time-parallel. That said, DMT should primarily be viewed as a lightweight fine-tuning phase following SMT. Table 1 shows the resource requirements for the different methods.

Table 1: **Resource requirements.** T is token sequence length. T_c is SMT encoder context length. For RNNs, M is the memory state size. We ignore $\log$ terms for simplicity. LA denotes linear attention (in its parallel and recurrent form). Complexity classes are from Merrill et al. (2026).

	Training (T-Length Sequence)				Inference (One-Step)		Complexity Class
Method	Memory	Compute	**Sequential Operations**	**Credit Path Length**	Memory	Compute	
Transformer	$\mathcal{O}(T)$	$\mathcal{O}(T^2)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(T)$	$\mathcal{O}(T)$	TC^0
LA (parallel)	$\mathcal{O}(T)$	$\mathcal{O}(T^2)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(M)$	$\mathcal{O}(1)$	PNC^1
LA (recurrent)	$\mathcal{O}(M)$	$\mathcal{O}(T)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(M)$	$\mathcal{O}(1)$	PNC^1
BPTT (RNN)	$\mathcal{O}(MT)$	$\mathcal{O}(T)$	$\mathcal{O}(T)$	$\mathcal{O}(T)$	$\mathcal{O}(M)$	$\mathcal{O}(1)$	L/P
SMT (RNN)	$\mathcal{O}(M + T_c)$	$\mathcal{O}(T_c^2)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(M)$	$\mathcal{O}(1)$	L/P
DMT (RNN)	$\mathcal{O}(MT)$	$\mathcal{O}(T_c^2 T)$	$\mathcal{O}(T)$	$\mathcal{O}(1)$	$\mathcal{O}(M)$	$\mathcal{O}(1)$	L/P

3 Experiments

We study the properties of SMT and compare against BPTT, the standard RNN training algorithm. We restrict our analysis to nonlinear RNNs, the primarily setting BPTT is applied. Transformers and linear RNNs are excluded as they are qualitatively distinct model classes (Merrill et al., 2026; Gu, 2025).

"BPTT RNN" denotes the BPTT baseline. "SMT Encoder*" generates memories m_t with the SMT-trained encoder and predicts next tokens using the decoder. This method is essentially a Transformer baseline with the same memory bottleneck as our RNNs. Since it serves as the teacher during SMT and DMT, it provides a reference upper bound on RNN performance. "SMT→DMT RNN" denotes the RNN pretrained with SMT and finetuned with DMT, which constitutes our full method.

Architectures We use RNN architectures based on a Transformer, MLP, GRU (Cho et al., 2014), and LSTM (Hochreiter & Schmidhuber, 1997) backbone.

Datasets We consider character-level language modeling on TinyStories (Eldan & Li, 2023) as a naturalistic task requiring long-range memory (Sutskever et al., 2011). As a more challenging problem, we test our method on raster-scan order pixel sequence modeling of sparse images from MNIST (LeCun & Cortes, 1998) and Sketchy (Sangkloy et al., 2016). This is a hard problem for RNNs (Lamb et al., 2016; Van Den Oord et al., 2016). Imagine you are an ant traversing an image pixel by pixel, row by row. When you see a new white pixel, in order to recognize the shape and slope of the stroke it belongs to, you must remember the white pixels you saw in the previous rows, which may be hundreds of timesteps ago, buried among black pixels. RNNs must achieve this with finite memory, meaning no direct attention to earlier pixels, and thus forcing long-range memory to emerge. We term this "Attneave's task", based on classic work from perceptual psychology (Attneave, 1954). More details on architectures, datasets, and experiments are in Appendix C.

3.1 Synthetic Task Experiments

We first evaluate BPTT and SMT on synthetic tasks designed to isolate and probe specific properties of the training algorithm. The RNN architecture with the Transformer backbone is used for these experiments. For these synthetic experiments, we set $T_c = T_f = T$ and train all timesteps in the $\mathcal{L}^{\text{smt}}$ expected value. Our tasks include the following (details of tasks are in Appendix C.2.1):

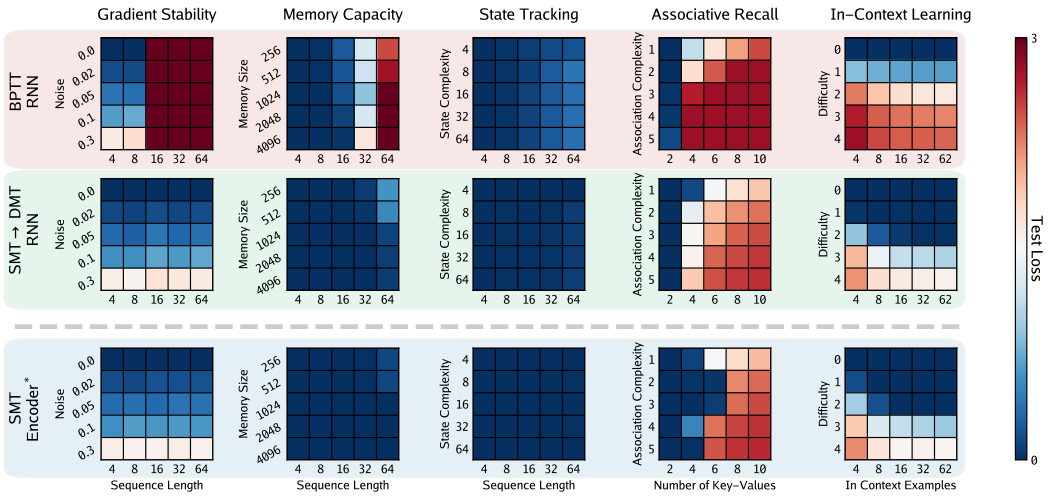

Figure 3: **Synthetic Task Experiments.** We evaluate BPTT, SMT, and SMT→DMT using five synthetic tasks with various settings to probe different properties of the algorithms. * signifies that the SMT Encoder is the teacher Transformer (*not* an RNN) and is used only as a reference. Across all tasks and task settings, SMT→DMT outperforms BPTT, signaling that SMT has better gradient properties, memory utilization, state tracking, associative recall, and in-context learning than BPTT.

1. Retrieval to test **Gradient Stability** (sweep sequence length and noise level).
2. String Copy to test **Memory Capacity** (sweep sequence length and memory state size).
3. Stack Operations to test **State Tracking** (sweep sequence length and state complexity).
4. Keys-Values to test **Associative Recall** (sweep number of and complexity of associations).
5. Modular Arithmetic to test **In-Context Learning** (sweep difficulty and number of examples).

Figure 3 shows that SMT→DMT outperforms BPTT in all settings of all tasks. BPTT struggles to learn as sequences get longer, even when the task is simple, e.g. retrieval. It also struggles to utilize memory capacity fully, do associative recall, and perform in-context learning, all of which require solving long-range credit assignment. In contrast, SMT seems agnostic to the sequence length, and is able to solve all of the harder credit assignment problems except associative recall. We attribute these differences to BPTT's $\mathcal{O}(T)$ credit path length, compared to SMT's $\mathcal{O}(1)$. Further analysis in Section 3.5 confirms the difference in gradient stability in both methods.

3.2 Attneave's Pixel Sequence Modeling

We now evaluate on Attneave's tasks. Figure 4 shows the stark difference between MNIST samples generated by RNNs trained with BPTT and SMT→DMT. Figure 19 shows images generated by an SMT→DMT RNN trained on Sketchy.

Along with the synthetic experiments, these results confirm that SMT doesn't suffer from a recency bias like BPTT, allowing it to properly attribute credit across long sequences.

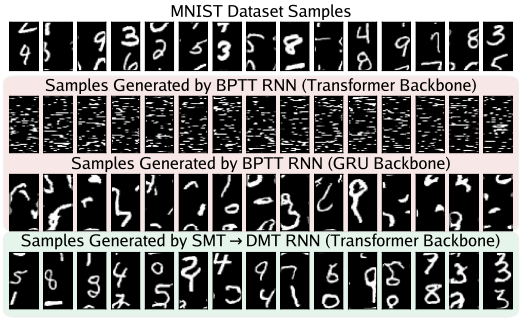

Figure 4: **Attneave's MNIST Generation.** BPTT fails to effectively capture the long-range dependencies required for pixel sequence modeling, even with a GRU. SMT→DMT captures these dependencies with a non-gated RNN architecture. More samples are in Appendix Figure 21.

3.3 Sequential Compute and Data

We now evaluate BPTT and SMT across real domains and various RNN architectures. Each method is allowed N optimization steps on token batches of shape $B \times T$ (number of sequences $\times$ sequence length). We sweep N, B, and T for each method to profile how much sequential compute and data each method uses to achieve a target performance. Sequential compute, measured in sequential FLOPs, is a metric proportional to the amount of inherently serial steps required to do the computation ($\sim$ time it would take on an infinitely parallel computer). Sequential

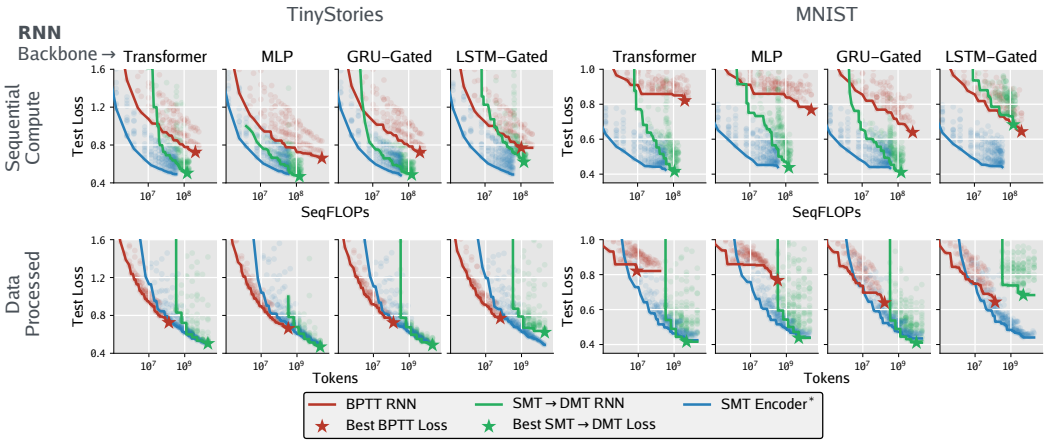

Figure 5: **Sequential Compute and Data Efficiency.** We sweep training hyperparameters for BPTT, SMT, and SMT→DMT and plot the resulting runs' performance along sequential compute (SeqFLOPs) used and data processed (Tokens), across different RNN architectures and datasets. Runs are capped at one day on an H200 GPU. * signifies that the SMT Encoder is the teacher Transformer (*not* an RNN) and is used only as a reference. Generally, SMT and SMT→DMT are more efficient than BPTT in sequential compute, and around the same or better efficiency in data.

compute is a useful quantity because modern hardware is highly parallel, making it the primary constraint in large-scale model training (Hooker, 2021). Data is measured by the number of tokens processed by the model during training. We elaborate on how sequential FLOPs and data is calculated for each method in Appendix B.

Figure 5 shows the results. In **sequential compute**, SMT Encoder and SMT→DMT RNN are significantly more efficient than BPTT with the Transformer and MLP backbones. In **data**, SMT Encoder and SMT→DMT RNN has approximately the same data efficiency as BPTT with the Transformer and MLP backbones on TinyStories. However on MNIST, SMT Encoder and SMT→DMT RNN shows significantly better data efficiency. This result is explained by the short vs long range memory information requirements of natural language (Fang et al., 2025) vs pixel sequence modeling (Tay et al., 2020).

3.4 Scaling Laws

We evaluate the scaling behavior of SMT→DMT along three axes: context length, memory state size, and model parameter count. For the first two, we logarithmically sweep T_c and the number of memory tokens in the Transformer-based RNN (Figure 10). For model scaling, we vary the width and depth of the RNN, encoder, and decoder. We use the TinyStories domain for these experiments.

Figure 6 shows that SMT→DMT exhibits smooth, predictable performance improvements with larger context length and bigger memory state size. Together with the previous experiments, these results reaffirm that SMT effectively leverages long contexts and large memory states. Figure 7 presents the parameter scaling results. The SMT encoder follows a standard power-law-like scaling trend. The

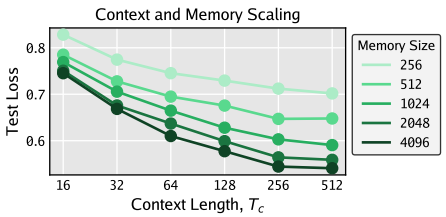

Figure 6: **Scaling Context and Memory.** SMT→DMT shows smooth performance improvements as you increase the context length and the memory size in TinyStories.

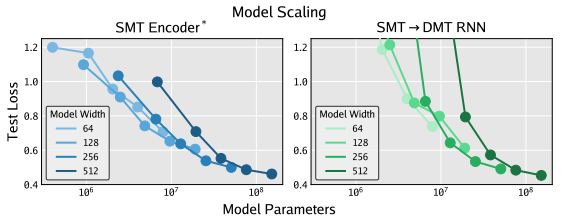

Figure 7: **Scaling Model Size.** Sweeping the width and depth of the RNN and teacher shows smooth performance improvements in TinyStories. The RNN imitates the teacher performance better at larger scale.

SMT→DMT RNN also improves smoothly with scale, albeit with a differently shaped scaling curve. Interestingly, the RNN appears to more closely match the encoder's performance at larger scales.

3.5 Ablations and Analysis

Predictive State and Detached RNN The impact of the predictive state objective (Equation 2) is evaluated by sweeping the future length T_f, while keeping T_c large enough to see the whole sequence. The impact of the dynamics objective (Equation 3) on memory representation is tested by detaching the model computation graph with stop grads at two locations such that the gradients from $\mathcal{L}_t^{\text{dyn}}$ flow to the RNN, but *not* the encoder (**detached**); the non-detached SMT baseline is referred to as **joint**. This ablation isolates the contribution of explicitly training m_t to be a Markovian representation.

Figure 8 shows the results on the needle retrieval task. To solve the task, and thus have proper credit assignment, SMT requires either large enough T_f *or* joint training. When T_f is large enough, there is a $\mathcal{O}(1)$ credit path length between the needle and the answer at all timesteps. Interestingly, when T_f is small, there exists no credit path to learn early timestep memories, yet joint training still learns effectively, even when $T_f = 1$. Credit must be propagating through the RNN dynamics from m_T to m_{T-1}, and so on, to m_0. But because the RNN is never unrolled, there is no computation graph for credit to propagate directly. The only explanation is that **credit is being amortized into gradient optimization steps**. Each optimization step sends information from m_t to m_{t-1} through f_θ; T such gradients steps sends information T steps back in the sequence. This implies that solving T sequence length credit assignment task when $T_f = 1$, requires at least T *gradient optimization steps*. This credit amortization phenomenon is reminiscent of value bootstrapping in RL (Sutton et al., 1998).

Gradient Properties of BPTT and SMT The fundamental difference between BPTT and SMT in long-range credit assignment is dictated by their gradients. Figure 9 shows the gradient magnitude of m_t, $\|\frac{\partial L}{\partial m_t}\|$, at different t for both methods with different model weight initializations on the needle retrieval task. In BPTT, gradients vanish or explode over time, due to BPTT's gradient propagation through recurrent modules. In SMT, gradient magnitudes are independent of t, because the credit path length between tokens is independent of the sequence length. This result explains why SMT does not suffer a recency bias and is able to do stably perform long-horizon credit assignment.

Additional Experiments Appendix Section D provides more results, analysis of the learned RNNs, and comparison with baselines like ParaRNN (Danieli et al., 2025).

4 Related Works

Recurrent Neural Networks (RNNs) RNNs were studied extensively early in AI because their recurrence mechanism resembles biological brains (McCulloch & Pitts, 1943) and can be applied to any sequential task (Elman, 1990). Many different algorithms were proposed for learning, including random guessing (Schmidhuber et al., 2001), evolutionary algorithms (Miller et al., 1989; Angeline et al., 1994; Stanley & Miikkulainen, 2002; Salimans et al., 2017; Sarkar et al., 2025), hebbian learning (Hebb, 1949; Hopfield, 1982; Miconi et al., 2018; Najarro & Risi, 2022), real-time recurrent learning (Williams & Zipser, 1989), and other algorithms (Ollivier et al., 2015; Bengio et al., 2015; Lamb et al., 2016; Bai et al., 2019; Kag & Saligrama, 2021). BPTT is the only widely adopted algorithm (Werbos, 1990).

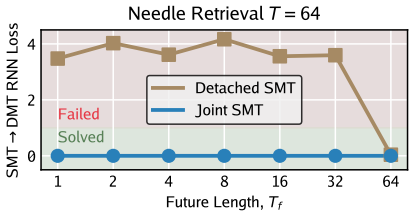

Figure 8: **Joint SMT Ablation.** Here, the task requires credit assignment across T timesteps. When the RNN is detached during SMT, T_f must be large enough to capture the task signal ($T_f = T$). With joint training, SMT solves the task even when T_f is small.

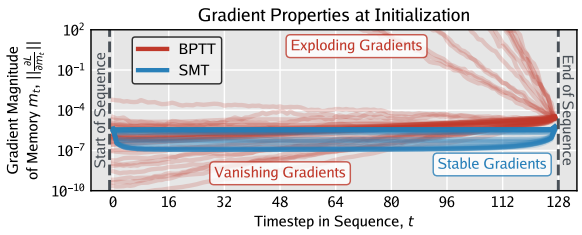

Figure 9: **Gradient Properties of BPTT and SMT.** In the needle retrieval task, the loss is applied at the last timestep. BPTT propagates gradients backward through all timesteps, risking vanishing/exploding gradients for each m_t, depending on the weight initialization. SMT is non-recurrent and has a $\mathcal{O}(1)$ credit path length, making its gradients agnostic to initialization and time-horizon.

However, it has repeatedly been shown that BPTT produces unstable gradients that vanish, explode, or exhibit high variance (Bengio et al., 1994; Hochreiter et al., 2001; Pascanu et al., 2013b; Bai et al., 2018). Several directions address this issue. One direction focuses on architectural modifications, including residual connections (Srivastava et al., 2015; He et al., 2016) and gating mechanisms (Chung et al., 2014), culminating in the development of the LSTM (Hochreiter & Schmidhuber, 1997) and GRU (Cho et al., 2014). A parallel direction addressed gradient instability through orthogonal weight parameterizations to prevent exponential growth or decay across time (Saxe et al., 2013; Arjovsky et al., 2016; Wisdom et al., 2016; Mhammedi et al., 2017; Vorontsov et al., 2017; Helfrich et al., 2018). Others explored external memory (Peshkin et al., 2001; Graves et al., 2014; Hansen, 2017), hierarchical modeling (Hihi & Bengio, 1995; Chung et al., 2016; Wang et al., 2025a; Jolicoeur-Martineau, 2025), other unique directions (Jaeger, 2001; Lukoševičius & Jaeger, 2009; Miller & Hardt, 2018).

Recently, there has been renewed interest in RNNs in the form of linear state space models (Gu et al., 2021; Smith et al., 2022; Gu & Dao, 2023), linear attention models (Katharopoulos et al., 2020; Sun et al., 2023; Dao & Gu, 2024; Yang et al., 2025b;a), and even nonlinear RNN models (Beck et al., 2024; Bulatov et al., 2022; Mishra et al., 2026; Oncescu et al., 2026). Recurrent computation more generally has been reappearing across paradigms including in diffusion (Ho et al., 2020), looped Transformers (Giannou et al., 2023), and reasoning (Hao et al., 2025; Geiping et al., 2025).

Time-Parallel Training Transformers revolutionized sequence modeling (Brown et al., 2020) largely because they have time-parallel training (Vaswani et al., 2017) (unlike prior attention methods (Bahdanau et al., 2016)), which is crucial for leveraging modern hardware (Hillis & Steele Jr, 1986; Hooker, 2021) to scale performance (Kaplan et al., 2020; Hoffmann et al., 2022). Linear RNNs gained popularity (Katharopoulos et al., 2020; Dao & Gu, 2024; Yang et al., 2025b; Feng et al., 2024) after it was realized they can be parallelized with the associative scan algorithm (Blelloch, 1990; Martin & Cundy, 2018; Wang et al., 2020).

A recent line of work attempts to parallelize nonlinear RNNs as well (Lim et al., 2024; Danieli et al., 2023). Rather than computing $[m_0, \ldots, m_T]$ with $m_{t+1} = f_\theta(m_t)$, they formulate the forward pass as an iterative optimization procedure. Starting with an initial guess $[m_0^0, \ldots, m_T^0]$, they construct a system of T equations, $\{m_{t+1} - f_\theta(m_t) = 0\}_{t=0}^T$, and solve this system with Newton's method (Ortega & Rheinboldt, 2000). Many works have further built on this approach (Danieli et al., 2025; Gonzalez et al., 2024). Although appealing, this approach approximates BPTT and hence will suffer from its $\mathcal{O}(T)$ credit path length and corresponding gradient instability, along with the added convergence worries of Newton's method (Gonzalez et al., 2025). In contrast, SMT uses an encoder to train m_t to be a predictive state while satisfying $m_{t+1} \approx f_\theta(m_t)$ and providing an $\mathcal{O}(1)$ credit path length.

Additional related work can be found in Appendix Section A.

5 Discussion

In SMT, the teacher model is time-parallel, and is thus constrained in expressivity (Merrill & Sabharwal, 2023), implying that SMT-trained RNNs may suffer the same problem. Therefore, BPTT finetuning might be required to achieve expressivity beyond the teacher. Additionally, while SMT is useful for learning how to encode sequences, it is not necessarily to be used for learning reasoning since intermediate steps are not supervised. The same limitation applies to Transformers yet post-training allows them to effectively solve longer-horizon tasks than the training horizon; the same might be true for SMT-trained RNNs.

The current SMT variant computes and trains only a single m_t within a sequence. We found that training on all memories $[m_0, \ldots, m_T]$ offered no improvement in our settings, but this may not hold at larger scales. After SMT, the RNN experiences drift away from the teacher memory trajectory. DMT provides one solution but is not time-parallel; however, it may be parallelized via DEER (Lim et al., 2024).

RNNs have the promise of solving problems that extend over unbounded horizons, such as the entire lifetime of an agent. However, training methods for RNN have been hindered by the inability of BPTT to assign credit effectively over such a long horizon. Our method circumvents the credit assignment issue with an $\mathcal{O}(1)$ connection path. In the regimes we studied, this effectively allows for learning memories that are only useful many steps later, an ability that is crucial for lifelong learning.

AI USE STATEMENT

In this work, we used generative AI tools only as simple auto-complete during the coding of the project. We also used it sparingly during writing to assist with phrasing of sentences for clarity. However, all the writing was originally generated by us, and extensively polished by us. We have not used generative AI tools for full sentence generation, agentic coding, or anything else. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work develops a core machine learning algorithm and does not raise specific ethical concerns requiring an ethics statement.

REPRODUCIBILITY STATEMENT

We plan to host our code for this project on GitHub, allowing users to reproduce the algorithm and plots in this paper.

REFERENCES

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. arXiv preprint arXiv:2211.15661, 2022.

Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In Proceedings of the April 18-20, 1967, spring joint computer conference, pp. 483–485, 1967.

Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. IEEE transactions on Neural Networks, 5(1):54–65, 1994.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In International conference on machine learning, pp. 1120–1128. PMLR, 2016.

Fred Attneave. Some informational aspects of visual perception. Psychological review, 61(3):183, 1954.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016. URL https://arxiv.org/abs/1409.0473.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271, 2018.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. Advances in neural information processing systems, 32, 2019.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. Advances in Neural Information Processing Systems, 37:107547–107603, 2024.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks, 2015. URL https://arxiv.org/abs/1506.03099.

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.

Max S Bennett. A brief history of intelligence: evolution, AI, and the five breakthroughs that made our brains. HarperCollins, 2023.

Guy E Blelloch. Prefix sums and their applications. 1990.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. Advances in Neural Information Processing Systems, 35:11079–11091, 2022.

Yingfa Chen, Zhen Leng Thai, Zihan Zhou, Zhu Zhang, Xingyu Shen, Shuo Wang, Chaojun Xiao, Xu Han, and Zhiyuan Liu. Hybrid linear attention done right: Efficient distillation and effective architectures for extremely long contexts, 2026. URL https://arxiv.org/abs/2601.22156.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In Proceedings of SSST-8, eighth workshop on syntax, semantics and structure in statistical translation, pp. 103–111, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. arXiv preprint arXiv:1609.01704, 2016.

Stephen A Cook. A taxonomy of problems with fast parallel algorithms. Information and control, 64 (1-3):2–22, 1985.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In Proceedings of the 57th annual meeting of the association for computational linguistics, pp. 2978–2988, 2019.

Federico Danieli, Miguel Sarabia, Xavier Suau Cuadros, Pau Rodriguez, and Luca Zappella. Deeppcr: Parallelizing sequential operations in neural networks. Advances in Neural Information Processing Systems, 36:47598–47625, 2023.

Federico Danieli, Pau Rodriguez, Miguel Sarabia, Xavier Suau, and Luca Zappella. Pararnn: Unlocking parallel training of nonlinear rnns for large language models, 2025. URL https://arxiv.org/abs/2510.21450.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. arXiv preprint arXiv:2405.21060, 2024.

Carlton Downey, Ahmed Hefny, and Geoffrey Gordon. Practical learning of predictive state representations. arXiv preprint arXiv:1702.04121, 2017a.

Carlton Downey, Ahmed Hefny, Boyue Li, Byron Boots, and Geoffrey Gordon. Predictive state recurrent neural networks, 2017b. URL https://arxiv.org/abs/1705.09353.

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? arXiv preprint arXiv:2305.07759, 2023.

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

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. arXiv preprint arXiv:2002.09402, 2020.

Lizhe Fang, Yifei Wang, Zhaoyang Liu, Chenheng Zhang, Stefanie Jegelka, Jinyang Gao, Bolin Ding, and Yisen Wang. What is wrong with perplexity for long-context language modeling?, 2025. URL https://arxiv.org/abs/2410.23771.

Leo Feng, Frederick Tung, Mohamed Osama Ahmed, Yoshua Bengio, and Hossein Hajimirsadeghi. Were rnns all we needed? arXiv preprint arXiv:2410.01201, 2024.

Yulu Gan and Phillip Isola. Neural thickets: Diverse task experts are dense around pretrained weights. arXiv preprint arXiv:2603.12228, 2026.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. arXiv preprint arXiv:2502.05171, 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In International Conference on Machine Learning, pp. 11398–11442. PMLR, 2023.

Xavier Gonzalez, Andrew Warrington, Jimmy T Smith, and Scott W Linderman. Towards scalable and stable parallelization of nonlinear rnns. Advances in Neural Information Processing Systems, 37:5817–5849, 2024.

Xavier Gonzalez, Leo Kozachkov, David M Zoltowski, Kenneth L Clarkson, and Scott W Linderman. Predictability enables parallelization of nonlinear state space models. arXiv preprint arXiv:2508.16817, 2025.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. arXiv preprint arXiv:1410.5401, 2014.

Albert Gu. On the tradeoffs of state space models and transformers, 2025. URL https://goombalab.github.io/blog/2025/tradeoffs/.

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. arXiv preprint arXiv:2111.00396, 2021.

Steven Stenberg Hansen. Long timescale credit assignment in neuralnetworks with external memory, 2017. URL https://arxiv.org/abs/1701.03866.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

Lukas Hauzenberger, Niklas Schmidinger, Thomas Schmied, Anamaria-Roberta Hartl, David Stap, Pieter-Jan Hoedt, Maximilian Beck, Sebastian Böck, Günter Klambauer, and Sepp Hochreiter. Effective distillation to hybrid xlstm architectures, 2026. URL https://arxiv.org/abs/2603.15590.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

Donald Olding Hebb. The organization of behavior: A neuropsychological theory. Psychology press, 1949.

Ahmed Hefny, Zita Marinho, Wen Sun, Siddhartha Srinivasa, and Geoffrey Gordon. Recurrent predictive state policy networks, 2018. URL https://arxiv.org/abs/1803.01489.

Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. In International Conference on Machine Learning, pp. 1969–1978. PMLR, 2018.

Salah Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. Advances in neural information processing systems, 8, 1995.

W Daniel Hillis and Guy L Steele Jr. Data parallel algorithms. Communications of the ACM, 29 (12):1170–1183, 1986.

Geoffrey E Hinton and James A Anderson. Parallel models of associative memory: updated edition. Psychology press, 2014.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. Advances in neural information processing systems, 33:6840–6851, 2020.

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

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. arXiv preprint arXiv:2203.15556, 10, 2022.

Sara Hooker. The hardware lottery. Communications of the ACM, 64(12):58–65, 2021.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences, 79(8):2554–2558, 1982.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

Edward S Hu, Kwangjun Ahn, Qinghua Liu, Haoran Xu, Manan Tomar, Ada Langford, Dinesh Jayaraman, Alex Lamb, and John Langford. Learning to achieve goals with belief state transformers. arXiv preprint arXiv:2410.23506, 2024.

Marcus Hutter. Universal artificial intelligence: Sequential decisions based on algorithmic probability, volume 300. Springer, 2005.

Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German national research center for information technology gmd technical report, 148(34):13, 2001.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. Artificial intelligence, 101(1-2):99–134, 1998.

Anil Kag and Venkatesh Saligrama. Training recurrent neural networks via forward propagation through time. In International Conference on Machine Learning, pp. 5189–5200. PMLR, 2021.

Eric R Kandel. Principles of neural science, 2000.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A. Smith. Finetuning pretrained transformers into rnns, 2021. URL https://arxiv.org/abs/2103.13076.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In International conference on machine learning, pp. 5156–5165. PMLR, 2020.

Louis Kirsch, James Harrison, Jascha Sohl-Dickstein, and Luke Metz. General-purpose in-context learning by meta-learning transformers. arXiv preprint arXiv:2212.04458, 2022.

Andrei Nikolaevic Kolmogorov. Three approaches to the quantitative definition of information. International journal of computer mathematics, 2(1-4):157–168, 1968.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. Advances in neural information processing systems, 29, 2016.

Yann LeCun and Corinna Cortes. The MNIST database of handwritten digits, 1998. URL http://yann.lecun.com/exdb/mnist/.

Belinda Z. Li, Zifan Carl Guo, and Jacob Andreas. (how) do language models track state?, 2025a. URL https://arxiv.org/abs/2503.02854.

Qinyu Li, Yee Whye Teh, and Razvan Pascanu. Noprop: Training neural networks without full back-propagation or full forward-propagation. arXiv preprint arXiv:2503.24322, 2025b.

Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah Kasim. Parallelizing non-linear sequential models over the sequence length, 2024. URL https://arxiv.org/abs/2309.12252.

Michael Littman and Richard S Sutton. Predictive representations of state. Advances in neural information processing systems, 14, 2001.

Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. arXiv preprint arXiv:2210.10749, 2022.

Yuxi Liu, Konpat Preechakul, Kananart Kuwaranancharoen, and Yutong Bai. The serial scaling hypothesis. arXiv preprint arXiv:2507.12549, 2025.

Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. Computer science review, 3(3):127–149, 2009.

Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length, 2018. URL https://arxiv.org/abs/1709.04057.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. Transactions of the Association for Computational Linguistics, 11:531–545, 2023.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought, 2024. URL https://arxiv.org/abs/2310.07923.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. arXiv preprint arXiv:2404.08819, 2024.

William Merrill, Hongjian Jiang, Yanhong Li, and Ashish Sabharwal. Why are linear rnns more parallelizable? arXiv preprint arXiv:2603.03612, 2026.

Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In International Conference on Machine Learning, pp. 2401–2409. PMLR, 2017.

Thomas Miconi, Jeff Clune, and Kenneth O. Stanley. Differentiable plasticity: training plastic neural networks with backpropagation, 2018. URL https://arxiv.org/abs/1804.02464.

Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In ICGA, volume 89, pp. 379–384, 1989.

John Miller and Moritz Hardt. Stable recurrent models. arXiv preprint arXiv:1805.10369, 2018.

Marvin Minsky. Steps toward artificial intelligence. Proceedings of the IRE, 49(1):8–30, 1961.

Mayank Mishra, Shawn Tan, Ion Stoica, Joseph Gonzalez, and Tri Dao. M2 rnn: Non-linear rnns with matrix-valued states for scalable language modeling. arXiv preprint arXiv:2603.14360, 2026.

Abhinav Moudgil, Ningyuan Huang, Eeshan Gunesh Dhekane, Pau Rodríguez, Luca Zappella, and Federico Danieli. Attention to mamba: A recipe for cross-architecture distillation. arXiv preprint arXiv:2604.14191, 2026.

Elias Najarro and Sebastian Risi. Meta-learning through hebbian plasticity in random networks, 2022. URL https://arxiv.org/abs/2007.02686.

Yann Ollivier, Corentin Tallec, and Guillaume Charpiat. Training recurrent networks online without backtracking. arXiv preprint arXiv:1507.07680, 2015.

Costin-Andrei Oncescu, Depen Morwani, Samy Jelassi, Alexandru Meterez, Mujin Kwun, and Sham Kakade. The recurrent transformer: Greater effective depth and efficient decoding. arXiv preprint arXiv:2604.21215, 2026.

James M Ortega and Werner C Rheinboldt. Iterative solution of nonlinear equations in several variables. SIAM, 2000.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. arXiv preprint arXiv:1312.6026, 2013a.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In International conference on machine learning, pp. 1310–1318. Pmlr, 2013b.

Leonid Peshkin, Nicolas Meuleau, and Leslie Kaelbling. Learning policies with external memory. arXiv preprint cs/0103003, 2001.

Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. arXiv preprint arXiv:2108.12409, 2021.

Shauli Ravfogel, Yoav Goldberg, and Tal Linzen. Studying the inductive biases of rnns with synthetic variations of natural languages. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 3532–3542, 2019.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011. URL https://arxiv.org/abs/1011.0686.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. nature, 323(6088):533–536, 1986.

Walter L Ruzzo. On uniform circuit complexity. Journal of Computer and System Sciences, 22(3): 365–383, 1981.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864, 2017.

Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: learning to retrieve badly drawn bunnies. Acm Transactions on Graphics (TOG), 35(4):1–12, 2016.

Bidipta Sarkar, Mattie Fellows, Juan Agustin Duque, Alistair Letcher, Antonio León Villares, Anya Sims, Clarisse Wibault, Dmitry Samsonov, Dylan Cope, Jarek Liesen, et al. Evolution strategies at the hyperscale. arXiv preprint arXiv:2511.16652, 2025.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120, 2013.

Jürgen Schmidhuber, Sepp Hochreiter, and Yoshua Bengio. Evaluating benchmark problems by random guessing. A Field Guide to Dynamical Recurrent Networks, pp. 231–235, 2001.

Makoto Shing, Masanori Koyama, and Takuya Akiba. Diffusionblocks: Block-wise neural network training via diffusion interpretation. arXiv preprint arXiv:2506.14202, 2025.

Satinder Singh, Michael James, and Matthew Rudary. Predictive state representations: A new theory for modeling dynamical systems. arXiv preprint arXiv:1207.4167, 2012.

Satinder P Singh, Michael L Littman, Nicholas K Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In Proceedings of the 20th International Conference on Machine Learning (ICML-03), pp. 712–719, 2003.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. arXiv preprint arXiv:2208.04933, 2022.

Ray J Solomonoff. A formal theory of inductive inference. part i. Information and control, 7(1):1–22, 1964.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. arXiv preprint arXiv:1505.00387, 2015.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2):99–127, 2002.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. Neurocomputing, 568:127063, 2024.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. arXiv preprint arXiv:2307.08621, 2023.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In Proceedings of the 28th international conference on machine learning (ICML-11), pp. 1017–1024, 2011.

Richard S Sutton, Andrew G Barto, et al. Reinforcement learning: An introduction, volume 1. MIT press Cambridge, 1998.

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, 2020. URL https://arxiv.org/abs/2011.04006.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. ACM Computing Surveys, 55(6):1–28, 2022.

Jayden Teoh, Manan Tomar, Kwangjun Ahn, Edward S. Hu, Pratyusha Sharma, Riashat Islam, Alex Lamb, and John Langford. Next-latent prediction transformers learn compact world models, 2025. URL https://arxiv.org/abs/2511.05963.

Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In International conference on machine learning, pp. 1747–1756. PMLR, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

Arun Venkatraman, Nicholas Rhinehart, Wen Sun, Lerrel Pinto, Martial Hebert, Byron Boots, Kris M. Kitani, and J. Andrew Bagnell. Predictive-state decoders: Encoding the future into recurrent networks, 2017. URL https://arxiv.org/abs/1709.08520.

Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In International conference on machine learning, pp. 3570–3578. PMLR, 2017.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025a. URL https://arxiv.org/abs/2506.21734.

Junxiong Wang, Daniele Paliotta, Avner May, Alexander M. Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models, 2025b. URL https://arxiv.org/abs/2408.15237.

Shang Wang, Yifan Bai, and Gennady Pekhimenko. Bppsa: Scaling back-propagation by parallel scan algorithm, 2020. URL https://arxiv.org/abs/1907.10134.

Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In International conference on machine learning, pp. 9929–9939. PMLR, 2020.

Paul J Werbos. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10):1550–1560, 1990.

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

David J Willshaw, O Peter Buneman, and Hugh Christopher Longuet-Higgins. Non-holographic associative memory. Nature, 222(5197):960–962, 1969.

Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. Advances in neural information processing systems, 29, 2016.

Qingyang Wu, Zhenzhong Lan, Kun Qian, Jing Gu, Alborz Geramifard, and Zhou Yu. Memformer: A memory-augmented transformer for sequence modeling. In Findings of the association for computational linguistics: AACL-IJCNLP 2022, pp. 308–318, 2022.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025a. URL https://arxiv.org/abs/2412.06464.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length, 2025b. URL https://arxiv.org/abs/2406.06484.

Morris Yau, Sharut Gupta, Valerie Engelmayer, Kazuki Irie, Stefanie Jegelka, and Jacob Andreas. Sequential-parallel duality in prefix scannable models, 2026. URL https://arxiv.org/abs/2506.10918.

A Additional Related Work

Relation with NextLat Related and concurrent to our work, Teoh et al. (2025) introduced Next-Latent Prediction (NextLat), which co-trains an RNN with memory state supervision from a Transformer. With a particular setting of hyperparameters, NextLat resembles DMT (and SMT when the rollout length is 1), though still utilizing BPTT on multi-step rollouts. However, the central focus of NextLat is to regularize the Transformer to learn compact world models, rather than providing a method to train RNNs. Consequently, their analysis focuses primarily on the Transformer. In contrast, our work focuses exclusively on training the RNN in a one-step manner.

Computation Complexity Class of Models A model's architecture determines the problems it can theoretically solve (Hornik et al., 1989). Some tasks are inherently sequential and cannot be efficiently parallelized (Amdahl, 1967); the **circuit depth** of a task is the minimum number of sequential steps required to solve it on an infinitely parallel computer (Ruzzo, 1981; Cook, 1985). Every neural network has a corresponding **sequential depth**—the longest nonlinear computation path from input to output—which bounds the class of problems it can solve (Merrill & Sabharwal, 2024). Models with constant or logarithmic sequential depth per layer, such as Transformers and linear RNNs, are provably limited to tasks with equivalently low circuit depth (Merrill & Sabharwal, 2023; 2024; Yau et al., 2026). While such models succeed on tasks amenable to parallelization (e.g. parity tracking via associative scan (Liu et al., 2022; Li et al., 2025a)), they systematically fail on tasks requiring deep sequential computation (e.g. tracking a chess board (Merrill et al., 2024)). Interestingly, the aspect that makes models parallelizable, limits their performance on harder problems (Merrill & Sabharwal, 2023). Nonlinear RNNs are one of the few classes of models where its sequential depth grows with the input sequence length (Merrill et al., 2026). Although these constraints were seen as theoretical, there is growing evidence they affect models in practice as well (Liu et al., 2025).

In SMT, we train a nonlinear RNN (which is fully expressive), using a time-parallel teacher Transformer (which has limits). We note this limitation but argue that SMT is a pretraining algorithm, which should be used with a lightweight post-training algorithm to solve downstream tasks (Gan & Isola, 2026).

Predictive State Representations (PSRs) A PSR is a way of modeling a partially observed dynamical system by representing its state only in terms of predictions about future observations (Littman & Sutton, 2001; Singh et al., 2012), a representation that is sufficient for optimal decision making (Singh et al., 2003). Early works interpreted PSRs as a literal vector of probabilities of future events, but have since been generalized (Downey et al., 2017a). Belief states are a similar concept, which also defines a sufficient statistic of the past (Kaelbling et al., 1998; Hu et al., 2024).

PSRs have been previously incorporated into RNNs (Downey et al., 2017b; Hefny et al., 2018). Venkatraman et al. (2017) introduce an auxiliary objective for RNNs that trains hidden states to predict statistics of future observations using a decoder. However, these works still unroll the RNN and use BPTT, and thus are not time-parallelizable and have a $\mathcal{O}(T)$ credit path.

Other Related Work Our work is related to the literature on cross-architecture teacher-student distillation (Kasai et al., 2021; Wang et al., 2025b; Hauzenberger et al., 2026; Chen et al., 2026; Moudgil et al., 2026), but these works do not address the challenges of training nonlinear RNNs. The Recurrent Transformer is an RNN architecture that attends to all past hidden states, creating an $\mathcal{O}(1)$ gradient path that stabilizes credit assignment (Oncescu et al., 2026). However, because it retains all past hidden states, its memory grows unboundedly during inference—making it more akin to a Transformer than a fixed-memory RNN. Crucially, training still requires sequential unrolling and BPTT. SMT, by contrast, replaces BPTT and supports arbitrary fixed-memory RNN architectures and enables time-parallel training by never unrolling the RNN. Other works similarly combine Transformers with recurrent processing but also train with sequential unrolling and BPTT (Fan et al., 2020; Bulatov et al., 2022; Wu et al., 2022). A new line of work uses principles from diffusion models to train blocks of a feed-forward network in parallel, avoiding global backpropagation (Li et al., 2025b; Shing et al., 2025).

B DEFINITIONS

Credit Assignment Path Length For any differentiable computational graph, backpropagation propagates gradients from the scalar loss backward through the graph to each leaf node (typically model weights). We define the **credit assignment path length** as the maximum distance between any two nodes (e.g. tokens) in the computation graph. Distance is measured as the number of intervening non-identity operations that modify gradients (e.g. matrix multiplications or nonlinearities). The longer this path, the less effective backpropagation is for properly learning associations between distant nodes and assigning credit (Bengio et al., 1994). Under this definition, BPTT has $\mathcal{O}(T)$ credit assignment path length, whereas Transformers and SMT have $\mathcal{O}(1)$ path length between any two tokens.

Sequential Computation (measured in SeqFLOPs) Sequential computation is the amount of serial (non-parallelizable) work required to complete a computation. Some computations may require substantial total work but little sequential work (e.g. matrix multiplication). As parallel hardware such as GPUs continues to scale, total work matters less than the amount of inherently sequential work required (Hooker, 2021).

Sequential compute is measured by analyzing the computation graph required to execute an algorithm and computing the graph's *critical path*: the number of floating point operations that must be executed sequentially on an infinitely parallel computer. We refer to this quantity as **sequential FLOPs** (SeqFLOPs).

For simplicity, we estimate SeqFLOPs by counting the number of sequential atomic deep learning operations (e.g., Linear, LayerNorm) executed over the course of the algorithm. The true SeqFLOPs, measured in floating-point operations, is proportional to this estimate.

We compute SeqFLOPs for BPTT, SMT, and DMT. SMT is fully parallelizable in time, incurring $\mathcal{O}(1)$ SeqFLOPs per optimization step, independent of the sequence length T. In contrast, BPTT and DMT require unrolling the RNN, which increases SeqFLOPs to $\mathcal{O}(T)$ per optimization step.

Data Processed (measured in Tokens) Data Processed is defined as the total number of tokens processed during training, including repeated tokens in multi-epoch settings.

C EXPERIMENT DETAILS

C.1 ARCHITECTURES

Our primary architecture used for most experiments is shown in Figure 10.

Encoder Architecture We use the same encoder model architecture across all experiments. The model begins with an embedding layer to embed input tokens, and a list of learned memory token registers. The input and register tokens are concatenated and processed by a stack of *bidirectional* (full attention mask) Transformer blocks. We use bidirectional model because the goal is to create a holistic representation of the entire input sequence. The register tokens are then interpreted as memory tokens at the output. Note that a single memory consists of a list of memory tokens, $m_t = [m_t^1, \ldots, m_t^M]$. Within a Transformer block, we use rotary position encodings (Su et al., 2024) and RMSNorm instead of LayerNorm. We perform RMSNorm on the output memory tokens for stability. Figure 10 shows the encoder architecture.

Decoder Architecture We use the same decoder model architecture across all experiments. The decoder has an embedding layer to embed input tokens, which is weight shared with the encoder model. The memory tokens from the encoder and the embedded future input tokens are concatenated, and then processed by a stack of *causally masked* Transformer blocks. We use a causal mask because the goal is to learn a generative model of the output sequence. Within the each Transformer block, we use rotary position encodings (Su et al., 2024) and RMSNorm instead of LayerNorm. The output predictions are read out at token positions such that $\hat{y}_{t+k}$ is a function of only m_t and $x_{t+1}, \ldots, x_{t+k}$. Figure 10 shows the decoder architecture.

Transformer-based RNN Architecture The Transformer-based RNN is our primary RNN architecture and is used for most experiments. It begins with an embedding layer for the current timestep's

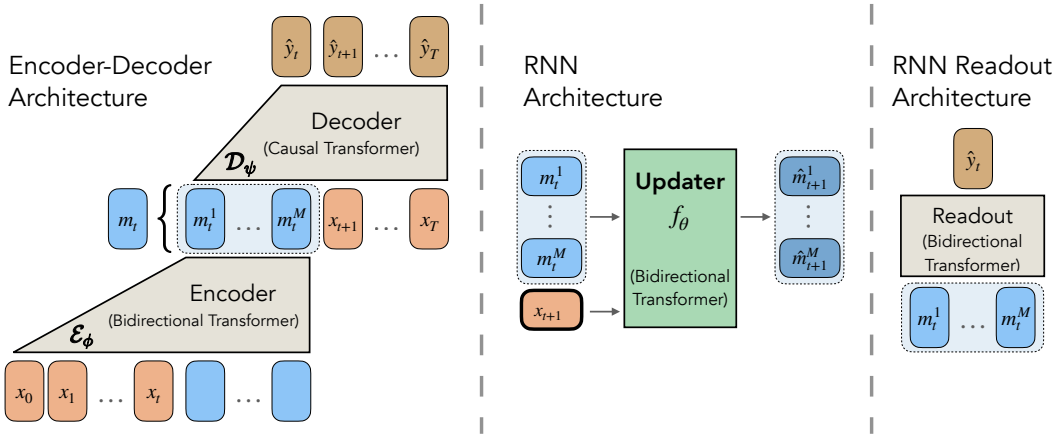

Figure 10: **Model Architecture for SMT. Left:** The encoder reads the input context tokens and a set of learned register tokens, and outputs the memory, m_t, which is a set of memory tokens. The decoder takes in this memory and the *future input* tokens and predicts the *future output* tokens, using a causal mask. This setup forces information from the context to be compressed into a memory that is useful for predicting the future outputs, given future inputs. **Middle:** Our RNN maps (m_t, x_{t+1}) to m_{t+1} using a Transformer-backbone. Since the memory is a list of tokens and the input is a token, we simply use a full attention Transformer to transform the current memory into the next timestep's memory. **Right:** Readout is performed by a full attention Transformer over the memory tokens.

input token. The memory tokens are concatenated with the input token and then processed by a stack of *bidirectional* (full attention mask) Transformer blocks to produce the output memory tokens. We perform RMSNorm on the output memory tokens.

MLP-based RNN Architecture The MLP-based RNN flattens the list of memory tokens into a single vector, concatenates it with the input token embedding, and passes them through an MLP. At the output, the model then unflattens them to be a list of memory tokens again. We perform RMSNorm on the output memory tokens.

GRU-Gated RNN Architecture The GRU-Gated RNN uses the Transformer-based backbone, but gates the memory update according to the GRU gate equations. Each gate variable is computed with a Transformer backbone.

LSTM-Gated RNN Architecture The LSTM-Gated RNN uses the Transformer-based backbone, but gates the memory update according to the LSTM gate equations. Each gate variable is computed with a Transformer backbone.

Stacked GRU RNN Architecture The GRU-based RNN processes the M memory tokens with a M layer stacked GRU. Layer l reads a single memory token, m_t^l, and outputs a single memory token for the next timestep, m_t^{l+1}. We do *not* RMSNorm on the output memory tokens, since that would undermine the GRU's residual structure.

Stacked LSTM RNN Architecture The LSTM-based RNN processes the M memory tokens with a $M/2$ layer stacked LSTM. Each layer processes two memory tokens, corresponding to the cell state and hidden state of the LSTM layer. We do *not* RMSNorm on the output memory tokens, since that would undermine the LSTM's residual structure.

RNN Readout Architecture We use the same readout model architecture for all RNNs. The readout architecture takes in the memory tokens and processes them through a stack of *bidirectional* (full attention mask) Transformer blocks Figure 10 shows the readout architecture.

C.2 DATASETS

C.2.1 SYNTHETIC TASKS

Retrieval to test Gradient Stability The retrieval task requires the model to remember and reproduce the token immediately following a designated identifier (token 0). For example, given $\mathbf{x} = [3, 4, 0, 2, 1, 3, 1, 0]$ the target is $\mathbf{y} = [\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, 2]$, where $\emptyset$ denotes no prediction target. With probability p, the label is corrupted to a random token. By varying **sequence length** and **noise level**, this task probes the algorithm's capacity for stable gradient credit assignment.

String Copy to test Memory Capacity The string copy task requires the model to reproduce a sequence in reverse order after a delimiter (token 0). For example, given $\mathbf{x} = [3, 4, 1, 2, 0, 0, 0, 0]$, the target is $\mathbf{y} = [\emptyset, \emptyset, \emptyset, \emptyset, 2, 1, 4, 3]$. By varying the **sequence length** and the **memory state size**, this task measures the algorithm's ability to leverage the RNN's memory capacity for memorization.

Stack Operations to test State Tracking The stack operations task requires the model to track the top element of a stack through a sequence of push and pop operations (denoted by token 0). For example, given $\mathbf{x} = [1, 0, 2, 3, 0, 1, 0, 0]$, the target is $\mathbf{y} = [\emptyset, 1, \emptyset, \emptyset, 3, \emptyset, 1, 2]$. By varying the **sequence length** and **state complexity** (maximum stack depth), this task evaluates the algorithm's capacity for state tracking.

Keys and Values to test Associative Recall The keys and values task requires the model to store and retrieve associations between keys and values, then recall the value corresponding to a queried key (Willshaw et al., 1969; Hinton & Anderson, 2014). For example, given $\mathbf{x} = [b, 1, a, 3, d, 2, 4, a]$, the target is $\mathbf{y} = [\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, 3]$. By varying the **number of associations** and **association complexity** (string length of keys and values), this task evaluates the algorithm's capacity for associative recall.

Modular Arithmetic to test In-Context Learning The modular arithmetic task requires the model to infer a latent linear rule from in-context examples and apply it to novel inputs (Akyürek et al., 2022; Kirsch et al., 2022). For each sequence, parameters a and b are sampled, then the sequence is presented as $\mathbf{x} = [x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3,]$, where $y_i = (ax_i + b) \mod V$, where V is the vocabulary size. Then, the target is $\mathbf{y} = [y_0, \emptyset, y_1, \emptyset, y_2, \emptyset, y_3, \emptyset]$. By varying the **difficulty** (range of values a, b can take on) and the number of **in-context examples**, this tasks evaluates the algorithm's ability to induce in-context learning.

C.2.2 NATURAL TASKS

TinyStories TinyStories is a curated dataset of short stories generated by OpenAI's GPT-4 (Eldan & Li, 2023). We use ASCII character-level tokenization, yielding a vocabulary of 256 tokens. Under this tokenization, the training and test sets contain 1.9B and 19.2M tokens, respectively.

MNIST MNIST is a classic image dataset consisting of handwritten digits (LeCun & Cortes, 1998). Rather than performing classification or 2D image generation, we consider the problem of 1D pixel-sequence modeling. The original 28×28 images are flattened into sequences of length 784 using raster-scan ordering. Each image is represented as a sequence of raw grayscale pixel intensities (0–255), yielding a vocabulary of 256 tokens. The training and test sets contain 47M and 7.8M tokens, respectively.

Sketchy Sketchy is an image dataset of human-drawn sketches (Sangkloy et al., 2016). Rather than performing classification or 2D image generation, we consider the problem of 1D pixel-sequence modeling. The original images are resized to 64×64 using Lanczos resampling, and the pixels are binarized. Non-overlapping 2×2 patches are tokenized, yielding a vocabulary of $2^{2\times 2} = 16$ tokens. The resulting image is flattened in raster-scan order to form sequences of length 1024. The training and test sets contain 69.5M and 7.7M tokens, respectively.

C.3 ALGORITHMS

For all experiments, we use the AdamW optimizer with a weight decay of 0.01 and learning rates tuned separately for each algorithm. For all methods, gradients are clipped to a maximum global norm of 1. Gradient clipping is expected to be particularly beneficial for BPTT.

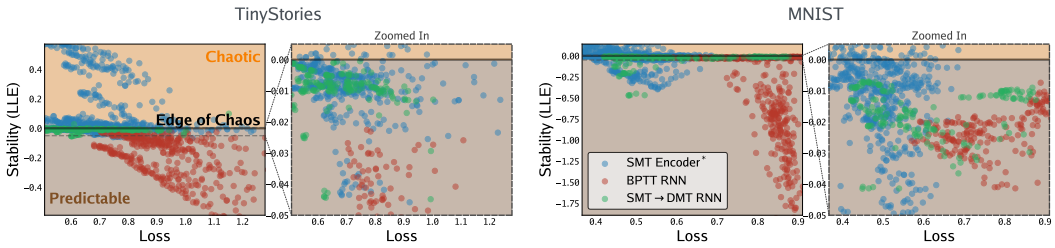

Figure 11: **Stability of RNNs.** We measure the Largest Lyapunov Exponent (LLE) of the dynamical system associated with the model from each method (Gonzalez et al., 2025). BPTT often finds systems which are predictable (i.e. those with vanishing gradients). The SMT Encoder often seems to be at the edge of chaos or fully chaotic. Interestingly, SMT→DMT consistently discovers RNNs which are at the edge of chaos. The SMT encoder's LLE is evaluated using SMT RNN at the encoder's memory states.

After SMT, we transfer the decoder weights to the RNN readout module. During DMT, this readout head is further finetuned to optimize the next-token prediction loss using RNN-generated memory states. Importantly, this task loss updates only the readout head and not the RNN dynamics function, and therefore does not constitute temporal credit assignment for the RNN itself. Instead, finetuning serves to adapt the readout head to imperfections in the memory states generated by the RNN.

Synthetic Experiments We set $T_c = T_f = T$. To evaluate the expectation in $\mathcal{L}^{\text{smt}}$, we compute the loss terms at all timesteps $t \in [0, \ldots, T]$. For earlier timesteps, where the available past context is shorter than the required context length, we pad the sequence and modify the attention mask so that padding tokens are ignored. The same procedure is applied to the future context. In these synthetic experiments, the prediction loss is applied only at positions where target output tokens are defined (e.g. the answer token in the needle task). We use batch sizes of 32 sequences during optimization, but this gets expanded to $32 \times T$ input contexts to the encoder.

Other Experiments To evaluate the expectation in $\mathcal{L}^{\text{smt}}$, we compute the loss term at a single timestep sampled uniformly from $t \sim \mathcal{U}[0, \ldots, T]$. The dataset is represented as one long sequence, meaning padding is not required, as both the past and future contexts extend indefinitely. We compute the $\mathcal{L}^{\text{unif}}$ over batches of memories from different sequences.

By default for SMT, we set $T_c = 256$, $T_f = 64$, $\lambda_{\text{dec}} = 1.0$, $\lambda_{\text{dyn}} = 0.1$, $\lambda_{\text{unif}} = 0.001$ and train for 150000 SGD iterations. Unless otherwise specified, models use a hidden dimension of 256 with 16 memory tokens. The encoder is 8 layers deep, while the decoder is 4 layers deep. The RNN is also 8 layers deep, and its readout function is 4 layers deep. We use a batch size of 128 sequences.

D Additional Experiments

λ Coefficients The values of λ_{dyn} and λ_{unif} are swept here to check their effects. Figure 20 shows the results. The best RNNs require $\lambda_{\text{dyn}} = 0.1$, and $\lambda_{\text{unif}} = 0.001$. When $\lambda_{\text{unif}} = 0$, although the RNN performance is preserved, the memory space is collapsed, as indicated by $\mathcal{L}^{\text{unif}}$.

Drift and DMT As described in Section 2.3, RNN suffers from drift post-SMT. Figure 13 shows an analysis of drift and DMT's mitigation of it. From a dynamical systems perspective, DMT seems to discover RNNs which have an initially higher drift, but which *plateau* at a much lower equilibrium drift. Interestingly, this equilibrium drift value is not fully predicted by the one-step drift, inviting future investigations into predicting and mitigating rollout drift in one-step during SMT.

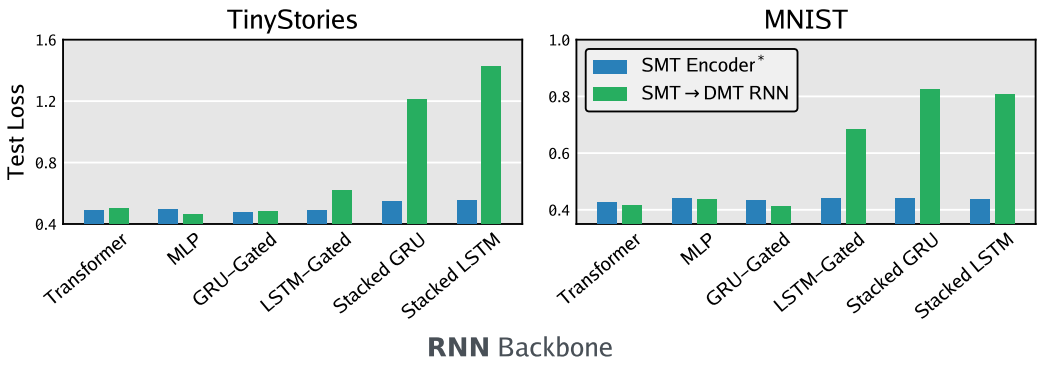

Figure 12: **SMT→DMT Performance over Architectures.** SMT→DMT is able to train RNNs with a variety of backbones. However, it struggles with vanilla stacked GRUs and LSTMs. We hypothesize this may be because these architectures are relatively obsolete and don't contain modern stability practices such as RMSNorm, LayerNorm, MLPs, etc. SMT is able to train GRU-Gated and LSTM-gated models, which are modern architecture RNNs with GRU/LSTM style gating applied on the memory updates.

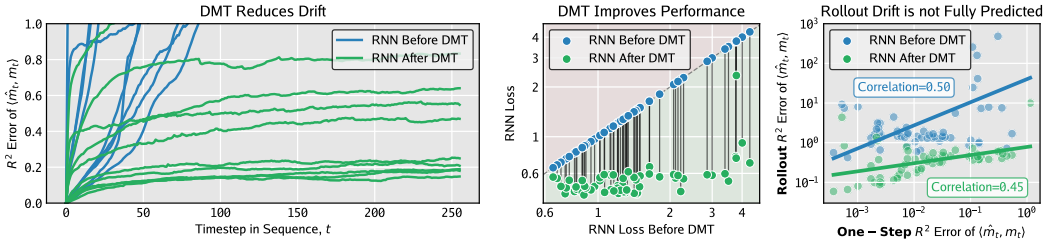

Figure 13: **Impact of DMT** across many runs with different SMT λ_{dec} and λ_{dyn} hyperparameters. **Left:** Applying DMT reduces the drift of the RNN rollout (measured with $1 - R^2$ of RNN memory prediction $\hat{m}_t$ of encoder ground truth m_t). **Middle:** DMT significantly improves RNN performance across settings. **Right:** The one-step drift of the RNN only partially correlates with the rollout drift.

D.1 Compression as a Scaling Axis

Neural scaling laws predict the relationship between a resource (e.g. compute, data) and a desired property (e.g. validation loss, benchmark accuracy) (Kaplan et al., 2020; Hoffmann et al., 2022). Can the desired property instead be *compression* (Hutter, 2005)? For RNNs, compression can be interpreted as achieving the same performance with a smaller memory state size. Thus, to answer this question, we train a set of SMT models on TinyStories across a sweep of memory state sizes and training compute budgets.

Figure 14 shows the scaling curve, confirming that SMT can achieve more compression when allocated more compute. Since compression is often speculated as being a core property of intelligent systems (Solomonoff, 1964; Kolmogorov, 1968), scaling along this compression axis may be a desired direction forward for future sequence models. Notably, Transformers perform no compression of the past (Gu, 2025), which may explain their training efficiency.

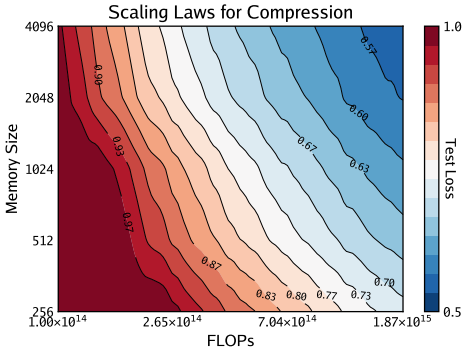

Figure 14: **Scaling Laws for Compression.** We plot iso-loss contours for SMT-trained encoder models across a range of memory state sizes and training compute budgets. For a fixed target performance, SMT can achieve higher compression (smaller memory size) using additional compute. This result suggests a new property to scale when given more training compute: memory state compression.

Benefit of RNNs over Transformers SMT trains an RNN to mimic a Transformer encoder model, raising the question of why an RNN is needed at all, given the Transformer. RNNs are qualitatively more efficient than Transformers at inference, requiring $O(1)$ rather than $O(T)$ memory and compute per generated token (Table 1). RNNs also constitute a more expressive class of models (Merrill et al., 2026; Liu et al., 2025).

Here, we compare an SMT→DMT RNN against a Transformer on the synthetic stack state tracking task. For a fair comparison, we use the SMT encoder as the Transformer baseline, since it imposes the same memory-information bottleneck as the RNN.

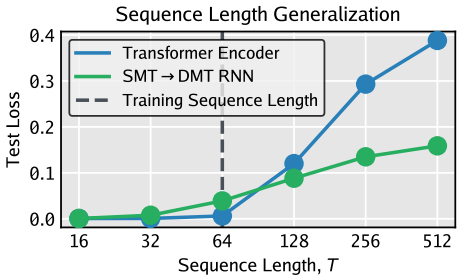

Figure 15: **Sequence Length Generalization.** An SMT→DMT trained RNN generalizes better than its Transformer teacher when evaluated on sequence lengths longer than training. The task is synthetic state tracking.

Figure 15 shows the Transformer outperforms the RNN on training sequence lengths, but significantly underperforms the RNN on sequence lengths longer than training. Prior work on length generalization reports similar findings (Press et al., 2021). This result reflects the distinct inductive biases of the architectures: Transformers behave like growing lookup tables in context, while RNNs update finite states (Gu, 2025). The latter is a better inductive bias for generalization.

Memory Space To better understand what SMT is learning, we train smaller SMT models that have a 2D memory state and directly visualize their memory space across three synthetic tasks in Figure 16. In the retrieval tasks, SMT learns to collapse many sequence states into only a few effective memory states: an initial state, a state indicating the next token is the needle, and states corresponding to the needle value. Then, the RNN learns finite-state machine behavior to transition between these states. In contrast, string copying requires lossless sequence compression and thus SMT cannot alias distinct memory states together. It learns to create a tree-like memory geometry to store all possible sequences, matching the tree structure of all possible strings. Figure 25 and Figure 26 show memory visualizations for models trained on MNIST. These results indicate SMT memories form effective temporal *abstractions* of the past depending on what the future requires.

Figure 20 shows the results of ablating the λ_{dyn} and λ_{unif}. Results show the optimal RNN performance requires a moderate dynamics loss, paired with a very low uniformity loss. However, a little uniformity is critical for avoiding memory space collapse.

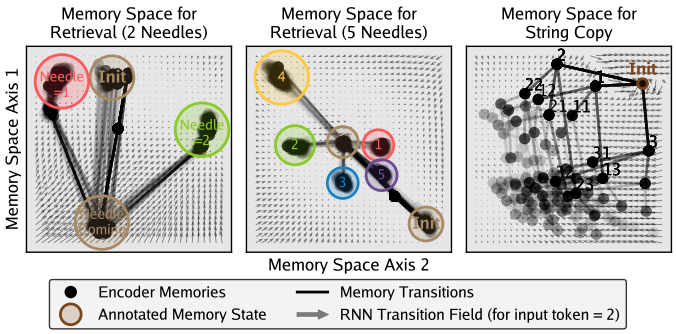

Figure 16: **Memory Space Visualization.** The encoder learns different memory geometries for different tasks. In Retrieval, the encoder collapses many sequence states into a few memory states, creating finite-state machine like behavior. In String Copy, the encoder constructs a tree-like memory geometry to compress all possible sequences. Some geometries induce more complex RNN transition fields.

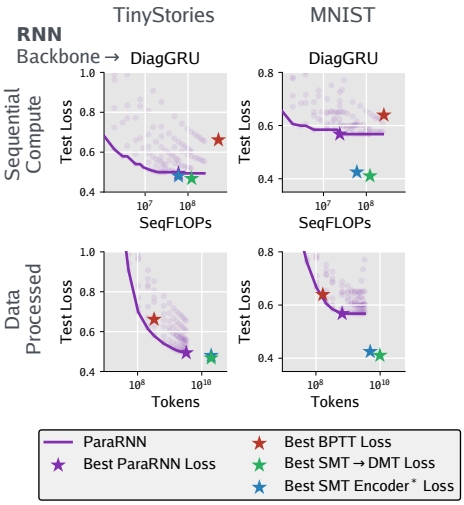

Figure 17: **ParaRNN comparison.** Here, we evaluate the performance of ParaRNN (Danieli et al., 2025) (an implementation of DEER (Lim et al., 2024)) in our setup. We find that ParaRNN significantly outperforms BPTT, as expected. On TinyStories, ParaRNN is able to match the performance of SMT→DMT. However, ParaRNN lags behind SMT→DMT on MNIST, likely a result of the long-range dependencies required for the task. DEER only approximates BPTT, and thus struggles with vanishing gradients and long-range dependencies.

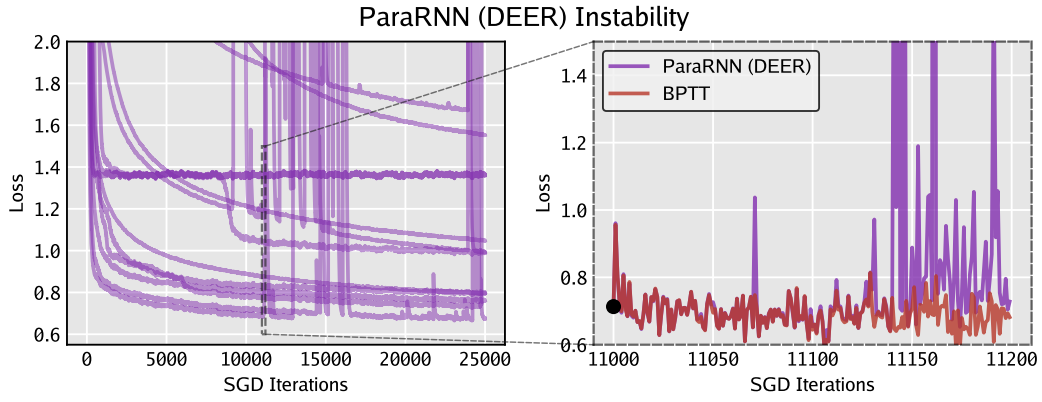

Figure 18: **Instability of ParaRNN. Left:** Over some runs of training RNNs with DEER (Lim et al., 2024; Danieli et al., 2025), we observed instability in the training runs, as also documented by Gonzalez et al. (2025). **Right:** DEER and BPTT are initialized with the same RNN checkpoint. DEER approximates BPTT, until it becomes unstable, likely because of the unpredictable dynamics (Gonzalez et al., 2025). In contrast, SMT enjoys stable optimization which does not depend on the properties of the RNN dynamics.

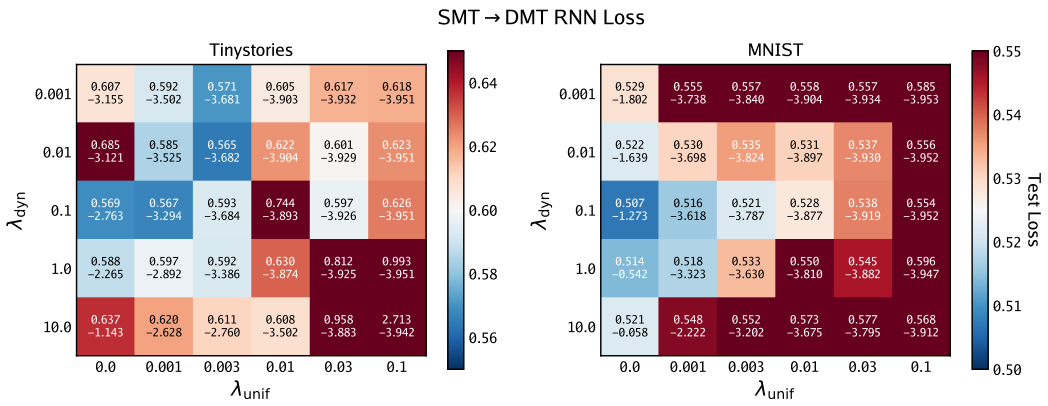

Figure 20: **Sweep of λ_{dyn} and λ_{unif}.** Cell color indicates the RNN test loss for each setting. Top number in each cell is the RNN test loss. Bottom number in each cell shows the $\mathcal{L}^{\text{unif}}$. $\mathcal{L}^{\text{unif}}$ varies from 0 (collapsed latent space) to -4 (fully uniform latent space).

Figure 21 shows more samples of generations from Figure 4. Samples generated by the BPTT RNN (Transformer backbone) seem to only pick up on short range context and act accordingly: either output large streaks of white or black based on the current row. BPTT RNN (GRU backbone) improves this significantly, but still fails to capture the nuanced structure of digits. SMT→DMT RNN (Transformer backbone) is able to capture this structure quite well.

Figure 22 shows more samples of generations from Figure 19. These generations are often not fully interpretable, but do capture the stroke structure of human-drawn sketches. Capturing this stroke structure is itself a difficult problem, given the long-horizon nature of pixel sequence modeling.

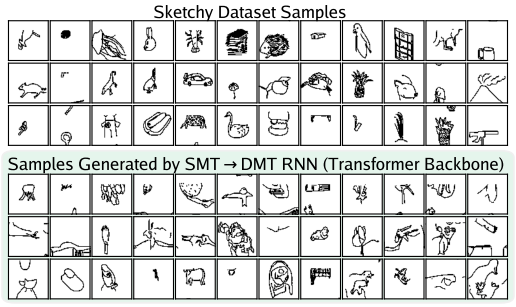

Figure 19: **Attneave's Sketchy Generation.** SMT→DMT captures the stroke structure of human-drawn sketches through only pixel sequence modeling on sparse images. More samples are in Appendix Figure 22.

Figure 23 provides an analysis of our Sketchy RNN as it "reads" a sequence corresponding to the classic Attneave's cat image (Attneave, 1954). The memory sequence does not seem to be fully interpretable, but does show significant structure. Figure 24 provides samples of generations when conditioned on partial context of Attneave's cat image. Figure 25, 26, 27, show additional analysis of the RNNs on MNIST and Sketchy data.

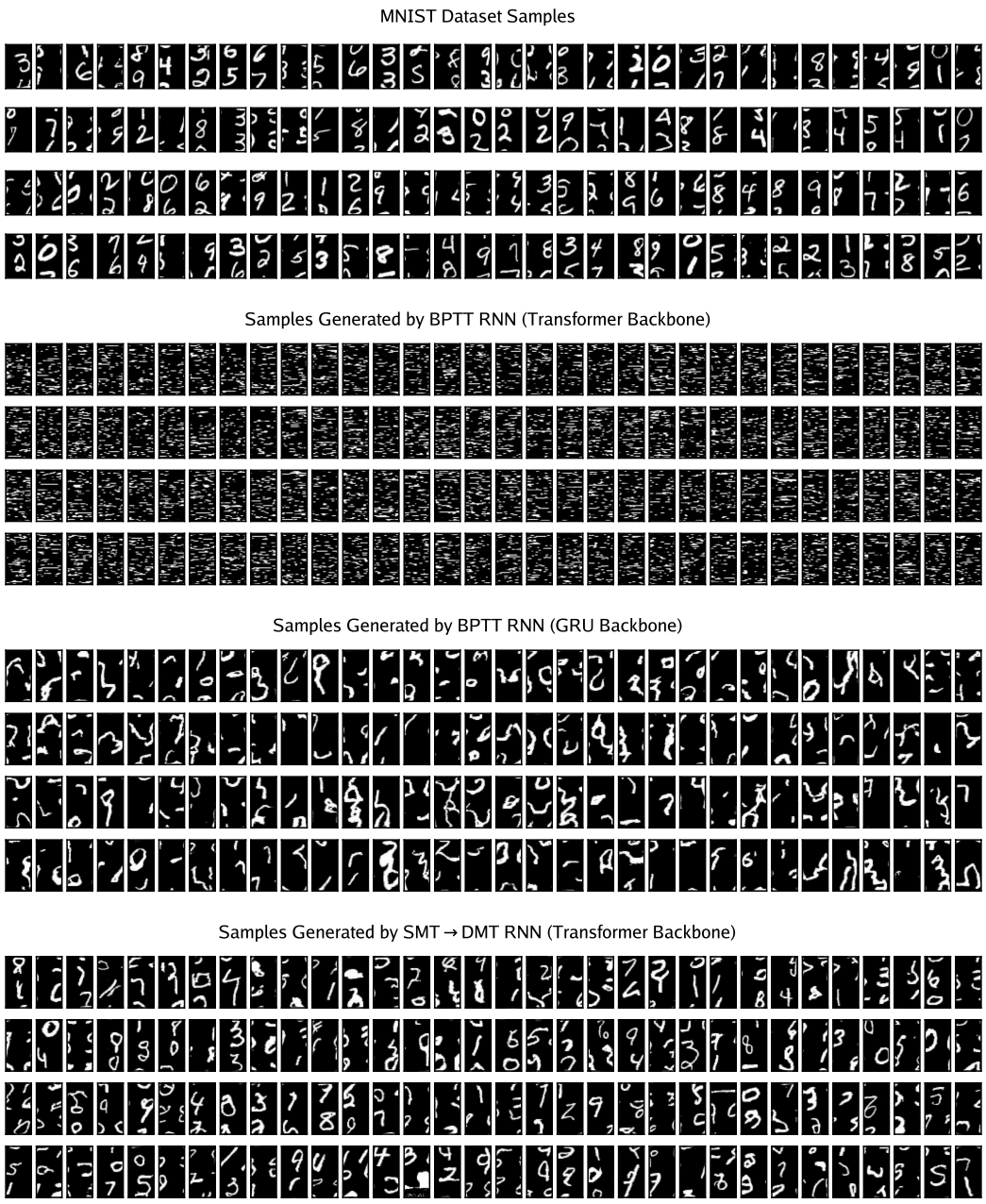

Figure 21: **Additional MNIST Samples.** Here we give more examples of samples of MNIST images generated by the various methods. SMT→DMT RNN outperforms BPTT, even when BPTT is applied on a GRU architecture, in processing long-horizon information, which is required for pixel modeling.

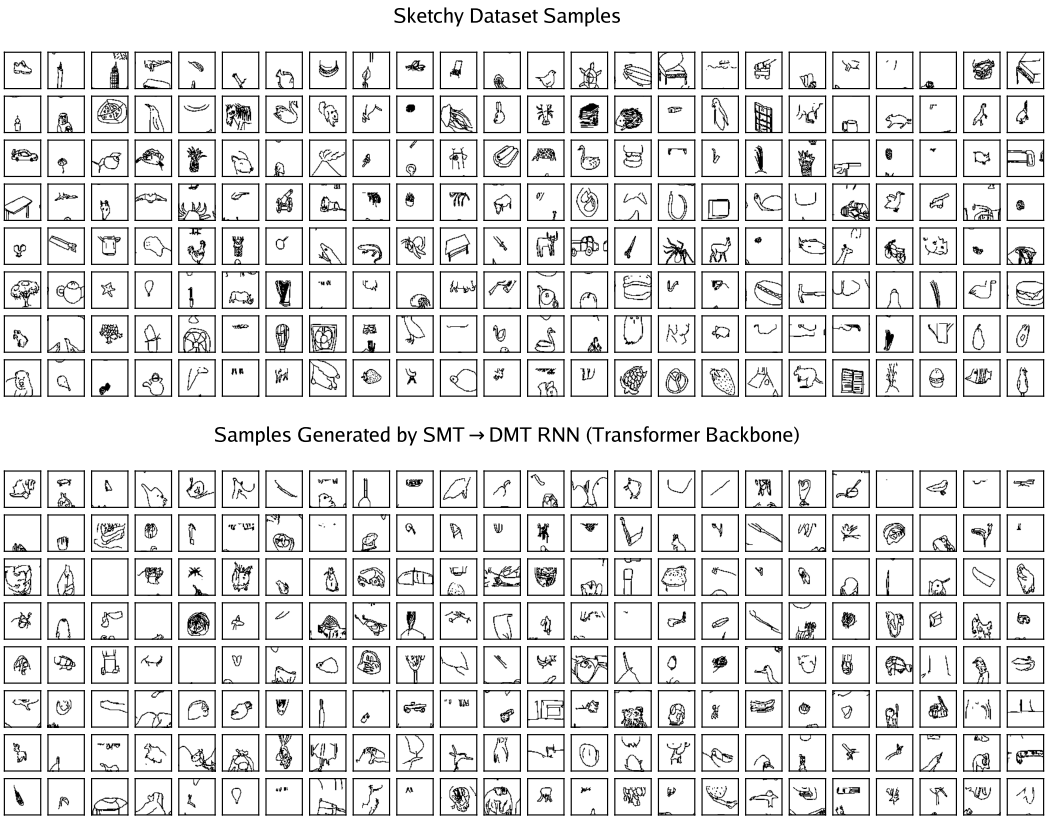

Figure 22: **Additional Sketchy Samples.** Here we give more examples of samples of Sketchy images from the dataset and generated by SMT→DMT. Even in this hard sparse domain, SMT→DMT can capture the overall stroke structure, which requires integrating information over hundreds of pixels.

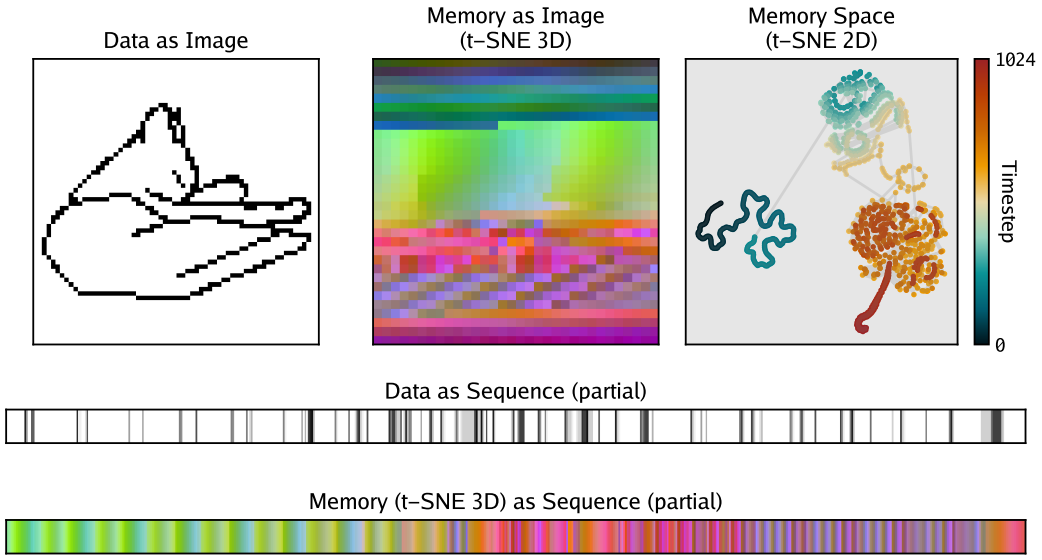

Figure 23: **Analysis on Attneave's Cat.** We apply the SMT→DMT-trained RNN on Sketchy and evaluate it on the classic image of Attneave's cat. The RNN reads the image pixel-by-pixel in raster scan order. **Top Left:** Input image presented in its original 2D form. **Top Middle:** 3D t-SNE projection of the RNN memory state, visualized as RGB values over time, showing the evolution of memory throughout sequence processing. **Top Right:** 2D t-SNE projection of the memory state trajectory over time. **Middle:** The same image presented as a flat token sequence. From the RNN's perspective, the task resembles modeling a barcode-like sequence, requiring long-range associations between distant tokens and highlighting the difficulty of pixel sequence modeling. **Bottom:** 3D t-SNE projection of the memory state visualized along the flattened sequence.

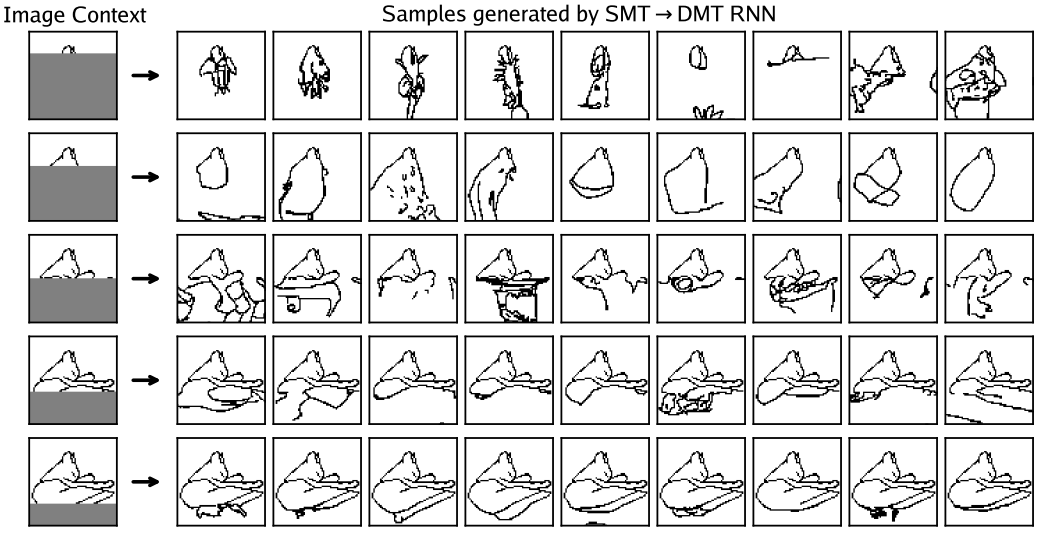

Figure 24: **Generations of Attneave's Cat.** We apply the SMT→DMT-trained RNN on Sketchy and apply it to generate part of the image of Attneave's cat. Given more of the image context, the RNN seems to understand the image better and make somewhat more plausible predictions.

E SEQUENCE TO SET REFRAMING

As described in Section 2.2, consider a hypothetical oracle memory-encoding model $\mathcal{Q}$ that takes as input the sequence of tokens and outputs an effective compressed memory. Here we show that $\mathcal{Q}$ does *not* have to be a recurrent function over the sequence of tokens, but can instead be represented as a permutation-invariant function over a *set* of timestamped tokens.

Claim Let $\mathbf{x}_{\text{seq}} = [x_0, x_1, \ldots, x_t]$ be the original sequence of tokens. We define the set $\mathbf{x}_{\text{set}} = \{(x_0, 0), (x_1, 1), \ldots, (x_t, t)\}$. Assume that $\mathcal{Q}$ is a recurrent function over $\mathbf{x}_{\text{seq}}$. In other words,

$$m = \mathcal{Q}(\mathbf{x}_{\text{seq}}) = f(\ldots f(f(m_\emptyset, x_0), x_1), \ldots, x_t)$$

with $m_\emptyset = \mathbf{0}$ for some function f. For any such $\mathcal{Q}$, $\exists$ a function g such that $g(\mathbf{x}_{\text{set}}) = \mathcal{Q}(\mathbf{x}_{\text{seq}})$.

Proof We construct g explicitly. Define $g(\mathbf{x}_{\text{set}})$ as follows: given the input set $\mathbf{x}_{\text{set}} = \{(x_0, 0), (x_1, 1), \ldots, (x_t, t)\}$, sort the elements in ascending order of their timestamp to recover the sequence $\mathbf{x}_{\text{seq}} = [x_0, x_1, \ldots, x_t]$, then apply $\mathcal{Q}$ to this sequence.

This is well-defined because the timestamps $\{0, 1, \ldots, t\}$ are distinct integers, so the sort order is unique. The resulting sequence is identical to the original $\mathbf{x}_{\text{seq}}$, and therefore $g(\mathbf{x}_{\text{set}}) = \mathcal{Q}(\mathbf{x}_{\text{seq}}) = m$.

Moreover, g is permutation-invariant: any permutation of the elements of $\mathbf{x}_{\text{set}}$ yields the same sorted sequence and thus the same output.

Since $\mathcal{Q}$ was arbitrary, this construction applies to every recurrent $\mathcal{Q}$, completing the proof. ■

Implication This result implies that any sufficiently expressive permutation-invariant set model can in principle exactly model a recurrent memory function. Because sets are unordered, time-parallel processing naturally follows. In particular, Transformer-based architectures can be interpreted as operating over sets of timestamped tokens rather than strictly ordered sequences.

Notably, the proof is constructive: g recovers the sequential computation by sorting the timestamps and implicitly applying the recurrent update rule f up to t times. Consequently, when implemented with bounded-depth architectures such as Transformers, the required depth may need to scale with sequence length, consistent with prior work on sequential depth and time-parallel training discussed in Section 4. Scaling depth with sequence length seems to present a major theoretical limitation.

However, our empirical results suggest that even relatively shallow Transformer encoders can learn highly effective memory representations for both synthetic and natural tasks. Thus, despite lacking full theoretical expressivity, this sequence-to-set reframing may still provide a practical strategy for memory *pretraining*. For full expressivity, some light-weight post-training may be required.

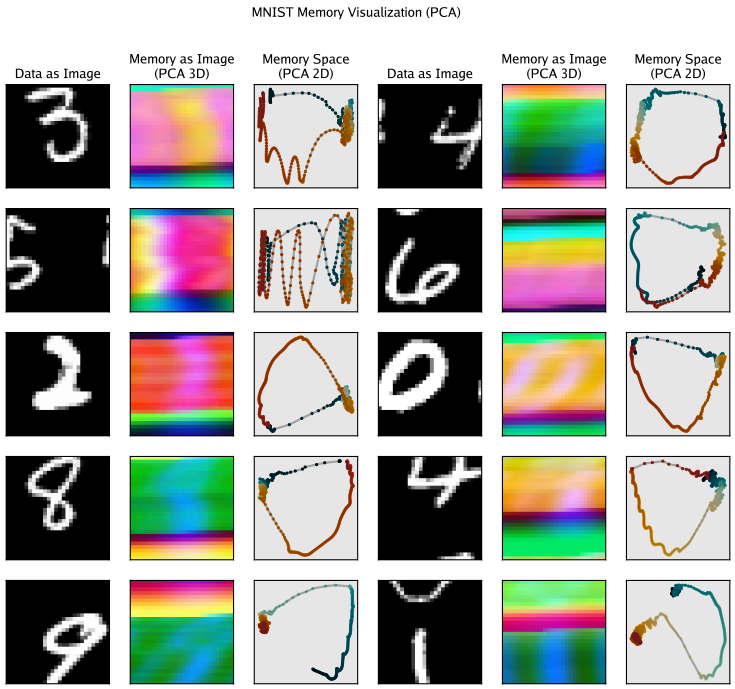

Figure 25: **RNN Memory Evolution on MNIST (PCA)**. We analyze the memory evolution of our SMT→DMT MNIST RNN. **Left**: Input image presented in its original 2D form. **Middle**: 3D PCA projection of the RNN memory state, visualized as RGB values over time, showing the evolution of memory during processing. **Right**: 2D PCA projection of the memory state trajectory over time.

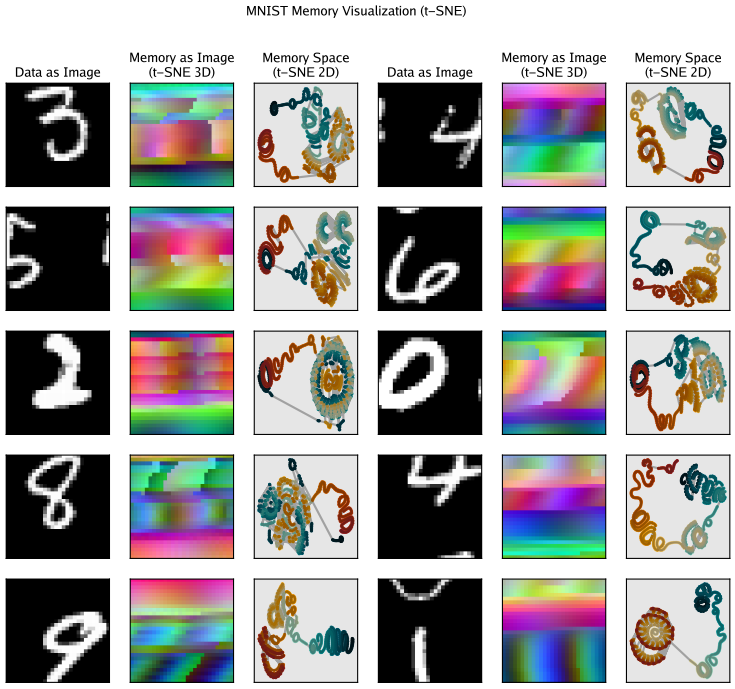

Figure 26: **RNN Memory Evolution on MNIST (t-SNE)**. We analyze the memory evolution of our SMT→DMT MNIST RNN. **Left**: Input image presented in its original 2D form. **Middle**: 3D t-SNE projection of the RNN memory state, visualized as RGB values over time, showing the evolution of memory during processing. **Right**: 2D t-SNE projection of the memory state trajectory over time.

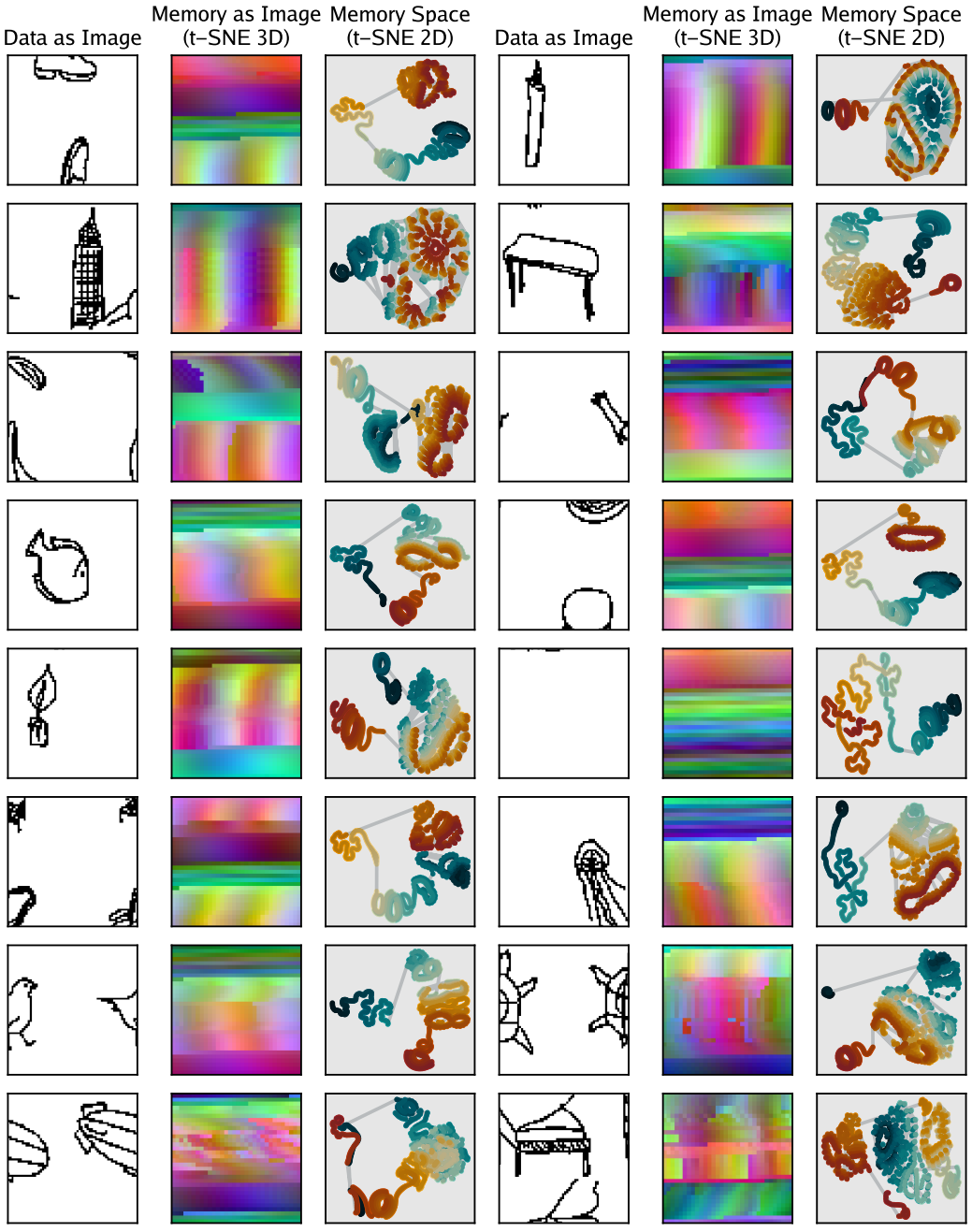

Figure 27: **RNN Memory Evolution on Sketchy (t-SNE)**. We analyze the memory evolution of our SMT→DMT Sketchy RNN. **Left**: Input image presented in its original 2D form. **Middle**: 3D t-SNE projection of the RNN memory state, visualized as RGB values over time, showing the evolution of memory during processing. **Right**: 2D t-SNE projection of the memory state trajectory over time.

F ENCODER MARKOVIAN TRAINING

SMT consists of two primary objectives: future predicting with $\mathcal{L}^{\text{dec}}$ and dynamics modeling with $\mathcal{L}^{\text{dyn}}$. The dynamics objective serves two purposes: (1) training the RNN to predict the next memory state from the current one, and (2) encouraging the encoder to produce memory states that are predictable from one another, i.e. approximately Markovian. In this section, we show that the predictive state objective $\mathcal{L}^{\text{dec}}$ alone is sufficient for learning Markovian memories, implying that $\mathcal{L}^{\text{dyn}}$ is theoretically unnecessary, though still practically useful.

Claim Let $\mathbf{x} = [x_0, x_1, \ldots, x_T]$ and $\mathbf{y} = [y_0, y_1, \ldots, y_T]$ be input and output sequences. For each timestep t, define

$$\mathbf{x}_t^{\text{ctx}} = [x_0, \ldots, x_t], \qquad \mathbf{x}_t^{\text{fut}} = [x_{t+1}, \ldots, x_T], \qquad \mathbf{y}_t^{\text{fut}} = [y_t, \ldots, y_T],$$

with memory state $m_t = \mathcal{E}_\phi(\mathbf{x}_t^{\text{ctx}})$ and reconstructed future $\hat{\mathbf{y}}_t^{\text{fut}} = \mathcal{D}_\psi(m_t, \mathbf{x}_t^{\text{fut}})$. If m_t is an *optimal* minimal sufficient statistic of $\mathbf{x}_t^{\text{ctx}}$ for predicting $\mathbf{y}_t^{\text{fut}}$ given $\mathbf{x}_t^{\text{fut}}$ at *every* t, then the memory sequence (m_t) is **Markovian**:

$$m_{t+1} \perp\!\!\!\perp \mathbf{x}_t^{\text{ctx}} \;\Big|\; m_t,\, x_{t+1}.$$

Proof By optimality of m_t, it is a minimal sufficient statistic of $\mathbf{x}_t^{\text{ctx}}$ for predicting $\mathbf{y}_t^{\text{fut}}$ given $\mathbf{x}_t^{\text{fut}}$:

$$H\big(\mathbf{y}_t^{\text{fut}} \mid m_t, \mathbf{x}_t^{\text{fut}}\big) = H\big(\mathbf{y}_t^{\text{fut}} \mid \mathbf{x}_t^{\text{ctx}}, \mathbf{x}_t^{\text{fut}}\big).$$

Note that $\mathbf{y}_{t+1}^{\text{fut}} \subseteq \mathbf{y}_t^{\text{fut}}$ and $\mathbf{x}_{t+1}^{\text{fut}} \subseteq \mathbf{x}_t^{\text{fut}}$, so optimality of m_t at time t implies it is also sufficient for $\mathbf{y}_{t+1}^{\text{fut}}$ given $\mathbf{x}_{t+1}^{\text{fut}}$:

$$H\big(\mathbf{y}_{t+1}^{\text{fut}} \mid m_t, x_{t+1}, \mathbf{x}_{t+1}^{\text{fut}}\big) = H\big(\mathbf{y}_{t+1}^{\text{fut}} \mid \mathbf{x}_t^{\text{ctx}}, x_{t+1}, \mathbf{x}_{t+1}^{\text{fut}}\big) = H\big(\mathbf{y}_{t+1}^{\text{fut}} \mid \mathbf{x}_{t+1}^{\text{ctx}}, \mathbf{x}_{t+1}^{\text{fut}}\big).$$

By optimality of m_{t+1}, it is a minimal sufficient statistic of $\mathbf{x}_{t+1}^{\text{ctx}}$ for predicting $\mathbf{y}_{t+1}^{\text{fut}}$ given $\mathbf{x}_{t+1}^{\text{fut}}$. Minimality means m_{t+1} retains no information from $\mathbf{x}_{t+1}^{\text{ctx}}$ beyond what is predictively necessary. Since (m_t, x_{t+1}) already constitutes a sufficient statistic for this same prediction task—as shown above—minimality of m_{t+1} forces it to be a function of (m_t, x_{t+1}):

$$m_{t+1} = f(m_t,\, x_{t+1})$$

for some measurable f. Therefore m_{t+1} is determined entirely by (m_t, x_{t+1}), and conditioning on these renders it independent of all earlier context:

$$H(m_{t+1} \mid m_t,\, x_{t+1}) = 0,$$

which is equivalent to $m_{t+1} \perp\!\!\!\perp \mathbf{x}_t^{\text{ctx}} \mid m_t, x_{t+1}$. Hence (m_t) is Markovian. ■

Implication This result establishes that under ideal conditions—sufficient encoder and decoder capacity, infinite future horizon, and exact optimization—the memory states learned by $\mathcal{E}_\phi$ form a Markov chain driven only by the previous state and the incoming token. In other words, the encoder *implicitly* learns a Markovian memory representation: m_{t+1} can be predicted from only (m_t, x_{t+1}).

In practice, finite capacity and approximate optimization relax this property, leaving m_{t+1} with residual dependence on $\mathbf{x}_t^{\text{ctx}}$ beyond (m_t, x_{t+1}). This gap motivates jointly training the dynamics loss $\mathcal{L}^{\text{dyn}}$ alongside $\mathcal{L}^{\text{dec}}$ to explicitly encourage Markovian structure in the learned memory sequence.

Relatedly, Teoh et al. (2025) provide a proof that one-step RNN dynamics with a $T_f = 1$ encoder also induce a predictive state memory representation.