



Mohit Kulkarni

Rethinking Sequence Models: Towards Scalable Test-Time Compute with Hybrid Reasoners

Master's Thesis

Institute of Neuroinformatics, University of Zurich and ETH Zurich
School of Engineering and Applied Sciences, Harvard University

Supervision

Prof. Dr. Cengiz Pehlevan, Harvard University
Prof. Dr. Valerio Mante, University of Zurich.

August 10th, 2025

Abstract

As scaling laws for pretraining began to plateau, a new paradigm of of scaling test-time compute emerged. Scaling test-time compute refers to the idea of chain-of-thought reasoning and aggregating information across multiple outputs to enhance model capability. These techniques have gained significant attention in the community with the latest reasoning models achieving state-of-the-art performance in a variety of tasks. As we are moving into the era of scaling inference-time compute, the need for efficient reasoning models becomes more pressing. Transformer based architectures, while effective, incur substantial costs at inference time due to quadratic complexity in the sequence length. Recent works propose the use of linear-attention variants that increase the inference efficiency over very long sequences. In this thesis, we explore the use of hybrid architectures that combine linear-attention variants with transformers to achieve efficient models that exhibit sufficient reasoning capabilities while being scalable to long sequences and multiple outputs. First, we present a unifying perspective of sequence-to-sequence models that allows us to tie a lot of the progress in sequence modelling over the past few years together. This perspective allows us to think of sequence-to-sequence models in a more general way and provides deep connections to vast existing literature on Hopfield networks and online learning. Second, we develop a novel cross-architecture distillation framework that allows us to distill reasoning capabilities from a pretrained transformer architecture and transfer them to a linear-attention-type hybrid architecture. This method allows us to get performant hybrid models with a fraction of the cost, providing academics with a tool to explore the design space of hybrid models. To further decouple architectural capacity and reasoning performance, we pretrain small hybrid models on math datasets. Using extensive ablations, we show the necessary elements of the architectural backbone and highlight avenues for further research. Third, we explore the representational differences between a transformer and a hybrid model. We specifically focus on the idea of in-context learning and design a simple task that allows us to visualize the representational differences between architectures.

Acknowledgements

I would like to thank my supervisor, Dr. Cengiz Pehlevan for his guidance and support throughout this thesis. This work was partially supported by the Heyning-Roelli Foundation.

Contents

Abstract	i
Acknowledgements	iii
1 Introduction	2
2 An Associative Memory Viewpoint	5
2.1 Historical Context	5
2.2 Hopfield Networks	7
2.2.1 Dense Associative Memories	8
2.3 Sequence-to-Sequence Models	9
2.3.1 Recurrent Neural Networks	10
2.3.2 Gated RNNs	11
2.3.3 Softmax attention	14
2.3.4 Linear Attention	16
2.3.5 Gated Linear Attention	19
2.3.6 Fast weight programmers and DeltaNet	20
2.4 A unifying perspective	23
2.4.1 The memory representation	23
2.4.2 The attentional Bias \mathcal{L}	24
2.4.3 Retention Regularizer	25
2.4.4 Optimization Algorithm	25
2.4.5 Deriving existing architectures	26
3 Methods	29
3.1 Architectural Backbone	29
3.1.1 Transformer Architecture	30
3.1.2 Sequence Mixing Layers: Softmax Attention	30
3.1.3 Sequence Mixing Layers: Mamba/GDN	32

3.1.4	Other architectural Components	33
3.2	Training Datasets and Evaluation Metrics	36
3.2.1	Training Datasets	37
3.2.2	OpenR1-Math-220k	37
3.2.3	Evaluation Datasets	37
3.3	Pretraining	38
3.4	Cross-architecture distillation	38
3.4.1	Overview and Previous Work	40
3.4.2	Four stage transfer procedure	40
3.5	Scaling inference-time compute	43
4	Results	47
4.1	Pretrained models	47
4.1.1	Transformer Baseline	47
4.1.2	Mamba	48
4.1.3	Gated DeltaNet	48
4.1.4	Ablation Studies: Transformer	49
4.1.5	Ablation Studies: Gated DeltaNet and Mamba	49
4.2	Cross architecture distillation	49
4.2.1	Gated DeltaNet and Mamba	51
4.3	Understanding Representations	55
4.3.1	In-context Graph Tracing	55
5	Discussion	59
5.1	Do we really need long context?	59
5.2	Recurrent Depth and Latent Reasoning	60
5.3	Test time training	61
5.4	Outlook on architectural design	61
6	Conclusion	63
A	Appendix	67
A.1	Chunkwise training of algorithms	67
A.2	Dataset Examples	69
A.2.1	OpenMathInstruct2	70
A.2.2	MetaMathQA	71
A.2.3	GSM8K	71
A.2.4	MATH500	72

A.2.5	OpenR1-Math-220k	73
A.2.6	OpenThoughts-114k-math	74
A.3	Further Pretraining Plots	75

Chapter 1

Introduction

An important ingredient in the success of LLMs is the ability to reason from information provided within a context. This capability, also referred to as in-context learning has often been attributed to the success of attention based architectures. Just as neural scaling laws for training compute were beginning to plateau, OpenAI's o1 pointed the community to a novel paradigm based on test-time scaling, also referred to as inference scaling (Kaplan et al., 2020; Henighan et al., 2020; Hoffmann et al., 2022; OpenAI et al., 2024). DeepSeek-AI et al. (2025)'s subsequent release of R1 further invigorated research by providing thorough details on the training process of reasoning models. As such, there has been a rapidly growing list of strategies for scaling test-time compute, the vast majority of which hinge on leveraging a model's internal chain of thought to "reason" before providing a final answer (Wei et al., 2022). These methods have all been shown to drastically improve the performance of models, especially on tasks such as mathematics and coding with verifiable rewards (Shao et al., 2024).

These test-time scaling methods can be categorized broadly into two main branches: sequential and parallel. Sequential scaling refers to increasing the length of a single chain-of-thought, where the model is trained to utilize or independently discovers techniques such as backtracking and self-reflection to improve the quality of the response (Kumar et al., 2024; Muennighoff et al., 2025). This can be achieved in a number of different ways, most notably through the use of reinforcement fine-tuning in DeepSeek-R1 and budget forcing in s1. Parallel scaling refers to increasing the number of chains of thought, with a fixed or learned strategy to combine answers by searching through completed responses to identify the best response candidates or average to reduce variance in responses (Snell et al., 2024; Brown et al., 2024). More advanced methods attempt to combine these methods together by encouraging the

model to search over multiple chains of thought simultaneously during generation and allow each chain-of-thought to learn from the others' mistakes/reasoning (Xie et al., 2024). Further yet, there exist some additional approaches to test-time scaling which allow the model to "reason" on internal representations without outputting tokens altogether (Hao et al., 2024). These methods have various advantages and disadvantages, and it is an area of active research to determine which methods are best-suited for a given task, model, and compute budget.

However, regardless of the specific test-time scaling approach, it remains the case that these boosts in reasoning capabilities always come at a great computational cost. For instance, OpenAI's o3 model was able to score an unprecedented 87.5% on the famous ARC-AGI benchmark, but this came at the cost of more than \$3000 per problem (OpenAI, 2024; Chollet, 2019). This inference cost is directly tied to the quadratic complexity of the Attention mechanism, which computes the relation between every token in a sequence resulting in a memory complexity of $\mathcal{O}(L^2)$ and time complexity of $\mathcal{O}(L^2D)$, compared to feed-forward layers which scale as $\mathcal{O}(LD^2)$, where L and D are the sequence length and embedding dimension respectively. Thus, as we scale to more or longer chains-of-thought, the total amount of compute spent on each new token within each chain scales quadratically.

To circumvent this, we use Attention-free architectures such as Mamba or Mamba2, also referred to as subquadratic architectures due to their complexity scaling sub-quadratically with sequence length (Gu and Dao, 2023; Dao and Gu, 2024). These architectures have been shown to closely match the performance of Transformers on many language tasks, and have been shown to be performant in a slew of other tasks (Liu et al., 2025; Ma et al., 2024; Nguyen et al., 2024; Yan et al., 2024). The development and popularization of this architecture has reinvigorated research into recurrent neural networks, resulting in a flurry of new subquadratic architectures (Yang et al., 2023; Sun et al., 2023a; Peng et al., 2023; Yang et al., 2024b; Beck et al., 2024). Additionally, recent methods have demonstrated that one can effectively distill from pre-trained Transformers into Mamba and other subquadratic architectures using supervised fine-tuning (Wang et al., 2024; Bick et al., 2025; Zhang et al., 2024a; Wang et al., 2025b). These distilled models outperform subquadratic models trained from scratch by leveraging the performance and optimization of high-quality models such as Meta's Llama series to develop highly performant subquadratic models with a limited compute and data budget (Dubey et al., 2024).

While these architectures can match Transformers on most tasks, recent works have identified a clear performance gap between Mamba and Transformers on in-context

learning tasks. In particular, these architectures struggle on language tasks requiring associative memory such as selective copying and multi-query associative recall (Arora et al., 2023; Jelassi et al., 2024), where the model must be able to effectively recall information seen only at inference time. This calls back to research on the Hopfield Network from Hopfield (1982), and its connections to the attention mechanism in Ramsauer et al. (2020). Empirical researchers have developed a suite of synthetic tasks that highlight differences between Mamba and Transformers while theoretical researchers have formally proven that transformers are strictly more expressive (Arora et al., 2024; Poli et al., 2024). Simply put, the finite size of Mamba’s hidden states limits its memory capacity. However, it is still unclear what impact this associative recall gap has on the performance of the model in real-world applications. Furthermore, researchers have found that hybrid models composed of Mamba layers interleaved with Attention layers can alleviate this associative recall gap in language tasks while retaining much of the computational gains (Lieber et al., 2024; Ren et al., 2024; Glorioso et al., 2024; Zuo et al., 2025).

In this thesis and our recent paper Chaudhry et al. (2025), we provide the first exploration into the efficacy of test-time scaling on these subquadratic architectures. We find that inclusion of a small proportion of attention layers, as little as 15-25% of the total layers, greatly increases performance on mathematical reasoning tasks. Further, we provide extensive ablations on the core components of these hybrid architectures, which precisely quantify their respective contributions. To extend our investigation to larger scale models, we adapt the cross-architecture distillation framework from Wang et al. (2025b) and train various linear-attention variants. We find that while distillation provides a viable pathway for experimentation, fully recovering the teacher’s performance proves challenging, making it difficult to disentangle inherent architectural limitations from procedural artifacts of the distillation process itself. To probe the underlying representational differences between architectures, we test these models on a simple graph task from Park et al. (2025) revealing that models with low attention struggle to form robust internal representations. We therefore hypothesize that the capacity for inferring semantics from context is a capability enabled by the attention mechanism. With recent open-source models from OpenAI OpenAI (2025) interleaving sliding window and full attention, it remains a critical and open question how to best design hybrid architectures with superior memory and reasoning capabilities in the era of test-time scaling.

Chapter 2

An Associative Memory Viewpoint

The ambition to create artificial systems that learn, store and retrieve information in a manner analogous to the human brain has been a central pursuit in science for nearly a century (Hebb, 1950; Willshaw et al., 1969; Hopfield, 1982; Rumelhart et al., 1986). Unlike computers, which consists of a location addressable memory, we seek to develop content-addressable memory. We want to retrieve a stored pattern of information by providing a partial, noisy or corrupted version of the pattern. This property, known as associative recall, is a hallmark of biological memory and its cognitive capabilities of error correction. The evolution of modern sequence models (Sutskever et al., 2014; Vaswani et al., 2023) is rooted in this long standing effort to find a powerful and scalable model of associative memory. To fully appreciate the modern sequence models, we must first trace their intellectual lineage back to the foundational ideas in computational neuroscience.

2.1 Historical Context

Neural network models that use their dynamics to perform some form of computation have been long studied in the field of computational neuroscience. McCulloch and Pitts (1943) proposed a binary neuron as a model for the functioning of biological neurons. Let $x_1 \dots x_n$ be the inputs to the neuron where each x_i takes a value of 0 or 1. Let $w_1 \dots w_n$ be the weights of the connections between the inputs and the neuron such that w_i can be either 1, -1, or 0. The neuron is said to be active if the weighted sum of the inputs is greater than a threshold θ . The output of the neuron is then given by,

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^n w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

The McCulloch-Pitts neuronal network of these simple units could implement any logical function (AND, OR, NOT). They demonstrated that complex computation could itself arise from the collective behavior of a network of simple, interconnected elements. This model, however was a static model. The weights and the threshold were fixed, and hence the model could compute, but not learn.

Donald Hebb in his 1949 book, *Organization of Behavior* (Hebb, 1950) proposed a principle of synaptic plasticity that would become one of the foundational postulates of neural networks. The Hebb's Postulate (Stent, 1973) states,

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

Often summarized as "neurons that fire together, wire together", this postulate was the first theory for how memories could be encoded in the brain. For two connected neurons with activations a_i and a_j , the change in synaptic weight is proportional to the product of their activations,

$$\Delta w_{ij} = \eta \cdot a_i \cdot a_j \quad (2.2)$$

This is the hebbian learning rule and allows a network to act on the statistical correlations present in the input data by strengthening the connections between neurons that are frequently active.

First matrix model of associative memory

Linear Associator (Anderson, 1972; Kohonen, 1972) was a simple feed-forward network that stored a set of P associations between input keys \mathbf{k}_i and output patterns \mathbf{v}_i in a weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$,

$$\mathbf{W} = \sum_{i=1}^P \mathbf{v}_i \mathbf{k}_i^T. \quad (2.3)$$

To retrieve a memory, and input key \mathbf{k}_j is presented to the network and the output is simply,

$$\mathbf{v}_{retrieved} = \mathbf{W} \mathbf{k}_j = \left(\sum_{i=1}^P \mathbf{v}_i \mathbf{k}_i^T \right) \mathbf{k}_j = \sum_{i=1}^P \mathbf{v}_i (\mathbf{k}_i^T \mathbf{k}_j) \quad (2.4)$$

If the input keys are orthonormal ($\mathbf{k}_i \mathbf{k}_j^T = \delta_{ij}$), then the retrieval is perfect. Our model has successfully associated each key with its corresponding value.

However, the requirement of orthonormal input patterns is a critical weakness of the linear associator model. If the keys are not orthonormal, the interference (referred to as crosstalk) between non-orthogonal memories severely limits the storage capacity of the system. These early models, however demonstrated the useful properties of recurrent architectures and led to the development of the Hopfield Network (Hopfield, 1982)

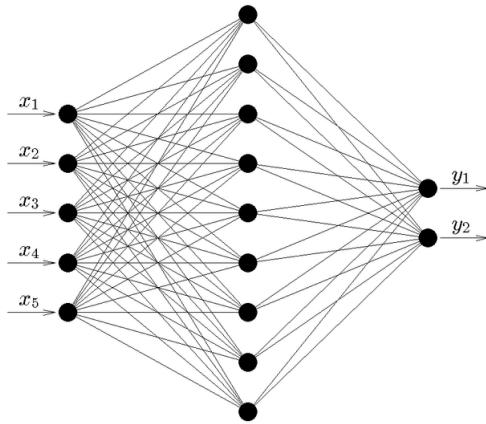


Figure 2.1 A Simple 3-layer perceptron first proposed in 1958 (Rosenblatt, 1962)

2.2 Hopfield Networks

John Hopfield proposed a model of associative memory in his seminal paper in 1982 (Hopfield, 1982). The Hopfield network is a type of recurrent neural network where each neuron is connected to every other neuron except itself. The model consists of a network of N interconnected neurons, each with a binary state $s_i \in \{-1, 1\}$. The state of the entire network is the vector $\mathbf{s} \in \{-1, 1\}^N$. To store a set of P memory patterns, $\{\xi^1, \xi^2, \dots, \xi^P\}$, the synaptic weights are set according to a hebbian rule,

$$\mathbf{W}_{ij} = \frac{1}{N} \sum_{\mu=1}^P \xi_i^\mu \xi_j^\mu \quad \text{for } i \neq j \text{ and } \mathbf{W}_{ii} = 0. \quad (2.5)$$

Borrowed from the physics of spin glasses (Amit et al., 1985; Sherrington and Kirkpatrick, 1975; Mezard et al., 1987), we can define an energy function for the network. This function $E(s)$ is designed such that the stored memory patterns ξ^i are the local

minima in this high-dimensional energy landscape.

$$E(s) = -\frac{1}{2} \sum_{i \neq j} \mathbf{W}_{ij} s_i s_j \quad (2.6)$$

The process of memory retrieval is then an optimization problem of finding the nearest energy minima! This is achieved through the network dynamics. The asynchronous update for a neuron is given by,

$$s_i(t+1) = \text{sgn} \left(\sum_{j=1}^N \mathbf{W}_{ij} s_j(t) \right), \quad (2.7)$$

where sgn is the sign function. The update rule guarantees that the networks energy will either decrease or remain the same with every update. This energy function is a Lyapunov function of the system and will eventually reach a stable fixed point. If the network is initialized in a noisy state of a stored memory, its dynamics will cause the energy to decrease and eventually settle in a basin of attraction corresponding to the complete memory.

However, the classical Hopfield network also has storage capacity limitations. Specifically, for random uncorrelated patterns, the network can reliably store $P = 0.138N$ memories (Amit et al., 1985).

2.2.1 Dense Associative Memories

Krotov and Hopfield (2016) introduced Modern Hopfield Networks, also known as Dense Associative Memories. The key innovation was to generalize the quadratic energy function to a non-linear function, such as an exponential

$$E(s) \propto \exp \left(\sum_{\mu=1}^P (\mathbf{s}^T \xi^\mu) \right) \quad (2.8)$$

This small change increased the storage capacity of the network from linear to exponential in the model dimension N . This modern formulation was then generalized to operate on continuous value states (Ramsauer et al., 2020). The energy function for a modern continuous Hopfield network can be written as,

$$E(\mathbf{q}) = -\text{lse}(\beta \mathbf{K}^T \mathbf{q}) = -\log \left(\sum_{\mu=1}^P \exp(\beta (\xi^\mu)^T \mathbf{q}) \right), \quad (2.9)$$

where the state is a continuous query vector \mathbf{q} and the stored patterns are in the columns of the key matrix \mathbf{K} . *lse* corresponds to the *log – sum – exp* function.

One step of gradient ascent on this potential yields the following update rule,

$$\mathbf{q}' = \text{softmax}(\beta \mathbf{K}^T \mathbf{q}) \mathbf{K} \quad (2.10)$$

where $\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_i \exp(z_i)}$. By introducing a separate value matrix \mathbf{V} that turns each key into its associated output we turn this network from an auto- to a hetero- associative memory. An auto-associative network stores patterns that it can recall from a noisy version of the same pattern. Hetero-associative memory stores pairs of patterns, mapping input keys \mathbf{K} to output values \mathbf{V}

$$\mathbf{q}' = \text{softmax}(\beta \mathbf{K}^T \mathbf{q}) \mathbf{V}. \quad (2.11)$$

With the scaling factor $\beta = 1/\sqrt{d_k}$, where d_k is the key dimension, this is equivalent to scaled-dot-product attention. This reveals that self-attention, the core mechanism in the transformer architecture, is a single step retrieval from a modern, continuous-state Hopfield network with exponential energy and storage capacity (Ramsauer et al., 2020; Hu et al., 2024).

2.3 Sequence-to-Sequence Models

Having established the fundamental connection between associative memories and modern architectures, we broaden our scope. We begin by formalizing the definition of an associative memory, keeping the notation consistent with the modern literature on attention mechanisms. Let \mathbf{M} be the associative memory which varies with time and stores associations. The associative memory at time t therefore is given by \mathbf{M}_t . Any new information to be stored in the memory can be written as a key-value pair $(\mathbf{k}_t, \mathbf{v}_t)$ where key $\mathbf{k}_t \in \mathbb{R}^{d_k}$ represents the content's address, and the value $\mathbf{v}_t \in \mathbb{R}^{d_v}$, represents the content itself. A simple way to "memorize" this information is to directly collect the outer product of the key-value pairs,

$$\mathbf{M}_t = \sum_{i=1}^t \mathbf{k}_t \mathbf{v}_t^\top \quad (2.12)$$

To retrieve these stored associations, we define a function, also referred to as the associative map, $f_M : \mathbb{R}^{d_k} \rightarrow \mathbb{R}^{d_v}$ parameterized by \mathbf{M} such that ideally $f_M(\mathbf{k}_t) = \mathbf{v}_t$. This function is akin to the process of recall in the human brain. For a fault tolerant

system, we want that even for a perturbed key, we still want to still recall the value. Specifically, we want that for a query \mathbf{q}_t close to \mathbf{k}_t , $f_M(\mathbf{q}_t) \approx \mathbf{v}_t$.

The concept of queryable, content-addressable memory is central to understanding the evolution of models designed for sequence to sequence processing. These tasks involve mapping an input sequence $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$ to a target sequence $Y = \{y_1, y_2, \dots, y_T\}$. The sequence-to-sequence framework is remarkably general, and encompasses a vast range of critical problems in machine learning, from machine translation (Sutskever et al., 2014; Bahdanau et al., 2016) and text summarization (Rush et al., 2015) to speech recognition (Chorowski et al., 2015) and protein structure prediction (Senior et al., 2020; Jumper et al., 2021). The primary challenge lies in effectively encoding the information from the source sequence and making it useful for the generation of the target sequence.

2.3.1 Recurrent Neural Networks

One of the first models to demonstrate significant success on sequence modeling tasks were Recurrent Neural Networks (RNNs) (Elman, 1990; Rumelhart et al., 1986). The defining characteristic of an RNN is the use of a hidden state memory \mathbf{h}_t which is updated at each step in the source sequence. This hidden state acts as a compressed representation of all information seen up to that point. Mathematically, the simplest formulation is given by this recurrence relation:

$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t) \quad (2.13)$$

Where, \mathbf{x}_t is the input at timestep t , \mathbf{h}_{t-1} is the memory from the previous timestep, \mathbf{W}_{xh} and \mathbf{W}_{hh} are the learned weight matrices, and ϕ is a non-linear activation function, typically ReLU or hyperbolic tangent. In theory, this recurrence allows information to persist over arbitrary long timescales and thus makes RNNs a natural candidate for modeling sequential data.

However, this theoretical promise was famously difficult to realize in practice due to the challenges of training. The standard algorithm for RNNs, was Backpropagation Through Time (BPTT) (Werbos, 1990; Rumelhart et al., 1986), which involves unrolling the network through the sequence to calculate the gradients. The gradient of the loss with respect to a hidden state early in the sequence, at timestep k is

given by,

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^t \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^t \text{diag}(\phi'(\mathbf{W}_{hh}\mathbf{h}_{i-1} + \mathbf{W}_{xh}\mathbf{x}_i))\mathbf{W}_{hh}^T, \quad (2.14)$$

where $\text{diag}(.)$ represents the diagonal matrix of elements. As we can see, the norm of the gradient is regulated by repeated multiplications of the matrix \mathbf{W}_{hh} . This means that if its singular values are predominantly less than 1, the gradient signal shrinks exponentially, a phenomenon known as the vanishing gradient problem (Bengio et al., 1994; Pascanu et al., 2013). Conversely, if they are greater than 1, then the gradient signal grows exponentially, leading to the exploding gradient problem. Both these issues make training over long sequences practically impossible and the model forgets inputs from earlier in the sequence as the training progresses. These challenges led to the development of gated RNNs (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), which introduced explicit mechanisms to control what information is remembered, what is forgotten and what is passed on to subsequent layers. These innovations transformed the capacity of RNNs to serve as effective associative memory systems of sequential data.

2.3.2 Gated RNNs

Gated RNNs fundamentally address the vanishing and exploding gradient problem by establishing a separate pathway for the flow of information over long sequences. Unlike simple recurrent connections where information is squeezed through a single non-linear transformation at each step, gated units create additive connections that allow gradients to flow more easily.

Long Short-Term Memory (LSTM)

The LSTM network, introduced by Hochreiter and Schmidhuber (1997), was one of the most widely adopted gated RNN architectures. Its core innovation lies in the cell state \mathbf{c}_t , which acts as a residual connection carrying information across many time steps with minimal changes. Information is written or forgotten from the cell state via controlled gates. Each gate is typically a sigmoid neural network layer that has a value between 0 and 1, which is multiplied elementwise with the data, effectively acting as a filter.

An LSTM unit at timestep t takes three inputs, the current input \mathbf{x}_t , the hidden

state from the previous timestep \mathbf{h}_{t-1} and the cell state from the previous time step \mathbf{c}_{t-1} . It then computes an updated hidden state \mathbf{h}_t and a new cell state \mathbf{c}_t . This update mechanism is orchestrated by three gates:

1. Forget Gate \mathbf{f}_t : This gate controls what information from the previous cell state should be discarded. It computes a value between 0 and 1 for each element in \mathbf{c}_{t-1} ,

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \quad (2.15)$$

Where, σ is the sigmoid function, \mathbf{W}_{fx} and \mathbf{W}_{fh} are weight matrices, and \mathbf{b}_f is a bias vector

2. Input Gate \mathbf{i}_t : This gate determines what new information from the current input \mathbf{x}_t and the previous hidden state \mathbf{h}_{t-1} should be stored in the cell state.

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i) \quad (2.16)$$

3. Output Gate \mathbf{o}_t : Finally, this gate determines what part of the current cell state should be exposed as the new hidden state.

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o) \quad (2.17)$$

The update rule of the entire mechanism is modulated by these gates. First we calculate the candidate cell-state:

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{Cx}\mathbf{x}_t + \mathbf{W}_{Ch}\mathbf{h}_{t-1} + \mathbf{b}_C) \quad (2.18)$$

The new cell state is calculated by selectively forgetting from \mathbf{c}_{t-1} and selectively adding from $\tilde{\mathbf{c}}_t$:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (2.19)$$

Where \odot denotes the elementwise hadamard product. This update is crucial and effectively mitigates the vanishing gradient problem.

Finally, the hidden state of the network is updated:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (2.20)$$

LSTMs gained widespread adoption due to their ability to learn dependencies spanning thousands of time steps, a task difficult for vanilla RNNs.

Gated Recurrent Units

The Gated Recurrent Unit (GRU) proposed by Cho et al. (2014) offers a simpler alternative to LSTMs while retaining much of the performance benefits. A GRU combines the forget and the input gates into a single update gate and merges the cell state and the hidden state. This reduces the number of parameters while still being effective in addressing the vanishing gradient problem.

A GRU unit features two main gates:

1. Update gate \mathbf{z}_t : This gate controls how much information from the past hidden state \mathbf{h}_{t-1} should be carried over to the current hidden state \mathbf{h}_t and how much information from the candidate hidden state should be incorporated.

$$\mathbf{z}_t = \sigma(\mathbf{W}_{zz}\mathbf{x}_t + \mathbf{W}_{zh}\mathbf{h}_{t-1} + \mathbf{b}_z) \quad (2.21)$$

2. Reset Gate \mathbf{r}_t : This gate determines how much of the past hidden state should be forgotten. A small value of \mathbf{r}_t means that the previous hidden state is largely ignored.

$$\mathbf{r}_t = \sigma(\mathbf{W}_{rx}\mathbf{x}_t + \mathbf{W}_{rh}\mathbf{h}_{t-1} + \mathbf{b}_r) \quad (2.22)$$

The candidate hidden state is calculated similar to the LSTM, but it applied the reset gate to the previous hidden state,

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \quad (2.23)$$

Finally, the hidden state is a linear interpolation between the previous hidden state and the candidate hidden state, modulated by the update gate,

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \quad (2.24)$$

From an associative memory perspective, LSTMs and GRUs represent a significant advancement. They imbue RNNs with a primitive form of dynamic memory. These gates can be seen as trainable controllers that decide the relevance of past information for the current hidden state. This selective retention of historical data allow the network's internal state to implicitly associate with long term dependencies. However, despite their power, gated RNNs still possess a fundamental limitation. The entire relevant history, no matter how long, must be compressed into a single fixed-dimensional hidden state vector \mathbf{h}_t . While this hidden state with gates is a much richer and more robust summary than in vanilla RNNs, it remains

a bottleneck. The network still lacks the ability to explicitly query and retrieve information from specific past tokens. It cannot dynamically attend to different parts of its input memory. This limitation set the stage for the next major architectural breakthrough, the attention mechanism.

2.3.3 Softmax attention

The fundamental limitation of the recurrent framework was its reliance on a single fixed size hidden state vector to summarize the entire input sequence. This information bottleneck made learning long-range dependencies exceedingly difficult. The breakthrough that removed this limitation was the attention mechanism (Bahdanau et al., 2016; Vaswani et al., 2023). Proposed initially in the context of neural machine translation, attention provided a method for the decoder to dynamically and selectively focus on different parts of the source sequence at each step. The most influential variant of this idea is the scaled dot-product attention, often referred to simply as softmax attention. This mechanism operates on three learned representations derived from each input token \mathbf{x}_t . The Query $\mathbf{q}_t \in \mathbb{R}^{d_k}$, the Key $\mathbf{k}_t \in \mathbb{R}^{d_k}$ and the Value $\mathbf{v}_t \in \mathbb{R}^{d_v}$. Given input tokens $\{\mathbf{x}_1 \dots \mathbf{x}_T\}$, these three vectors are generated via a linear projection with learnable weight matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ as,

$$\mathbf{k}_t = \mathbf{W}_k \mathbf{x}_t, \quad \mathbf{q}_t = \mathbf{W}_q \mathbf{x}_t, \quad \mathbf{v}_t = \mathbf{W}_v \mathbf{x}_t \quad (2.25)$$

The output of the self-attention mechanism at time t , denoted by \mathbf{y}_t is then computed as a weighted sum of all values as,

$$\mathbf{y}_t = \sum_{i=1}^t \mathbf{v}_i \frac{\exp(\mathbf{q}_t^\top \mathbf{k}_i / \sqrt{d_k})}{\sum_{j=1}^t \exp(\mathbf{q}_t^\top \mathbf{k}_j / \sqrt{d_k})} \quad (2.26)$$

Intuitively, the attention mechanism consists of three steps. First, we calculate the similarity score between the current query vector \mathbf{q}_t and the past key vectors \mathbf{k}_i in the history using the dot product. Higher the dot product, higher the alignment or relevance between the query and the key. Second, we normalize these scores and apply a nonlinearity. This normalizing factor of $1/\sqrt{d_k}$ keeps the variance of the dot product of order one. This is crucial for stability, and makes sure that the attention scores don't grow unbounded. These scaled scores are then passed through a softmax function, which converts these scores into a probability distribution over

the input tokens.

$$weight_i = softmax(scores)_i = \frac{\exp(\mathbf{q}_t^\top \mathbf{k}_i / \sqrt{d_k})}{\sum_{j=1}^t \exp(\mathbf{q}_t^\top \mathbf{k}_j / \sqrt{d_k})} \quad (2.27)$$

Finally, we calculate the output \mathbf{y}_t as a weighted sum of the value vectors. Each value \mathbf{v}_i is multiplied by its corresponding weight and the result is a linear combination over all the tokens in the input sequence.

Associative Memory Interpretation

The set of all key-value pairs generated from the input $\{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}$ forms a memory bank. The query acts like a probe into this memory. The mechanism does not retrieve a single memory from the bank, but rather a soft retrieval, returning a linear combination of all stored values, weighted by their score (Zhong et al., 2025). During autoregressive generation (inference), where tokens are produced one at a time, this memory bank is critical for efficiency (Bietti et al., 2023; Pope et al., 2022). For generating the output at timestep $t + 1$, the attention mechanism needs to compute the scores for the query \mathbf{q}_{t+1} and the keys of all preceding tokens, $\{\mathbf{k}_1, \dots, \mathbf{k}_{t+1}\}$. This however consists of a lot of redundant computation. The solution to this redundancy, is that the keys and values for the past tokens can be pre-computed and stored. This store of past keys, and values is known as the key-value Cache (KV Cache).

While generating the $(t + 1)$ -th token, we only need to compute the new key, the new value and append them into the cache. The new query is then used to calculate scores against the entire cached history of keys. This changes the generation process from $\mathcal{O}(T^2)$ to $\mathcal{O}(T)$ for each step. While the overall complexity remains quadratic, this caching drastically reduces the computational cost of inference, making models practical for generating long sequences. The size of the KV cache grows linearly with sequence length as $\mathcal{O}(T.d_k + T.d_v)$ (Brandon et al., 2024).

Significance and Bottleneck

The introduction of softmax attention was revolutionary for several reasons. First, and foremost, it solved the information bottleneck problem. By providing direct connections from any two positions in the sequence, the similarity between any two points, regardless of distance could be calculated. This allows the model to capture extremely long range dependencies with ease. Second, it enabled massive parallelization during training. Unlike an RNN, which must process sequence step-

by-step due to its recurrent hidden state, the attention computation for all queries can be computed simultaneously. Written in terms of matrix multiplications, the attention function can be written as,

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \mathbf{V} \right) \quad (2.28)$$

Where, Q, K, V are matrices formed by stacking the query, key and value vectors. Modern accelerators, particularly GPUs allow for training of vastly large models on larger datasets than ever possible with RNNs.

While we no longer have the bottleneck of learning long range dependencies, softmax attention introduces a new bottleneck: quadratic computational and memory complexity. The core issue being the materialization of the $T \times T$ attention matrix. Both the time and the memory required for this step to scale are $\mathcal{O}(T^2)$. For a sequence of millions of tokens, this is completely intractable. This forces us to ask a critical question; Can we retain the parallelizability of attention while circumventing its quadratic cost? This has motivated a new wave of research into "efficient" transformer alternatives (Yuan et al., 2025; Lu et al., 2025). These models aim to approximate softmax attention with operations that are more favorable with sequence length, ideally linear. This pursuit of scalable yet powerful associative memory mechanisms leads us to consider linear attention, and its many sophisticated variants like Mamba, Gated-DeltaNet etc (Yang et al., 2023, 2024a; Gu and Dao, 2023; Schlag et al., 2021; Behrouz et al., 2024; von Oswald et al., 2025).

2.3.4 Linear Attention

The self-attention mechanism fundamentally reshaped the field of sequence modeling. By providing a direct parallelizable path between all tokens in a sequence, it shattered the limitations of recurrent models and enabled the era of large-scale Transformers. However, the attention mechanism introduces a quadratic complexity in both time and memory with respect to the sequence length T . This $\mathcal{O}(T)$ scaling has become the primary obstacle to applying these models to truly long context domains, such as audio processing and genomic data.

To understand linear attention, it is useful to frame the standard softmax attention from a kernel perspective (Katharopoulos et al., 2020; Qin et al., 2022). The attention mechanism can be viewed as a form of kernel smoothing, where the kernel is

the exponentiated dot product,

$$\text{sim}(\mathbf{q}_t, \mathbf{k}_i) = \exp\left(\frac{\mathbf{q}_t^T \mathbf{k}_i}{\sqrt{d_k}}\right) \quad (2.29)$$

The computational bottleneck arises from the non-linearity of the exponential function. Linear attention proposes a simplification: replace this exponential kernel with a separable kernel, which can be decomposed using a feature map $\phi(\cdot)$ (Kasai et al., 2021; Katharopoulos et al., 2020; Zhang et al., 2024b).

$$\kappa(\mathbf{k}_i, \mathbf{q}_t) = \phi(\mathbf{k}_i)^T \phi(\mathbf{q}_t) \quad (2.30)$$

Where, the function $\phi : \mathbb{R}^{d_k} \rightarrow \mathbb{R}^{d'_k}$ is a nonlinear mapping into the feature space. The simplest choice of the feature map is the identity function $\phi(\mathbf{x}) = \mathbf{x}$ which makes the kernel a pure dot product of the keys and queries. The output of linear attention operation can thus be written as,

$$y_t = \sum_{i=1}^t \mathbf{v}_i \frac{\phi(\mathbf{q}_t^T) \phi(\mathbf{k}_i) / \sqrt{d_k}}{\sum_{j=1}^t \phi(\mathbf{q}_t^T) \phi(\mathbf{k}_j) / \sqrt{d_k}} \quad (2.31)$$

The key advantage of a separable kernel is that it allows us to exploit the associativity to change the order of computation. Thus ignoring the scaling factor,

$$y_t = \sum_{i=1}^t \mathbf{v}_i \frac{\phi(\mathbf{k}_i)^\top \phi(\mathbf{q}_t)}{\sum_{j=1}^t \phi(\mathbf{k}_j)^\top \phi(\mathbf{q}_t)} \quad (2.32)$$

$$= \frac{\sum_{i=1}^t \mathbf{v}_i \phi(\mathbf{k}_i)^\top \phi(\mathbf{q}_t)}{\sum_{j=1}^t \phi(\mathbf{k}_j)^\top \phi(\mathbf{q}_t)} \quad (2.33)$$

$$= \frac{\mathbf{M}_t \phi(\mathbf{q}_t)}{\mathbf{Z}_t \phi(\mathbf{q}_t)} \quad (2.34)$$

where we introduce the \mathbf{M}_t and \mathbf{Z}_t terms to denote the associative memory and the normalization respectively. Note that both \mathbf{M}_t and \mathbf{Z}_t can be computed from the previous time step as,

$$\mathbf{M}_t = \mathbf{M}_{t-1} + \mathbf{v}_t \phi(\mathbf{k}_t)^\top \quad (2.35)$$

$$\mathbf{Z}_t = \mathbf{Z}_{t-1} + \phi(\mathbf{k}_t)^\top \quad (2.36)$$

Instead of computing a $T \times T$ matrix of scores, we compute a $d_v \times d'_k$ state matrix and a d'_k dimensional state vector. The complexity of updating these states is dom-

inated by the outer product and the matrix vector multiplication in the numerator, resulting in a complexity of $\mathcal{O}(d_v d'_k)$, independent of the sequence length T . The total complexity over the complete sequence of length T is therefore $\mathcal{O}(T \cdot d^2)$.

Lets take a closer look at the update mechanism $\mathbf{M}_t = \mathbf{M}_{t-1} + \mathbf{v}_t \phi(\mathbf{k}_t)^\top$ and $\mathbf{Z}_t = \mathbf{Z}_{t-1} + \phi(\mathbf{k}_t)^\top$. This is a direct parallel to the simple associative memory we defined. The matrix M_t is precisely these sum of the outer products of the key-value pairs in feature space. Linear attention, thus is mathematically equivalent to writing to a matrix memory using a classical hebbian update rule. This provides a dual view of this mechanism as a linear version of attention, as well as a linear version of RNNs.

This linear complexity however comes with significant tradeoffs,

1. **Loss of Expressivity:** The exponential kernel is exceptionally good at tasks that require precise recall, such as in-context learning, where the answer must be copied from the past context. The linear dot-product kernel in contrast produces a much more smoother distribution over tokens, making it inherently weaker at such high-precision retrieval tasks (Choromanski et al., 2022; Xiong et al., 2021; Zheng et al., 2023; Arora et al., 2023).
2. **Normalization:** The denominator $\mathbf{Z}_t \phi(\mathbf{q}_t)$ while necessary for averaging the numerator, was found to be a source of major training instability. The denominator can become zero or very small, leading to exploding outputs and gradients. Consequently, many subsequent works (Sun et al., 2023b) chose to drop the normalization term entirely. This simplifies the associative memory update to just $\mathbf{M}_t = \mathbf{M}_{t-1} + \mathbf{v}_t \phi(\mathbf{k}_t)^T$ and the retrieval map to $f_{\mathbf{M}}(\mathbf{q}_t) = \mathbf{M}_t \phi(\mathbf{q}_t)$

A significant body of research focused on mitigating the loss of expressivity by designing a more powerful feature map $\phi(\cdot)$. The goal is the find a function whose dot product in the feature space can approximate the behavior of a non-linear exponential kernel, while remaining decomposable.

To increase expressivity, Katharopoulos et al. (2020) proposed the use $\phi(\cdot) = 1 + \text{ELU}(\cdot)$ to ensure non-negativity and introduce a nonlinearity. ELU is defined as,

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \quad (2.37)$$

Another set of methods draw from the theory of kernel methods. The work on "Performer" by Choromanski et al. (2022) showed that the softmax kernel can be

approximated using random feature maps. This involves projecting the queries and the keys with random matrices and applying trigonometric functions. Other recent works have suggested numerous other alternatives for feature maps such as cosine functions, random features or even learnable feature maps (Qin et al., 2022; Schlag et al., 2021; Zhang et al., 2024b; Peng et al., 2023).

Despite these advancements, linear attention and variants have not managed to universally supplant softmax attention. They often trade some degree of performance and robustness for their efficiency gains. The subsequent wave of research, therefore, moved beyond this to reconsider the recurrent state update itself. Instead of a simple additive accumulation, architectures like state-space models introduce more sophisticated, dynamic update rules for the memory state. We start this discussion with a simple gated version of linear attention.

2.3.5 Gated Linear Attention

Linear attention reformulated attention into a linear recurrent process. This was achieved by defining the associative memory matrix which simply accumulated the key-value outer products. However this introduces a fundamental problem deeply rooted in the history of associative memory, namely memory capacity saturation.

Classical Hopfield networks faces a well known capacity limit. As the number of stored patterns P approaches $P \approx 0.14N$, adding new patterns corrupts the energy landscape where previously stable stored memories are no longer stable attractors. This phenomenon of interference is a direct consequence of the memory matrix \mathbf{M} becoming saturated. The finite size hidden state \mathbf{M} must compress an unbounded history of associations. As the sequence length grows, the state becomes an average of all past key-value pairs, causing individual memories to be lost in the noise of the accumulation.

This necessitates a mechanism for managing the finite memory state, by prioritizing new information and decay the old, potentially irrelevant, information. The inspiration for such a mechanism comes directly from the innovations in gated RNNs.

More specifically, we augment the recurrent update with a learnable forget gate (Yang et al., 2023), which dynamically controls the retention of past information. The memory update is modified to include a multiplicative decay factor $\gamma_t \in [0, 1]$, such that when $\gamma_t \approx 1$, past memories are preserved and when $\gamma_t \approx 0$, they are forgotten. This transforms the associative state from an accumulator into a dynamic memory with exponentially fading history. They way γ_t is implemented defines the fine grained control:

1. **Input dependent Scalar gating:** The most common approach is to make the gate dynamic and data-dependent. Typically, we take the input \mathbf{x}_t , pass it through a linear projection followed by a sigmoid function.
2. **Input dependent vector gating:** A more expressive extension employs a vector gate $\boldsymbol{\gamma}_t \in \mathbb{R}^{d_v}$ for a channelwise gating. The update rules becomes $\mathbf{M}_t = \text{diag}(\boldsymbol{\gamma}_t)\mathbf{M}_{t-1} + \mathbf{v}_t \mathbf{k}_t^T$. This allows different dimensions to have different retention rates enabling the model to maintain long-term memory for some features while rapidly updating others. This update rule is equivalent to discrete time update of Mamba.

While this model provides a much better control of memory dynamics, it still suffers from some limitations. The model can control when and how much to forget, but it cannot decide what to forget. There is no way for the model to update specific contents in the memory. This process of error-driven modification is the principle behind the Delta Rule. Its implementation in modern sequence architectures, known as fast-weight programmers and DeltaNet present the next logical step in the evolution of associative memory, moving beyond simple accumulation and global decay to a more sophisticated content-addressable rewriting.

2.3.6 Fast weight programmers and DeltaNet

Long before the modern Transformers era, (Schmidhuber, 1992) proposed an architecture known as fast weight programmers. The core idea was to create a neural network with two distinct types of weights:

1. **Slow Weights:** The standard parameters of the network, learned slowly via backpropagation over a large training corpus.
2. **Fast Weights:** These are temporary, rapidly changing weights that change within the forward pass for each individual sequence. They serve as a task specific, short term associative memory

In this model, at each timestep, its hidden state is used to generate the fast weights of a separate, typically feed-forward network. These fast weights are often stored as an outer-product memory matrix, much like the state in linear attention. The key difference is that the update to these fast weights is not a simple accumulation but a function determined by the slow-weight controller. This architecture allows the model to effectively learn a rapid learning algorithm. While computationally

expensive and ahead of its time, the FWP concept introduced the powerful idea of a neural network that explicitly modifies its own parameters as part of its forward pass, treating memory updates as a dynamic, learned process (Irie et al., 2021; Schlag et al., 2021).

Delta Rule

The principle of dynamically updating weights based on performance is formalized by one of the most fundamental learning rules in neuroscience and machine learning: the Delta Rule (Schlag et al., 2021). The rule states that the change in a weight should be proportional to the error in the output, corrected in the direction of the input that caused the error. For an associative memory trying to map a key \mathbf{k}_t to the value \mathbf{v}_t , this can be expressed as,

$$\Delta \mathbf{M}_t = (\text{target} - \text{actual}) \otimes \text{input} \quad (2.38)$$

The modern efficient implementation of this in a sequence model is the DeltaNet (Yang et al., 2024b). This operationalizes the Delta rule for the linear associative memory state \mathbf{M}_t . The update follows three steps.

First, the model probes its existing memory \mathbf{M}_{t-1} with the new key \mathbf{k}_t to retrieve the associated value, $\mathbf{v}_{old} = \mathbf{M}_{t-1}\mathbf{k}_t$.

Second, we calculate the error or the "delta" between the correct target value and the retrieved value. This error vector represents the "surprise" or the missing information. $\delta_t = \mathbf{v}_t - \mathbf{v}_{old} = \mathbf{v}_t - \mathbf{M}_{t-1}\mathbf{k}_t$.

Third, the memory is then updated by adding the outer product of this error vector and the key, scaled by a learning rate as,

$$\mathbf{M}_t = \mathbf{M}_{t-1} + \eta_t \delta_t \mathbf{k}_t^T \quad (2.39)$$

$$= \mathbf{M}_{t-1} + \eta_t (\mathbf{v}_t - \mathbf{M}_{t-1}\mathbf{k}_t) \mathbf{k}_t^T \quad (2.40)$$

In a more interpretable form, this can be written as,

$$\mathbf{M}_t = \mathbf{M}_{t-1}(\mathbf{I} - \eta_t \mathbf{k}_t \mathbf{k}_t^T) + \eta_t \mathbf{v}_t \mathbf{k}_t^T \quad (2.41)$$

The first term in (2.41) performs a targeted forgetting operation. Multiplying the memory state with the matrix $(\mathbf{I} - \eta_t \mathbf{k}_t \mathbf{k}_t^T)$ has the effect of removing components of the memory that are aligned with the key \mathbf{k}_t . The second term $\eta_t \mathbf{v}_t \mathbf{k}_t^T$ is the familiar hebbian update rule, which writes new key-value associations into the memory.

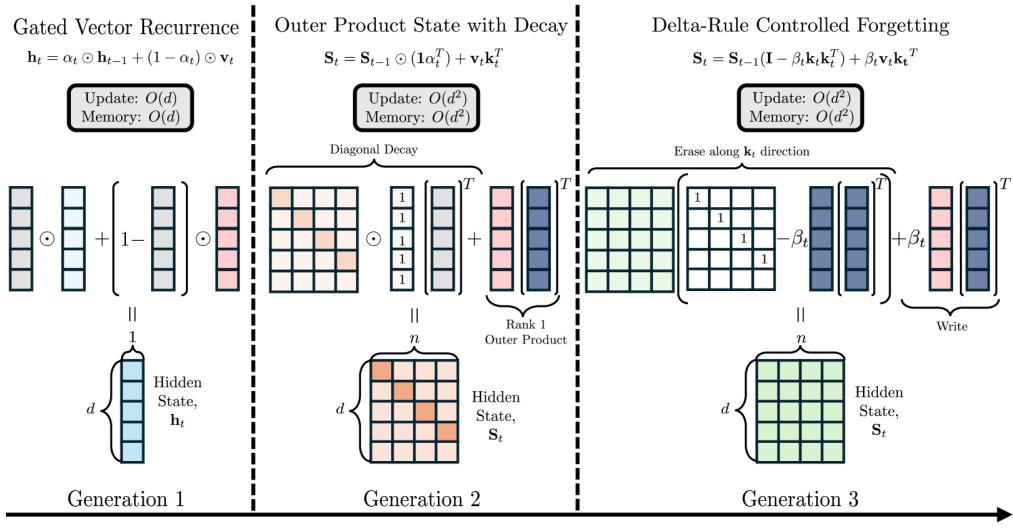


Figure 2.2 Wang et al. (2025a) divides the update rules for linear attention into three generations.

Gated DeltaNet

Reintroducing global control over the entire memory state, we augment the update rule with a global gate γ_t computed from the input x_t which scales the entire update as,

$$\mathbf{M}_t = \text{diag}(\gamma_t) \mathbf{M}_{t-1} (\mathbf{I} - \eta_t \mathbf{k}_t \mathbf{k}_t^T) + \eta_t \mathbf{v}_t \mathbf{k}_t^T \quad (2.42)$$

The gate now acts as a meta controller, deciding the tradeoff between stability and plasticity. Gated DeltaNet (Yang et al., 2024a) thus represents a synthesis of the architectural ideas explored so far. It possesses a sophisticated memory that can perform both global, context-dependent decay (via γ_t) and local, content-addressable rewriting (via the Delta Rule).

This progress from simple outer-product accumulation in Linear attention, to global decay in Gated Linear Attention and finally content-addressable rewriting in DeltaNets illustrates a clear trajectory towards more flexible associative memory system. Each of these architectural update rules, while heuristically motivated, can be seen as algorithmic advancements. This raises a deeper question: what problems are these steps trying to solve? This question forms the bridge to our next section, where we will reframe the entire hierarchy of memory updates through the unifying perspective of online optimization and test-time memorization.

2.4 A unifying perspective

The previous section traced architectural evolution for linear RNNs from simple outer-product accumulation in linear attention, to gated decay and finally to delta rule based update rule. Each innovation can be understood not merely as an ad-hoc engineering solution, but a more sophisticated algorithm for updating an internal model of the world (hidden state). This observation invites a more fundamental perspective, wherein we move from describing the update rules of these models, to defining what they are solving (Behrouz et al., 2024, 2025; Wang et al., 2025d)

The forward pass of any sequence model with memory can be rigorously framed as a sequential online optimization process. At each timestep t , the model receives a new piece of information, the $(\mathbf{k}_t, \mathbf{v}_t)$ pair, and it must update its internal memory state, from \mathbf{M}_{t-1} to \mathbf{M}_t . This update is not arbitrary, it is a solution to an implicit single step optimization problem. This entire process can be encapsulated in a single general objective function following Behrouz et al. (2024),

$$\mathbf{M}_t = \underset{\mathbf{M} \in \mathcal{M}}{\operatorname{argmin}} \underbrace{\mathcal{L}(\mathbf{M}, \mathbf{k}_t, \mathbf{v}_t)}_{\text{Attentional Bias}} + \underbrace{\mathcal{R}(\mathbf{M}, \mathbf{M}_{t-1})}_{\text{Regularizer}} \quad (2.43)$$

This equation provides a powerful and generative framework for understanding and designing sequence architectures. The vast diversity of models seen in the literature emerges from different, principled choices for the three core components of this objective: the memory representation \mathcal{M} , the attentional bias \mathcal{L} , and the retention regularizer \mathcal{R} , as well as the algorithm used to solve the optimization problem.

2.4.1 The memory representation

The set \mathcal{M} defines the hypothesis space for the memory itself. It is the functional form of the associative map, and its choice dictates the model's capacity, expressivity and the complexity.

1. **Parametric, Linear Memory:** The simplest and the most common choice is to represent the memory as a linear operator, typically a matrix $\mathbf{M} \in \mathbb{R}^{d_v \times d_k}$. The retrieval map is simply then, $f_{\mathbf{M}}(\mathbf{k}) = \mathbf{M}\mathbf{k}$. This is the foundation for architectures like linear attention and its variants. The memory capacity is fixed by the dimensions of the matrix, $d_v \times d_k$, making it $\mathcal{O}(1)$ with respect to the sequence length.
2. **Parametric, Non-linear Memory:** To increase expressivity, the memory can

be parameterized by a non-linear function, such as an MLP. Here, \mathbf{M} represents the weights of the MLP and the retrieval map is the forward pass of the MLP, $f_{\mathbf{M}}(\mathbf{k}) = \text{MLP}(\mathbf{k})$. This approach as seen in recent models like TTT-MLP (Sun et al., 2024), and Titans (Behrouz et al., 2024), leverage the expressivity of MLP to learn far more complex key-value relationships. The memory complexity remains $\mathcal{O}(1)$, fixed by the size of the MLP but its capacity is significantly increased.

3. **Non-Parametric Memory:** Another approach is to define the memory by the data itself. In this view, the memory state \mathbf{M}_t is the entire history of key-value pairs $\{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^t$. The map is merely a lookup table from the KV-Cache. This is the implicit choice made by softmax attention. Such models have a memory capacity that grows linearly with the sequence length $\mathcal{O}(T)$. This grants them immense representational power for in-context recall but it comes at the cost of quadratic complexity.

2.4.2 The attentional Bias \mathcal{L}

The attentional bias $\mathcal{L}(\mathbf{M}, \mathbf{k}_t, \mathbf{v}_t)$ is the per-step loss function. It defines the model’s inductive bias for what a good association is. Following we discuss some common choices,

1. **Dot product Objective:** Here, we try to maximize alignment between the retrieved value and the true value.

$$\mathcal{L}_{align} = -\langle \mathbf{M}\mathbf{k}_t, \mathbf{v}_t \rangle \quad (2.44)$$

This objective simply encourages the memory matrix \mathbf{M} to be positively correlated to the key-value outer product. This simple choice leads to the purely additive update of unnormalized linear attention.

2. **ℓ_2 loss:** The most prevalent choice in the literature is the squared Euclidean distance:

$$\mathcal{L}_{\ell_2}(\mathbf{M}; \mathbf{k}_t, \mathbf{v}_t) = \frac{1}{2} \|\mathbf{v}_t - f_{\mathbf{M}}(\mathbf{k}_t)\|_2^2 \quad (2.45)$$

This choice is foundational to classical statistics and is also known as Ordinary Least Squares (OLS) regression (James et al., 2023). It is statistically optimal under the assumption of gaussian noise on the values. Architectures like DeltaNet are a result of applying online gradient descent to this objective.

3. **ℓ_p loss:** A well-known drawback to the ℓ_2 loss is its sensitivity to outliers. A natural extension to this is to use ℓ_p -norm class of objectives. Different values of p gives rise to different properties of the architecture.

$$\mathcal{L}_{\ell_p}(\mathbf{M}; \mathbf{k}_t, \mathbf{v}_t) = \frac{1}{2} \|\mathbf{v}_t - f_{\mathbf{M}}(\mathbf{k}_t)\|_p^p \quad (2.46)$$

2.4.3 Retention Regularizer

The retention regularizer $\mathcal{R}(\mathbf{M}, \mathbf{M}_{t-1})$ is the cost function that penalized changes to the memory state. It is the mechanism that enforces stability and prevents forgetting. We discuss some common choices of regularizers,

1. **ℓ_2 regularizer:** The most common form of regularizer is a quadratic penalty on the deviation from the previous state:

$$\mathcal{R}_{\ell_2}(\mathbf{M}, \mathbf{M}_{t-1}) = \frac{\lambda}{2} \|\mathbf{M} - \mathbf{M}_{t-1}\|_F^2 \quad (2.47)$$

This is the standard proximal term in optimization theory and is equivalent to the weight decay used ubiquitously in deep learning. It encourages the new memory state \mathbf{M}_t to be close to the old one \mathbf{M}_{t-1} with the strength controlled by λ . This term is the origin of the gating mechanism in Gated Linear Attention.

2. **Sparsity regularizer (ℓ_1):** Applying an ℓ_1 regularizer $\mathcal{R}_{\ell_1} = \lambda \|\mathbf{M} - \mathbf{M}_{t-1}\|_1$ induces sparsity on the updates. This is the principle behind LASSO in statistics (Bühlmann and van de Geer, 2011). In the context of sequence models, this would lead to an update where the model learns to modify only a small subset of its internal memory parameters at each step.
3. **KL-Divergence:** For a normalized memory state \mathbf{M} , we can employ a natural measure of the information theoretic distance such as the Kullback-Leibler Divergence $\mathcal{R}_{KL} = \lambda D_{KL}(\mathbf{M} || \mathbf{M}_{t-1})$.

2.4.4 Optimization Algorithm

The final component is the choice of algorithm used to solve or approximate the argmin operator at each time step.

1. **First Order Methods (Gradient Descent):** The simplest and most computationally efficient approach is to take one step of online gradient descent. The

update rule then becomes $\mathbf{M}_t \approx \mathbf{M}_{t-1} - \eta_t \nabla_{\mathbf{M}}(\mathcal{L} + \mathcal{R})$. It is computationally cheap and easy to implement. Nearly all modern efficient RNNs can be interpreted as an instance of this first-order approximation (Cabannes et al., 2024b,a).

2. **Second Order Methods (Newton's Methods):** This is more powerful but expensive alternative to gradient descent. A single step of newton's method becomes $\mathbf{M}_t = \mathbf{M}_{t-1} \mathbf{H}^{-1} \nabla_{\mathbf{M}}(\mathcal{L} + \mathcal{R})$, where \mathbf{H} is the Hessian matrix. The hessian matrix describes the local curvature information and can lead to faster convergence and more stable updates. This however comes at a higher computational cost of $\mathcal{O}(d^3)$ per step for the matrix inversion.

Having established this general framework, we will now demonstrate its utility by deriving the core architectural primitives we discussed in the previous section.

2.4.5 Deriving existing architectures

In this section, we derive some of the existing architectures through the online optimization framework.

Linear Attention as pure Hebbian Learner

We choose an objective that simply seeks to maximize the alignment between the memory state and the new key-value pair. This can be expressed as the negative inner product of the memory matrix and the outer product. There is no explicit regularizer.

$$\mathcal{L} = -\langle \mathbf{M}\mathbf{k}_t, \mathbf{v}_t \rangle \quad (2.48)$$

Taking the gradient of the objective with respect to the memory \mathbf{M} is straightforward,

$$\nabla_{\mathbf{M}} \mathcal{L} = -\mathbf{v}_t \mathbf{k}_t^T \quad (2.49)$$

Taking one step of gradient descent update with a learning rate of $\eta_t = 1$ leads to the desired update

$$\mathbf{M}_t = \mathbf{M}_{t-1} - 1(-\mathbf{v}_t \mathbf{k}_t^T) = \mathbf{M}_t + \mathbf{v}_t \mathbf{k}_t^T \quad (2.50)$$

As is easy to see, the objective function can grow unbounded, specifically the norm fo $\mathbf{M}_t \mathbf{k}_t$ can grow unbounded to decrease the loss and hence might cause issues with stability

Gated Linear Attention as regularized Hebbian Learning

We choose the same alignment function as linear attention, but include a regularizer to counteract the unbounded growth of the memory state. An easy way to counteract this growth is to regularize using the norm of the matrix,

$$\mathcal{R}_{\ell_2}(\mathbf{M}) = \frac{\lambda}{2} \|\mathbf{M}\|_F^2 \quad (2.51)$$

The gradient of this term is $\nabla \mathcal{R}_{\ell_2} = \lambda_t \mathbf{M}$. Combining this with the gradient of the loss within the online update rule:

$$\mathbf{M}_t = \mathbf{M}_{t-1} - \eta_t \nabla_{\mathbf{M}} [\mathcal{L} + \mathcal{R}_{\ell_2}] \quad (2.52)$$

$$= \mathbf{M}_{t-1} - \eta_t (-\mathbf{v}_t \mathbf{k}_t^T + \lambda_t \mathbf{M}_{t-1}) \quad (2.53)$$

$$= (1 - \eta_t \lambda_t) \mathbf{M}_{t-1} + \eta_t \mathbf{v}_t \mathbf{k}_t^T \quad (2.54)$$

Define $\gamma_t = (1 - \eta_t \lambda_t)$ and $\eta_t = 1$. Then the final form of the update rule is $\mathbf{M}_t = \lambda_t \mathbf{M}_{t-1} + \mathbf{v}_t \mathbf{k}_t^T$.

Gated DeltaNet as Regularized Online Regression

Define the learning objective as the squared error between the target value and the value retrieved from memory.

$$\mathcal{L}_{\ell_2} = \frac{1}{2} \|\mathbf{v}_t - \mathbf{M}_t \mathbf{k}_t\|_2^2 \quad (2.55)$$

The gradient of this loss evaluated at the previous timestep \mathbf{M}_{t-1} is,

$$\nabla_{\mathbf{M}} \mathcal{L}(\mathbf{M}_{t-1}) = -(\mathbf{v}_t - \mathbf{M}_{t-1} \mathbf{k}_t) \mathbf{k}_t^T \quad (2.56)$$

To incorporate global forgetting we use the same ℓ_2 regularization as in GLA,

$$\mathcal{R}_{\ell_2}(\mathbf{M}) = \frac{\lambda}{2} \|\mathbf{M}\|_F^2 \quad (2.57)$$

Combining these components in the online gradient descent update yields the Gated DeltaNet rule,

$$\mathbf{M}_t = \mathbf{M}_{t-1} - \eta_t \nabla_{\mathbf{M}} [\mathcal{L} + \mathcal{R}_{\ell_2}] \quad (2.58)$$

$$= \mathbf{M}_{t-1} - \eta_t (-(\mathbf{v}_t - \mathbf{M}_{t-1} \mathbf{k}_t^T) \mathbf{k}_t^T + \lambda_t \mathbf{M}_{t-1}) \quad (2.59)$$

$$= (1 - \eta_t \lambda_t) \mathbf{M}_{t-1} + \eta_t (\mathbf{v}_t - \mathbf{M}_{t-1} \mathbf{k}_t) \mathbf{k}_t^T \quad (2.60)$$

Setting the gate $\gamma_t = (1 - \eta_t \lambda_t)$, and $\eta_t = \beta_t$ we arrive at the Gated DeltaNet update rule,

$$\mathbf{M}_t = \gamma_t(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^T) \mathbf{M}_{t-1} + \beta_t \mathbf{v}_t \mathbf{k}_t^T \quad (2.61)$$

Table 2.1 Table adapted from [Behrouz et al. \(2025\)](#). Provides a overview of recent linear attention variants and how they fit into the framework.

Model	$\ell(\cdot; \cdot)$	Optimizer	Memory Write Operation
Attention	$\sum_{t=1}^L a_t \ M k_t - v_t\ _2^2$	NP	$M_t = M_{t-1} \cup \{(k_t, v_t)\}$
SWA	$\sum_{t=c}^L a_t \ M k_t - v_t\ _2^2$	NP	$M_t = (M_{t-1} \setminus \{(k_c, v_c)\}) \cup \{(k_t, v_t)\}$
Linear Attention	$\langle M_t k_t, v_t \rangle$	GD	$M_t = M_{t-1} + v_t k_t^\top$
RetNet	$\langle M_t k_t, v_t \rangle$	GD	$M_t = \alpha_t M_{t-1} + v_t k_t^\top$
GLA	$\langle M_t k_t, v_t \rangle$	GD	$M_t = \text{Diag}(\alpha_t) M_{t-1} + v_t k_t^\top$
DeltaNet	$\ M_t k_t - v_t\ _2^2$	GD	$M_t = (\mathbf{I} - \beta_t k_t k_t^\top) M_{t-1} + \beta_t v_t k_t^\top$
Gated DeltaNet	$\ M_t k_t - v_t\ _2^2$	GD	$M_t = \alpha_t (\mathbf{I} - \beta_t k_t k_t^\top) M_{t-1} + \beta_t v_t k_t^\top$
Longhorn	$\ M_t k_t - v_t\ _2^2$	Implicit GD	$M_t = (\mathbf{I} - \delta_t k_t k_t^\top) M_{t-1} + (\delta_t \odot v_t) k_t^\top$
RWKV-7	$\ M_t k_t - v_t\ _2^2$	GD	$M_t = (\text{diag}(\alpha_t) - \beta_t k_t k_t^\top) M_{t-1} + \beta_t v_t k_t^\top$
TTT	$\ M_t(k_t) - v_t\ _2^2$	GD	$M_t = M_{t-1} - \eta_t \nabla \ell(M_{t-1}; k_t, v_t)$
Titans	$\ M_t(k_t) - v_t\ _2^2$	GD w/ M	$M_t = \alpha_t M_{t-1} + S_t$ $S_t = \eta_t S_{t-1} - \theta_t \nabla \ell(M_{t-1}; k_t, v_t)$

In the next chapter, we will discuss some common architectural choices and methods for training large scale hybrid attention models.

Chapter 3

Methods

3.1 Architectural Backbone

To fully understand the mechanisms giving rise to computation, we first need to understand the network topology at play. In the past decade, neural network architectures have gone through a sequence of transformations. The transformer architecture (Vaswani et al., 2023) was a turning point in this evolution. Transformer has become the defacto standard for Natural Language processing, Computer Vision etc. The full "LLM" architecture has been at significant interest for the Machine Learning community and has gone through countless iterations. However, a lot still remains to be discovered. Many of these modifications are now seen as essential for a performant language model, particularly but not limited to, layer normalization (Ba et al., 2016; Henry et al., 2020), positional encodings (Su et al., 2023; Press et al., 2022), gated MLPs (Shazeer, 2020) have emerged through years of empirical investigation and ablation studies. Another major front in this search has been the

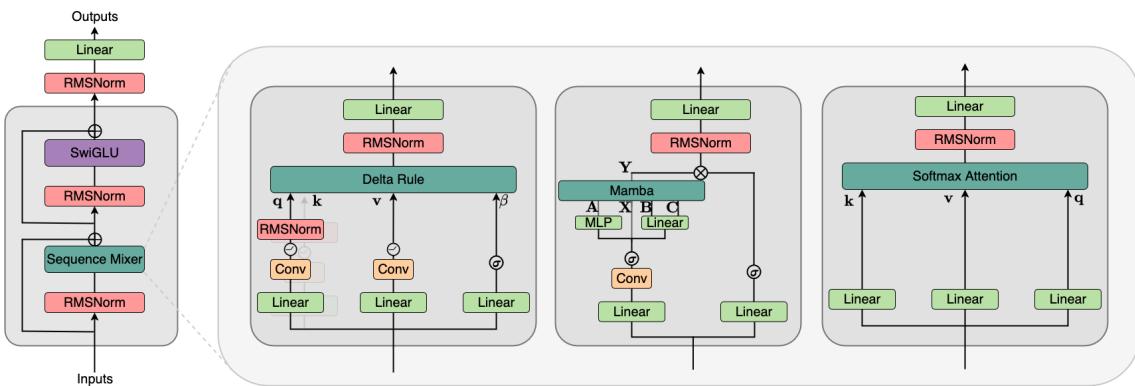


Figure 3.1 An Overview of model architecture and the architectural backbone. (**Left**): Mamba, (**Middle**): Gated DeltaNet, (**Right**): Attention

development of linear attention mechanisms. These architectures had limited impact, but started getting attention after the release of Mamba (Gu and Dao, 2023) in early 2023. Mamba demonstrated that it is possible to scale to long sequences efficiently and introduced a parallel-scan algorithm. Further architectural developments over the past couple of years have given us a variety of architectures and update rules as noted in Chapter 2. What we present here is only a subset of these innovations, and we encourage the reader to continue their exploration beyond the scope of this presentation.

3.1.1 Transformer Architecture

We highlight some of the important modifications to the original transformer architecture here:

3.1.2 Sequence Mixing Layers: Softmax Attention

Multi Head Attention (MHA)

Multi-Head Attention (MHA) was introduced by (Vaswani et al., 2023) as the sequence-mixing primitive in the Transformer architecture. Let $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_T] \in \mathbb{R}^{T \times d_{\text{model}}}$ be the matrix of token embeddings for a sequence of length T . For each head $i \in \{1, \dots, h\}$ the model learns three linear projections $\mathbf{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ that map the embedding space into separate query, key and value subspaces. The projections produce $\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^Q$, $\mathbf{K}_i = \mathbf{X}\mathbf{W}_i^K$ and $\mathbf{V}_i = \mathbf{X}\mathbf{W}_i^V$. A single head computes the scaled dot-product attention

$$\mathbf{O}_i = \text{softmax} \left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d_k}} \right) \mathbf{V}_i \quad (3.1)$$

where the softmax is applied along the rows. The scale factor $1/\sqrt{d_k}$ makes the attention scores unit variance thereby keeping the gradients in a sensible range. The h head outputs are concatenated and linearly recombined,

$$MHA(\mathbf{X}) = [\mathbf{O}_1, \mathbf{O}_2, \dots, \mathbf{O}_h]\mathbf{W}^O, \quad \mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}} \quad (3.2)$$

before adding a residual connection and normalization to pass the result to subsequent feedforward blocks. For input \mathbf{X} the output \mathbf{Y} of a block can be written

as,

$$X_1 = X + MHA(LN(X)) \quad (3.3)$$

$$Y = X_1 + MLP(LN(X_1)) \quad (3.4)$$

Empirical mechanistic interoperability studies (Olsson et al., 2022) have revealed that certain heads, dubbed as "induction heads", learn to copy an n -gram once they have matched its first two occurrences. This online copy operation underpins the emergence of in-context learning and is critical for few shot generalization. (Brown et al., 2020)

Group Query Attention (GQA)

Group-Query Attention is a generalization of the key-value sharing idea in Multi-Query Attention (Shazeer, 2019). In standard MHA each of the H heads have its own set of key and value projections, and so this presents a significant memory bandwidth bottleneck during autoregressive decoding. MQA reduces this bottleneck by letting every head share a single key and a single value. The key-value cache therefore drops from $\mathcal{O}(THd_k)$ to $\mathcal{O}(Td_k)$. GQA interpolates between these two endpoints. Partition the H query heads into G disjoint groups of equal size H/G . All heads in the same group share one key matrix $\mathbf{K}^{(g)} \in \mathbb{R}^{T \times d_k}$ and one value matrix $\mathbf{V}^{(g)} \in \mathbb{R}^{T \times d_v}$ while retaining independent query projections for each head. Therefore, the output of head i is given by,

$$\mathbf{O}_i = \text{softmax} \left(\frac{\mathbf{Q}_i \mathbf{K}^{g(i)^T}}{\sqrt{d_k}} \right) \mathbf{V}^{g(i)} \quad (3.5)$$

Where i belongs to the group $g(i)$. When $G = H$, this reduces to MHA and when $G = 1$ this is equivalent to Multi-Query Attention. Intermediate choices trade a small loss in representational flexibility for a large gain in memory bandwidth. Empirically, the perplexity gap between GQA and full MHA is negligible for moderate choices of G .

Multi-head Latent Attention

MLA from DeepSeek-AI et al. (2024) builds upon the observation that key-value spectra are low rank and thus we can compresses the key-value tensor by projecting it into a lower-rank latent space before the attention weights are applied. The query matrix is as usual $\mathbf{Q} \in \mathbb{R}^{T \times d_k}$ but the keys and values are produced in two steps.

First, we project them down using shared latent matrices, $\mathbf{K}_\ell, \mathbf{V}_\ell \in \mathbb{R}^{T \times r}$ and then projected back up using head specific expansion matrices $\mathbf{E}_i^K, \mathbf{E}_i^V \in \mathbb{R}^{r \times d_k}$. The attention operation can be written down as

$$\mathbf{O}_i = \text{softmax}\left(\frac{\mathbf{Q}_i \mathbf{E}_i^{K^T} \mathbf{K}_\ell^T}{\sqrt{d_k}}\right) \mathbf{V}_\ell \mathbf{E}_i^V \quad (3.6)$$

and the key-value cache during autoregressive decoding scales as $\mathcal{O}(Ld_k)$. This was first introduced in DeepSeek-AI et al. (2024) and reported upto a $4\times$ reduction in memory-bandwidth with negligible loss in perplexity.

3.1.3 Sequence Mixing Layers: Mamba/GDN

As discussed in the previous chapter, GDN and Mamba2 (Gu and Dao, 2023; Yang et al., 2024a) can be thought of as special cases of the associative memory update rule. Figure 3.1 shows the architectural backbone around the specific update rule.

Mamba

Mamba maps an input sequence $\mathbf{x}(t)$ to an output $\mathbf{y}(t)$ via a hidden state governed by a continuous time ODE:

$$\frac{d\mathbf{h}(t)}{dt} = \mathbf{A}(t)\mathbf{h}(t) + \mathbf{B}(t)\mathbf{x}(t) \quad (3.7)$$

$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{h}(t) \quad (3.8)$$

The state matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are dynamically generated from the input sequence. For practical implementation, we convert this ODE to a discretized setting and the resulting formulation is computed efficiently using a parallel scan algorithm (see Appendix A.1).

GatedDeltaNet

As derived on Chapter 2, the memory state \mathbf{M}_t is updated as:

$$\gamma_t \mathbf{M}_{t-1} (\mathbf{I} - \eta_t \mathbf{k}_t \mathbf{k}_t^T) + \eta_t \mathbf{v}_t \mathbf{k}_t^T \quad (3.9)$$

Where γ_t is an input dependent gate. This update explicitly probes the memory with the current key \mathbf{k}_t , computes the retrieval error, and then modifies the memory trace corresponding to that key.

These two architectures, while mathematically distinct, share a common goal: to

create a fixed-state recurrent system that is content-aware and computationally efficient, making them ideal building blocks for our hybrid models.

3.1.4 Other architectural Components

The sequence mixing layer is the heart of a Transformer block, but its function is supported by several other critical components that are essential for learning, stability, and expressivity. These components are largely shared across both standard Transformer and our hybrid architectures.

Feed Forward Network

The second major sub-layer within a Transformer block is the position-wise Feed-Forward Network (FFN). Following the sequence mixing layer, the FFN applies a two-layer Multi-Layer Perceptron (MLP) independently to each token representation in the sequence. In the context of our associative memory framework, the FFN can be viewed as the repository of static, parametric knowledge learned during pretraining. While the sequence mixer handles dynamic, in-context associations, the FFN stores the distilled, general knowledge of the model.

Modern architectures have largely replaced the original ReLU activation in FFNs with Gated Linear Units (GLUs), such as SwiGLU (Shazeer, 2020). The SwiGLU activation is defined as:

$$\text{SwiGLU}(x) = (x\mathbf{W}_1) \otimes \text{SiLU}(x\mathbf{W}_2) \quad (3.10)$$

where $\text{SiLU}(z) = z \cdot \sigma(z)$ and \otimes is element-wise multiplication. The gating mechanism, which introduces a multiplicative interaction between two parallel linear projections, has been empirically shown to significantly improve model performance. This FFN structure is a core component of our architectural backbone and, in our distillation experiments, its weights are directly transferred from the teacher to the student model.

Short Convolutions

A crucial insight from the development of modern recurrent models is the importance of local context. The projections that create the query, key, and value vectors in a standard Transformer are purely point-wise (Allen-Zhu and Alfarano, 2025). The representation of the input is mapped to $(\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t)$ without any information from its immediate neighbors. This can be a limitation for linear recurrent models, which

process information strictly sequentially.

To address this, architectures like Mamba and our hybrid models incorporate a short 1D causal convolution as a preliminary step before the main sequence mixing layer. The input sequence \mathbf{X} is first passed through a short convolutional kernel (of size 2, 4). This allows the representation for each token to be a blend of itself and its immediate predecessors. The resulting vectors that are then projected into keys, values, and queries are thus already locally contextualized. This simple addition has been shown to be a critical ingredient for enabling linear-time models to effectively process complex, information-dense sequences like natural language, and it is a standard component of the recurrent blocks used in our experiments.

Positional Embeddings

Without positional information, the attention mechanism cannot distinguish between shuffled and unshuffled sequences. The original transformer architecture uses fixed sinusoidal embeddings where for position p and where each position p is mapped to a vector of sinusoidal functions of different frequencies.

$$PE_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d}}\right), \quad PE_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d}}\right) \quad (3.11)$$

This choice allowed for extrapolation to unseen sequence lengths as the embeddings were not tied to a learned index. However, as we will see later, positional embeddings while necessary make length extrapolation more difficult.

Recently rotary positional embeddings (RoPE), introduced in (Su et al., 2023) have become increasingly popular. RoPE modifies the self attention mechanism by rotating the query and key vectors based on their position. RoPE introduces position information by rotating vectors in a way that the resulting dot product encode relative positions.

Assume that the hidden dimension d is even. We split the \mathbf{K}, \mathbf{Q} vectors into $d/2$ pairs s.t

$$x = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \dots] \Rightarrow [(\mathbf{x}_1, \mathbf{x}_2), (\mathbf{x}_3, \mathbf{x}_4) \dots] \quad (3.12)$$

Each of these pairs can be seen as vectors in \mathbb{R}^2 and we define the rotation operation as,

$$Rot_\theta(x_1, x_2) = (x_1 \cos \theta - x_2 \sin \theta, x_1 \sin \theta + x_2 \cos \theta) \quad (3.13)$$

Define a frequency vector $\omega = (\omega_1, \omega_2, \dots, \omega_{d/2})$ where $\omega_i = 10000^{-2i/d}$. Then for position p the rope transformation on a query or key vector is

$$RoPE(x, p) = \bigoplus_{i=1}^{d/2} \text{Rot}_{\theta_{p,i}}(x_{2i-1}, x_{2i}). \quad (3.14)$$

We can see that the dot product satisfies,

$$\langle RoPE(x, p), RoPE(y, q) \rangle = \langle x, RoPE(y, q - p) \rangle \quad (3.15)$$

The attention score between the positions p and q only depend on their relative offsets. It also preserves the euclidean norm since rotation is an orthonormal operation.

Normalization

Normalization methods play a key role in ensuring stable forward pass and gradients. This plays a key role in accelerating convergance and improving generalization. Here we discuss some widely adopted normalization strategies.

- **LayerNorm:** Originally proposed in (Ba et al., 2016) for recurrent neural networks, LayerNorm has found its place in most modern LLMs. Here, we normalize across the feature dimension independently for each sample ensuring consistent layer statistics during forward and backward passes. Given a vector $\mathbf{x} \in \mathbb{R}^d$,

$$\text{LayerNorm}(\mathbf{x}) = \gamma \cdot \frac{\mathbf{x} - \mu}{\sigma} + \beta, \quad \mu = \frac{1}{d} \sum_{i=1}^d x_i, \quad \sigma = \sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2 + \epsilon} \quad (3.16)$$

Where γ, β are trainable parameters and ϵ is a small constant for numerical stability. It is easy to see that LayerNorm stabilizes the variance of activations, thereby preventing vanishing or exploding gradients.

- **RMSNorm:** Root Mean Square norm is a slight simplification of LayerNorm by removing the mean subtraction. i.e

$$\text{RMSNorm}(\mathbf{x}) = \gamma \cdot \frac{\mathbf{x}}{\text{RMS}(\mathbf{x})}, \quad \text{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2 + \epsilon} \quad (3.17)$$

This removes computational overhead and maintains similar or improved per-

formance. Llama3 (Grattafiori et al., 2024) uses RMSNorm before attention and before MLP layer.

- **QKNorm:** QKNorm specifically addresses instabilities in the attention computation by normalizing query and key vectors,

$$\mathbf{Q}' = \frac{\mathbf{Q} - \mu_Q}{\sigma_Q}, \quad \mathbf{K}' = \frac{\mathbf{K} - \mu_K}{\sigma_K} \quad (3.18)$$

Qwen3 (Yang et al., 2025) uses RMSNorms before attention and MLP and uses QKNorm inside the attention calculation.

Activation Functions

Activation Functions increase the expressive power of a model by transforming the input using a nonlinear function.

- **GELU:** Gaussian Error Linear Unit (Hendrycks and Gimpel, 2023) is one of the most common activations used with transformers. It is defined as,

$$GELU(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2} \left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right) \right] \quad (3.19)$$

Where Φ is the standard gaussian CDF and erf is the error function. This activation is smooth, differentiable and non-monotonic and exhibits curvature at all points. This increased curvature and non-monotonicity allows GELU to approximate complicated functions than ReLUs or ELUs.

- **Swish:** The swish function is a modification of sigmoid and the swish family was designed to smoothly interpolate between a linear function and the ReLU function. SwiGLU combines the gating mechanism of Gated Linear Units (GLU's) (Shazeer, 2020) with the swish activation function, defined as,

$$SwiGLU(x, y) = \text{Swish}(x).y, \quad \text{Swish}(x) = x \cdot \sigma(\beta x) \quad (3.20)$$

Where σ is the sigmoid function and β is the learnable parameter.

3.2 Training Datasets and Evaluation Metrics

Since we are focused on understanding the reasoning differences in hybrid architectures, we train the models on a mixture of math datasets. For all the 150M

models, we use a 1:1 mixture of OpenMathInstruct2 (Toshniwal et al., 2024) and MetaMathQA (Yu et al., 2024). We pretrain on question answer datasets by concatenating the prompt and the answer. We do not use any chat template or special formatting during pretraining. For evaluation, we generally report test accuracy on the test split of GSM8K, MATH500 (Cobbe et al., 2021; Hendrycks et al., 2021).

3.2.1 Training Datasets

More information for these datasets with problem-solution examples are provided in Appendix ??

OpenMathInstruct2

OpenMathInstruct (Toshniwal et al., 2024) consists of 14M problem-solution pairs generated from the GSM8K and MATH Datasets and generated by Llama-3.1-405B-Instruct (Grattafiori et al., 2024).

MetaMathQA

MetaMathQA is bootstrapped from GSM8K and MATH500 training datasets. These questions are diverse in their reasoning directions (Yu et al., 2024).

OpenThoughts-114k-math

Guha et al. (2025) consists of 89K generations from DeepSeek-AI et al. (2025) on questions from Numinamath. These generations are usually much longer ranging from 1K-30K.

3.2.2 OpenR1-Math-220k

Similar to OpenThoughts, these are also generated from DeepSeek-AI et al. (2025) on Numinamath-1.5.

3.2.3 Evaluation Datasets

GSM8K

Introduced in (Cobbe et al., 2021), it has become the most widely used benchmark to understand mathematical capabilities of language models. The test set consists of 1319 problems of grade school math.

MATH500

This dataset is a subset of 500 problems from the MATH benchmark (Hendrycks et al., 2021).

3.3 Pretraining

To understand architectural tradeoffs, we pretrain 150M models of various hybrids percentages (See 4.1) and various architectures like Mamba, GDN, Transformers. For this, we use the OLMo (OLMo et al., 2025) codebase released by Allen Institute for AI. For Mamba2 and Gated DeltaNet we use FlashLinearAttention (FLA) (Yang and Zhang, 2024) for accurate and fast triton implementations. We integrate these architectures within the OLMo codebase and implement our own caching mechanism to handle the combination of caches. Each attention layer keeps its own KV-Cache, that grows with sequence length. Each Mamba/GDN layer keeps its own copy of cache with `recurrent_state` and `conv_state`. We sweep over a variety of architectural choices and report the results of the sweep in Sections ?? and 4.1.5. For attention, we use fast kernels from FlashAttention2 (Dao, 2023). Also see Appendix A.1 for more information on the Chunkwise training of these linear attention architectures as implemented in FLA.

First, we tokenize OpenMathInstruct2 and MetaMathQA using the Llama 2 (Touvron et al., 2023) tokenizer. The reason behind this choice is the comparatively smaller vocabulary of 32K. Since our models are small (150M), we keep the vocabulary size small. The total number of non embedding parameters for pure attention models is 137.8M. We use an mlp dimension of $8 \times d_{model}$ and use RoPE (Su et al., 2023) for all attention layers. For Mamba/GDN, we do not use positional embeddings since these architectures already represent positional information in its sequential processing. We use a max sequence length of 2048 for training and use the AdamW optimizer (Loshchilov and Hutter, 2019). We use a cosine schedule for learning rate with $t_{warmup} = 5000$ and $lr = 0.001$. Further learning rate sweeps are given in 4.1.4. We use the Kempner Cluster for training all the models, and each training run takes about 30 Hours on 2 H100 GPUs.

3.4 Cross-architecture distillation

Training large language models is a costly endeavor and requires significant capital and human effort. Even modest sized Llama (Grattafiori et al., 2024) like Llama-

Table 3.1 Default hyperparameters for 150M models

Parameters	Values
Model dimension	768
Number of Layers	12
Number of Heads	12
Number of Key-Value Heads	12
MLP Ratio	8
Max Sequence length	2048
Embedding Size	32000
Activation Type	SwiGLU
Positional Embedding	RoPE
RoPE Theta	10000
Optimizer Type	AdamW
Adam Betas	(0.9, 0.95)
Learning Rate	0.001
Weight Decay	0.1
Tokenizer	Llama-2-7b

3.1-8B required 1.46 Million H100 hours of training time. Bigger models in the same family, like Llama-3.1-405B used about 30.9 Million H100 hours of training. Training LLMs now use an everincreasing amount of energy, and now require significant amount of new infrastructure to meet this demand. Besides energy constraints, training models is a huge engineering effort, making sure that the training runs are stable and without any loss spikes.

Over the course of the last few years, the research community has figured out tricks to make transformers stable during training. While these techniques have improved the quality of training runs significantly, it remains to be seen if these techniques transfer over to other types of architectures discussed here. Training Mamba/GDN etc architectures has been a pain-point in the community and hence it becomes harder to scale these architectures in terms of model size and training flops.

To overcome these challenges, and to effectively leverage the immense investment in existing open-source models, we develop and employ a multi-stage cross-architecture distillation framework. The central goal is to transfer the sophisticated reasoning capabilities of a large, pretrained Transformer model (the "teacher") into an inference efficient hybrid architecture (the "student") that uses linear recurrent layers. Our methodology closely follows (Wang et al., 2025b), and combines an extra stage for attention score alignment. This allows us to create powerful and scalable models

at a fraction of the computational cost of pretraining from scratch. Our methodology is a four-stage pipeline that combines attention transfer, attention alignment, knowledge distillation and supervised fine-tuning.

3.4.1 Overview and Previous Work

The core hypothesis behind these approaches is that a significant portion of a language model’s learned knowledge is stored in the non-attention part of the network, specifically the feed-forward networks (FFNs), which act as memories. The sequence mixing layers are responsible for dynamic processing and routing of this information in context. Therefore, it should be possible to replace the computationally expensive self-attention mechanism with a more efficient recurrent alternative, while persevering the parametric knowledge stored in the rest of the network.

This idea of converting transformers into RNNs was first introduced in Kasai et al. (2021) where the authors proposed a swap-then-finetune procedure. Specifically, they replaced the softmax attention with a linear-complexity RNN with a learned feature map and then finetuned the entire model. Mercat et al. (2024) proposed Scalable UPtraining for Recurrent Attention (SUPRA) to uptrain to its recurrent formulation. While these methods require full-model updates to recover the initial quality. Zhang et al. (2025) combined attention transfer and another stage of LoRA finetuning to rapidly recover quality.

Zhang et al. (2025); Wang et al. (2025c) demonstrated that initializing a hybrid model by directly copying the FFN and embedding weights from a pretrained Transformer provides a massive head start. Building on this, Wang et al. (2025b) proposed M1, a comprehensive framework for transferring reasoning capabilities from a Transformer to a Mamba-based model using distillation and finetuning. Our method adapts this paradigm into a four stage process, omitting the final RL stage to focus purely on the transfer of reasoning capabilities through distillation and supervised learning.

3.4.2 Four stage transfer procedure

Our framework systematically transfers knowledge from a 1.3B parameter pretrained Llama model (Grattafiori et al., 2024) to a hybrid model of roughly the same size. The student architecture is similar to the teacher, but with some of the layers replaced with a linear sequence mixer, such as Mamba and GatedDeltaNet, and its backbone.

Stage 1: Knowledge Transfer

The first stage provides the student model with a strong foundation of pretrained knowledge. Instead of starting with a randomly initialized student, we perform a direct weight transfer from the teacher model.

- The weights for all non-attention layers, including token and positional embeddings, the FFNs and the lm-head are copied directly from the teacher to the student.
- For each replaced layer, we copy the key, query and value weight matrices $\mathbf{W}_k, \mathbf{W}_q, \mathbf{W}_v$, directly to its recurrent alternative.
- All the other parameters newly introduced, such as gating, convolutions etc are randomly initialized.

This architectural seeding provides the student with the vast repository of parametric knowledge learned by the teacher during its extensive pretraining. The subsequent training stages can therefore focus on the more targeted task of teaching the new recurrent layers how to process and route this information effectively, rather than learning everything from scratch.

Stage 2: Attention Alignment

Similar to [Zhang et al. \(2025\)](#), we use (3.22) to mimic attention logits in our linear RNN model. Specifically, attention alignment step refers to closely approximating softmax attention using a linear attention module. For each head and layer,

$$\mathbf{y}_n = \underbrace{\sum_{i=1}^T \frac{\exp(\mathbf{q}_n^T \mathbf{k}_i / \sqrt{d_k})}{\sum_{i=1}^T \exp(\mathbf{q}_n^T \mathbf{k}_i / \sqrt{d_k})} \mathbf{v}_i}_{\text{Softmax Attention}} \quad \hat{\mathbf{y}}_n = \underbrace{\sum_{i=1}^T \frac{\phi(\mathbf{q}_n)^T \phi(\mathbf{k}_i)}{\sum_{i=1}^T \phi(\mathbf{q}_t)^T \phi(\mathbf{k}_i)} \mathbf{v}_i}_{\text{Linear Attention}} \quad (3.21)$$

for all $n \in \{1, \dots, T\}$ where T is the sequence length. We train the feature maps (or gates for GLA) to minimize the mean squared error,

$$\ell_2 = \frac{1}{MH} \sum_{m=1}^M \sum_{h=1}^H \ell_{MSE}^{h,m} \quad \ell_{MSE}^{h,m} = \frac{1}{d} \|\mathbf{y}_n - \hat{\mathbf{y}}_n\|^2 \quad (3.22)$$

where h represents the head, and m the layer.

Stage 3: Knowledge Distillation

The second stage aims to teach the student's recurrent layers to mimic the full predictive behavior of the teacher's powerful self-attention mechanism. This is achieved through knowledge distillation. The student model is trained on the pretraining corpus (OpenMathInstruct2) using logits from the teacher model on the same dataset. The distillation loss minimizes the Kullback-Leibler (KL) divergence between the students and the teachers probability distribution. The teacher model, having been trained on a massive corpus produces a rich distribution over the vocabulary. The student model is trained to replicate these soft labels from the teachers.

The KL-Divergence from a distribution \mathcal{Q} to a distribution \mathcal{P} is an information theoretic measure of how different one probability distribution is from a second probability distribution. For a vocabulary size V , it is defined as

$$D_{KL}(\mathcal{P} \parallel \mathcal{Q}) = \sum_{i=1}^V \mathcal{P}(i) \log \left(\frac{\mathcal{P}(i)}{\mathcal{Q}(i)} \right) \quad (3.23)$$

Intuitively, it represents the "information gain" achieved if we revise our beliefs from a prior distribution \mathcal{Q} to the posterior distribution \mathcal{P} . It is important to note that KL-Divergence is not a metric since it is not symmetric. Meaning for any two probability distributions $D_{KL}(\mathcal{P} \parallel \mathcal{Q}) \neq D_{KL}(\mathcal{Q} \parallel \mathcal{P})$. The choice of these two measures, has implications on our training, and we discuss those here.

- **Forward KL Divergence:** $\mathcal{L}_{Forward} = D_{KL}(P_{Teacher} \parallel P_{Student})$

Minimizing the forward KL divergence is often described as "mode covering". To make this loss smaller, the student distribution is heavily penalized if it assigns low probability to any token where the teacher assigns a non-zero probability. This forces the student's distribution to cover all the modes (peaks) of the teacher's distribution. The resulting student tends to cover a larger distribution over the vocabulary, but might be uncertain.

- **Reverse KL Divergence:** $\mathcal{L}_{Reverse} = D_{KL}(P_{Student} \parallel P_{Teacher})$

Minimizing the Reverse KL divergence is described as "mode seeking". The student is heavily penalized if it assigns a nonzero probability to a token where the teacher assigns a near zero probability. This forces the student to have zero probability in regions where the teacher's distribution is small. The student therefore is incentivized to focus its probability mass on the high probability modes of the teacher distribution.

In this framework, we use the reverse KL divergence since we want the student to learn the teacher’s high probability modes. To provide a richer training signal, it is standard practice in knowledge distillation to use a temperature parameter T . The logits from the teacher and the student are divided by T before the softmax is applied.

$$P_{Student}(i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \quad (3.24)$$

Stage 4: Supervised Finetuning

After the distillation stage, the student has learned to approximate the teacher’s general language modelling capabilities. Since our focus is on models that perform mathematical reasoning, we perform a final stage that hones its abilities to perform multi-step reasoning and instruction following.

We use the standard paradigm of next-token prediction using the cross-entropy loss defined as,

$$\mathcal{L}_{CE}(\mathbf{y}, \mathbf{p}) = - \sum_{i=1}^V \mathbf{y}_i \log(\mathbf{p}_i) \quad (3.25)$$

Minimizing the cross-entropy loss is equivalent to maximizing the log-probability of the correct next token. Similar to the Knowledge Distillation stage, we use Open-MathInstruct2 for 2 epochs for a maximum sequence length of 8192. Other hyperparameters used in the entire procedure are given in Table 3.3.

At the conclusion of this stage, the student model is fully trained: a computationally efficient hybrid architecture endowed with the distilled knowledge of a large Transformer and finetuned for specifically for mathematical problem solving. The Knowledge Distillation and SFT stages each take around 24 Hours on 4 H100 GPUs for a meta-llama/Llama-3.2-1B (Dubey et al., 2024) teacher model.

3.5 Scaling inference-time compute

The traditional scaling laws (Kaplan et al., 2020) that have guided the development of large language models have primarily focused on scaling pretraining-compute. The scaling laws state that the performance predictably improves by increasing model size, dataset size and training duration. However, as the costs of pretraining reach astronomical levels and returns begin to diminish, a new and complementary paradigm has emerged: scaling inference-time compute. This approach increases the model’s effective intelligence and reasoning capabilities can be significantly enhanced

Table 3.2 Hyperparameters for cross-architecture distillation on 1.3B models

Parameters	Values
Teacher Model	Llama-3.2-1B-Instruct
Model dimension	2048
Number of Layers	16
Number of Heads	32
Number of Key-Value heads(GQA)	8
Embedding Size	131072
MLP Size	8192
FFN Type	swiglu
Positional Embedding Type	rope
RoPE Theta	500000
Distillation Loss Type	Revere KL
Optimizer Type	AdamW
Adam Betas	(0.9, 0.95)
Learning Rate (Distillation)	5e-6
Learning Rate (SFT)	1e-5
Weight Decay	0.1
Train Sequence Length	8192
Dataset (Distillation + SFT)	OpenMathInstruct2
Number of Epochs (Distillation)	1
Number of Epochs (SFT)	2

Table 3.3 Layers Replaced during cross-architecture distillation on meta-llama/Llama-3.2-1B-Instruct

Model Name	Layers Replaced
Llama 1B	None
Mamba/GDN 25	3,7,11,15
Mamba/GDN 50	1,3,5,7,9,11,13,15
Mamba/GDN 75	1,2,3,5,6,7,9,10,11,13,14,15
Mamba/GDN 87_5	1,2,3,4,5,6,7,9,10,11,12,13,14,15
Mamba/GDN 100	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

not just by making the model itself larger, but by allowing it to "think longer" and explore multiple reasoning paths at the time of solving a problem.

This paradigm shifts the computational burden from training the model, to the recurring cost of inference. Scaling inference-time compute equips language models with continuous train of thought that leads to exploration of different avenues, verify intermediate steps and consolidate our findings, akin to humans. These techniques can be broadly categorized into two families, parallel methods, which explore many independent lines of thought, and sequential methods, which iteratively refine a single line of thought.

Parallel Scaling

Parallel methods (Snell et al., 2024; Brown et al., 2024) are highly parallelizable and involve generating a large number of independent solutions to the same problem and then aggregating them to produce a final answer. The underlying assumption is that while the model may make many errors in a single path, multiple processes will equip the model with stronger consensus. N candidate solutions are generated by sampling the models output distribution, typically with a non-zero temperature. There are multiple aggregating methods that can be used

- **Maj@k:** We aggregate the final answers from all the N paths and the final answer is the most frequently occurring.
- **Best-of-N:** When an external verifier exists, typically another language model or an answer-checker, we can assign a confidence score to each answer. The final answer is the best one, as decided by the verifier. Weighted voting extends this method by weighing each answer with a confidence score and then calculating the confidence score across similar answers.
- **Pass@k:** It measures the probability that atleast one correct solution exists within a set of k generated samples. It directly quantifies the effectiveness of parallel sampling and is also called as coverage.

Sequential Scaling

While parallel methods explore breadth, sequential methods (Muennighoff et al., 2025; Xie et al., 2024; Kumar et al., 2024; Wei et al., 2022) explore depth. They are based on the idea of iterative refinement, where the model generates a solution, reflects upon its own work, and then uses that reflection to generate a better solution.

This process transforms a single forward pass into a multi-step reasoning loop. For this to be effective, two key capabilities are required:

- **Verification:** The same model (or an external model) must be able to act as a critic or verifier. After generating an initial solution, it is prompted again, this time to analyze its own work, check for logical fallacies, verify calculations, and provide structured feedback.
- **Refinement:** The model must then be able to take its original prompt, and its own critique as a combined input, and generate a new, improved solution that addresses the identified flaws.

Need for inference-efficient architectures

The primary obstacle to the widespread exploration and adoption of these methods is the immense computational cost when using the standard Transformer architecture. Generating N independent samples requires running N separate generation processes. In transformers, the memory required for the KV Cache for each sample grows linearly with the sequence length T , and in total is therefore $\mathcal{O}(N \times T)$. Sequential methods are even more demanding. The quadratic scaling in sequence length increases the computational burden of longer and longer chain of thoughts. Linear Recurrent architectures have a fixed state size, independent of the sequence length. Generating a sequence of length T only requires $\mathcal{O}(1)$ memory for the state. The time to generate next token is also independent of the generated sequence length in contrast to transformers. This combination of constant memory and constant time per step makes linear RNNs the ideal architectural backbone for scaling inference-time compute. The development of a performant hybrid model, as detailed in our methodology, is therefore a direct step toward building a system capable of leveraging these state-of-the-art reasoning strategies in an efficient and scalable manner.

Chapter 4

Results

4.1 Pretrained models

In this section we discuss results related to pretraining 150M models of various hybrids and its evaluation on GSM8K and MATH500 datasets. All models are trained for 4 epochs on a mixture of OpenMathInstruct and MMQA, for a total of 37.1 Billion tokens. We also present various hyperparameter ablations.

4.1.1 Transformer Baseline

For comparison, we first pretrain a baseline Transformer model.

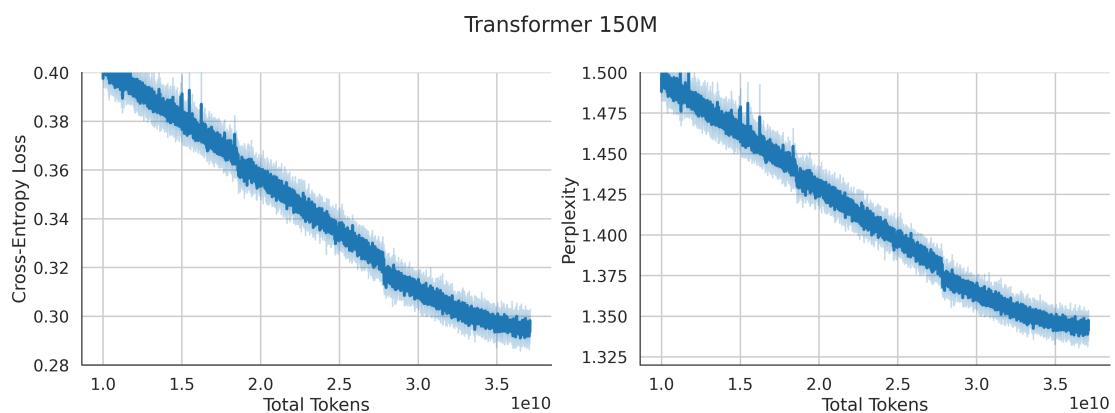


Figure 4.1 (Left): Cross-entropy loss during training, **(Right):** Perplexity during training. The sudden drops represent epoch changes since we repeat the data for 4 epochs. Note that the x-axis starts from 10B Tokens.

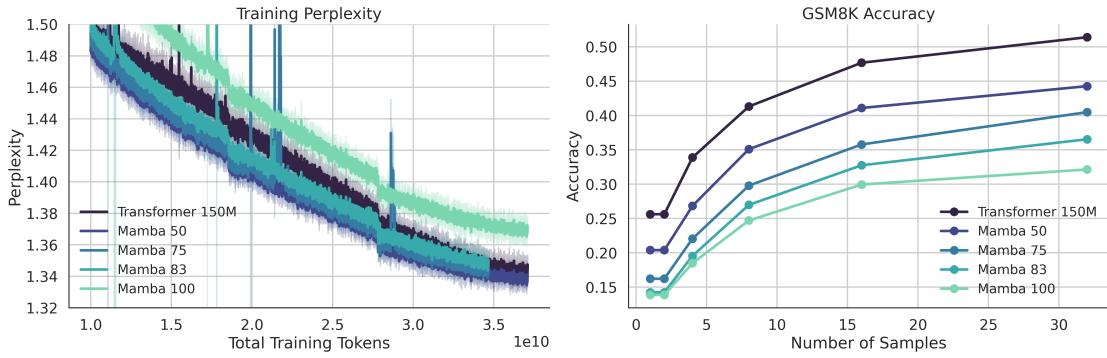


Figure 4.2 (Right) Training perplexity for Mamba Hybrid models. (**left**) Majority Accuracy on GSM8K.

Table 4.1 The pretrained models have 12 Layers. For each hybrid type, these are the default values for layers with attention. We follow the usual striped design (Lieber et al., 2024; Glorioso et al., 2024; Ren et al., 2024). Note that the layers mentioned are full attention

Model Name	Attention Layers
Mamba/GDN 50	1,3,5,7,9,11
Mamba/GDN 75	3,7,11
Mamba/GDN 83	5, 11
Mamba/GDN 100	None

4.1.2 Mamba

Figure 4.2 shows the accuracy of our pretrained mamba hybrid models on GSM8K with majority voting to scale test-time compute. Across the board, higher attention model shows better GSM8K performance. As we will see in the next section, our cross-architecture distilled models show a similar trend. In Figure A.1, we present the Loss and Perplexity plots for Mamba-Transformer Hybrid models. For each of these models, we try multiple hyperparameters and plot the one with lowest perplexity at the end of training., see 4.1.5 for other sweeps.

4.1.3 Gated DeltaNet

Figure 4.3 shows the accuracy on GSM8K for Gated DeltaNet hybrid models. These show an interesting trend where 2-3 layers of interleaved attention show the highest performance. As we will see in Figure 4.12, our cross architecture distilled models also show a similar interesting trend. These results show that it is possible to develop

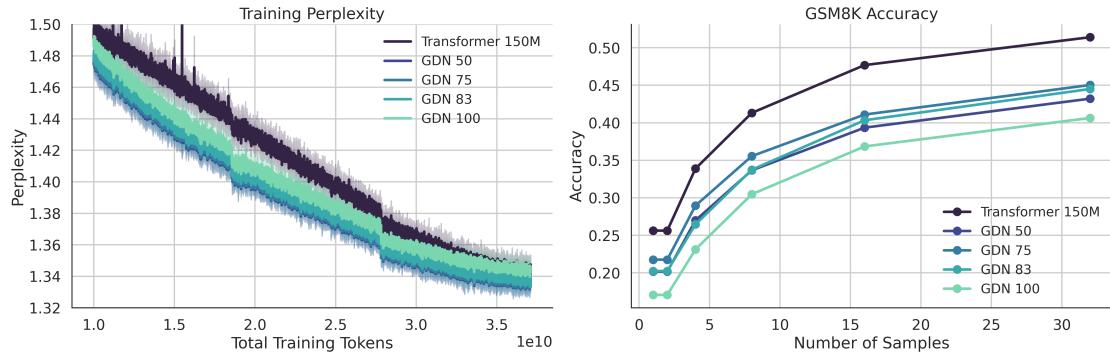


Figure 4.3 (Right) Training perplexity for Gated DeltaNet Hybrid models. **(Left)** Majority Accuracy on GSM8K.

hybrid models who reason similar to pure attention models using Gated DeltaNet.

4.1.4 Ablation Studies: Transformer

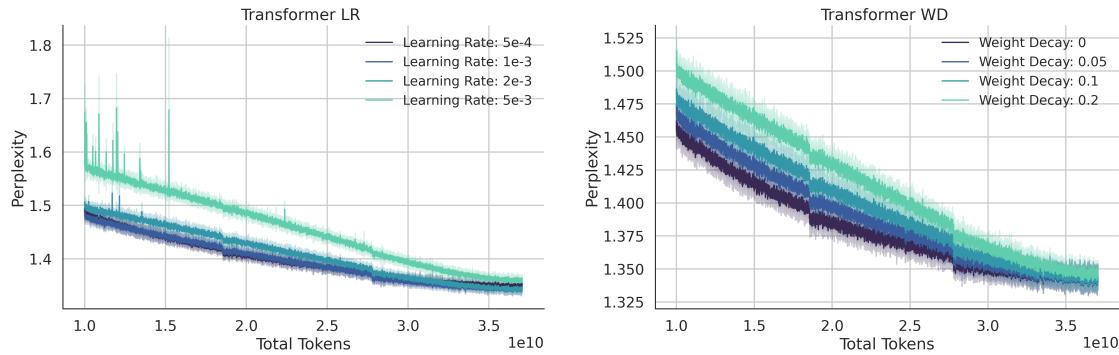


Figure 4.4 Learning rate and Weight decay sweep for Transformer 150M model. Higher learning rate makes the training unstable. The best performing model was chosen with the lowest perplexity and corresponds to $lr = 0.001$ and $wd = 0.1$

4.1.5 Ablation Studies: Gated DeltaNet and Mamba

To further understand hyperparameter choices, we present ablation studies. For all the hyperparameters not mentioned, assume default values given in 3.1.

4.2 Cross architecture distillation

In this section we explore the test time scaling of our cross-architecture distilled models. We see a similar trend in Mamba of more attention leading to better per-

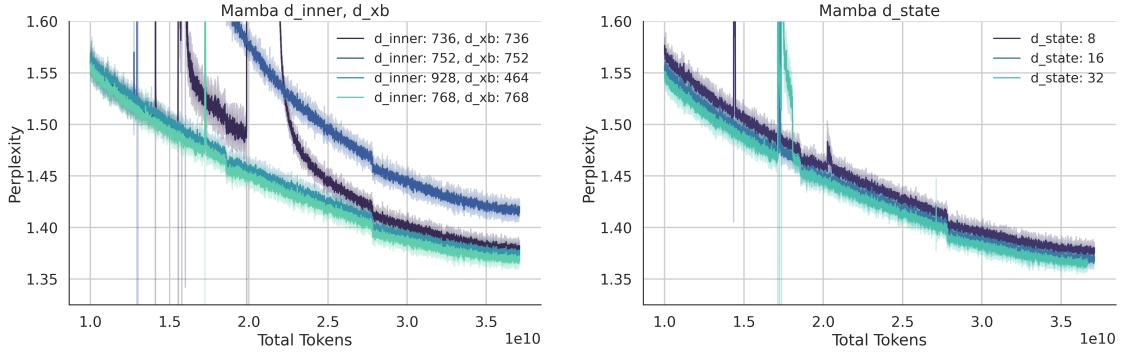


Figure 4.5 For mamba models, (Right) we sweep over some reasonable choices of d_{inner} and d_{xb} . Note that these choices are independent of the model dimension (768). (Left) The SSM state dimension. Mamba models are very susceptible to spikes.

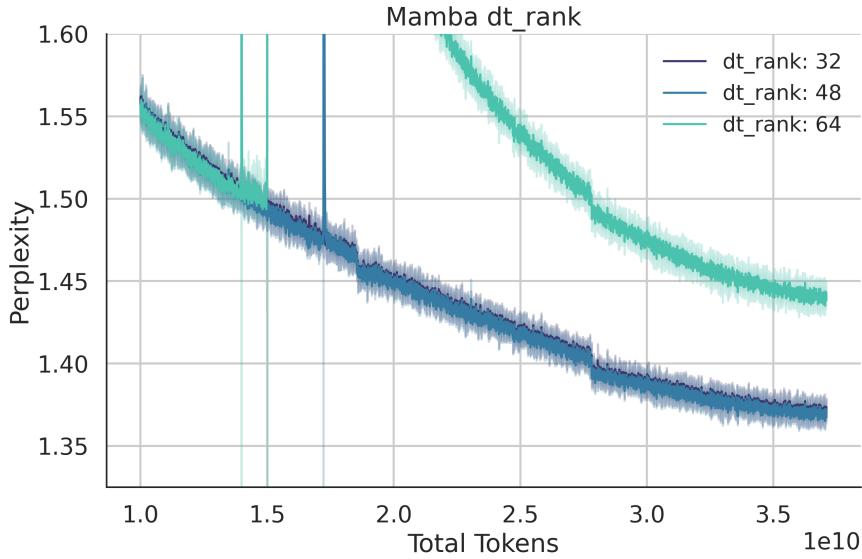


Figure 4.6 To understand why spikes occur in Mamba models, we tried changing the dt_rank parameter. Intuitively this controls the information written to the hidden state. We find conservative values of dt_rank reduce the instabilities. Related, Zuo et al. (2025) reported that clipping dt to positive values also help with instabilities.

formance and scaling. Gated DeltaNet on the other hand has the unusual scaling, similar to results from pretrained models. We also use Needle In a Haystack ([Kamrardt, 2023]) benchmark to test the long context capabilities of our distilled models. With any percent of attention, there is a sudden dropoff in quality after its training length. A pure linear attention model performs poorly but does not show this sudden drop off. This highlights the long-context evaluation tradeoffs for these models

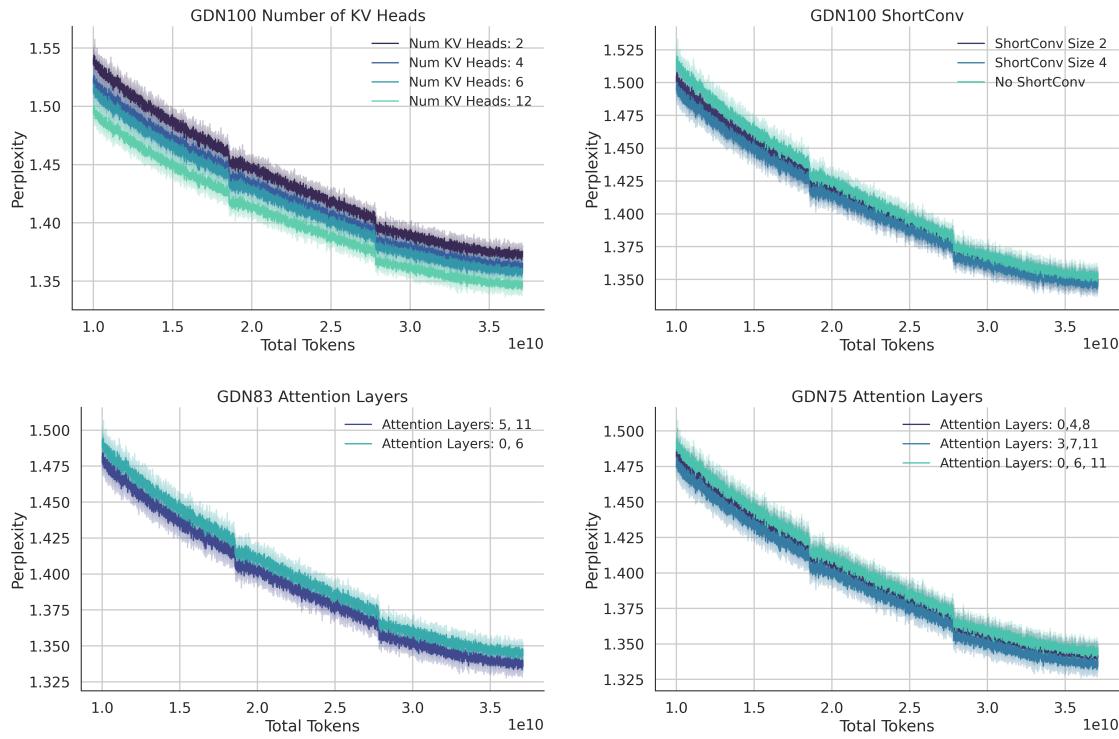


Figure 4.7 (Top Left) Sweep over the number of KV heads. The number of Query heads are always 12. **(Top Right)** ShortConv is an important piece in the architectural backbone of Gated DeltaNet. We show that the size of the convolution does not matter, but it is important to retain reasoning capabilities. **(Bottom Left)** For Gated DeltaNet 83, we try various permutations of inserting attention Layers. Attention in the last layer (11) greatly improves performance, also seen in **(Bottom Right)** for GDN73.

4.2.1 Gated DeltaNet and Mamba

Figures 4.8 and 4.10 represents an interesting phenomenon. While designing hybrid models, we aim to increase efficiency while keeping the performance intact. Practitioners usually make a tradeoff on the performance gap that they are okay with and design their models. Gated DeltaNet results show that there might be an ideal architectural layout choice that gives us maximum efficiency while maintaining reasonable performance. We hope to explore these architectural details and their properties further in our future work. The next section focuses on the representational difference between these architectures.

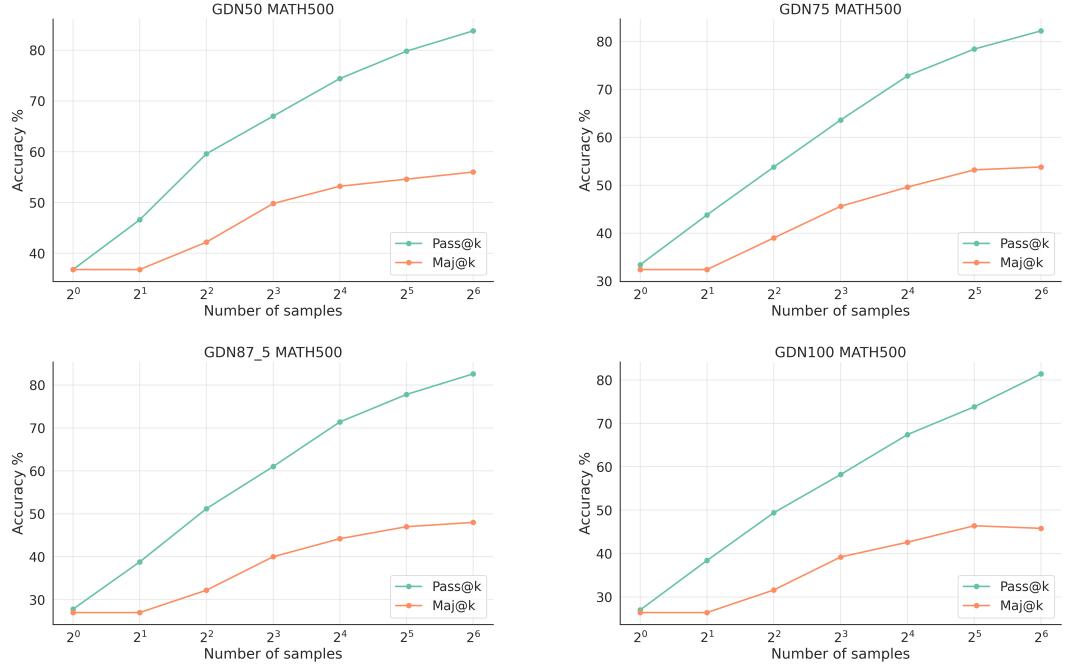


Figure 4.8 Our Cross-architecture distilled Gated DeltaNet Hybrids on MATH500 with Pass@k and Maj@k. The teal plot is pass@k, and shows that the model coverage is high. 100 percent GatedDeltaNet also performs fairly well.

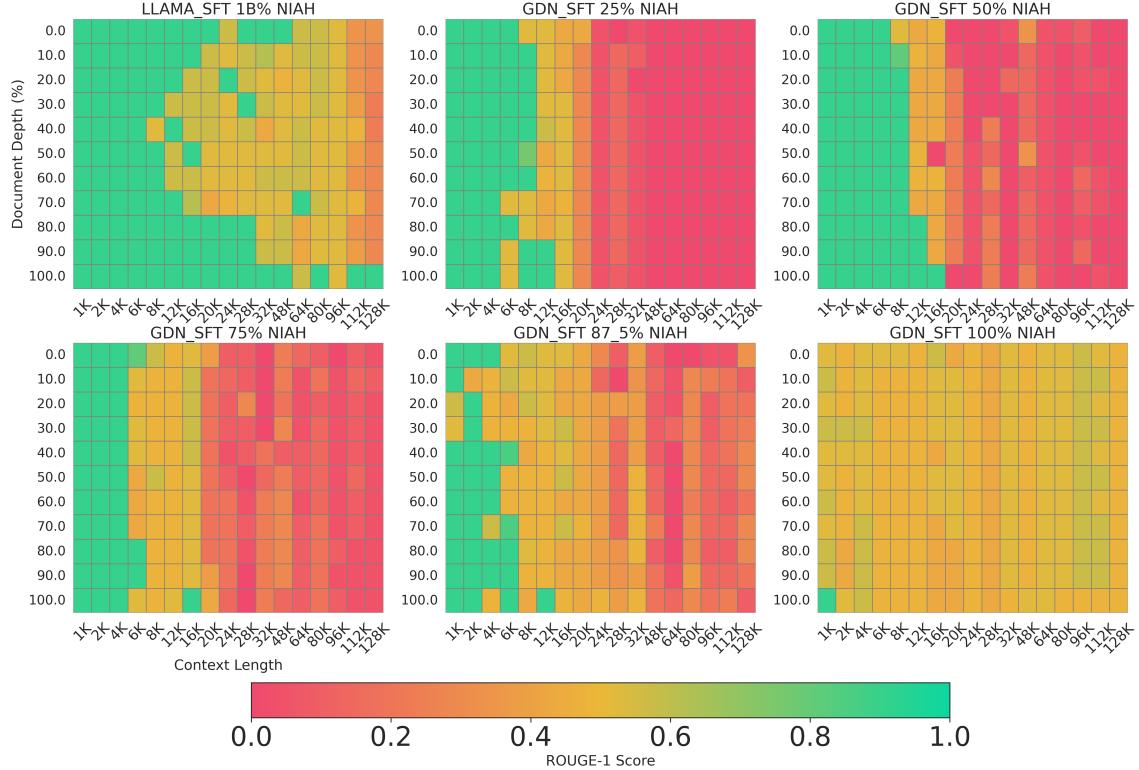


Figure 4.9 Needle in a Haystack plots for GatedDeltaNet and Llama-1B. .

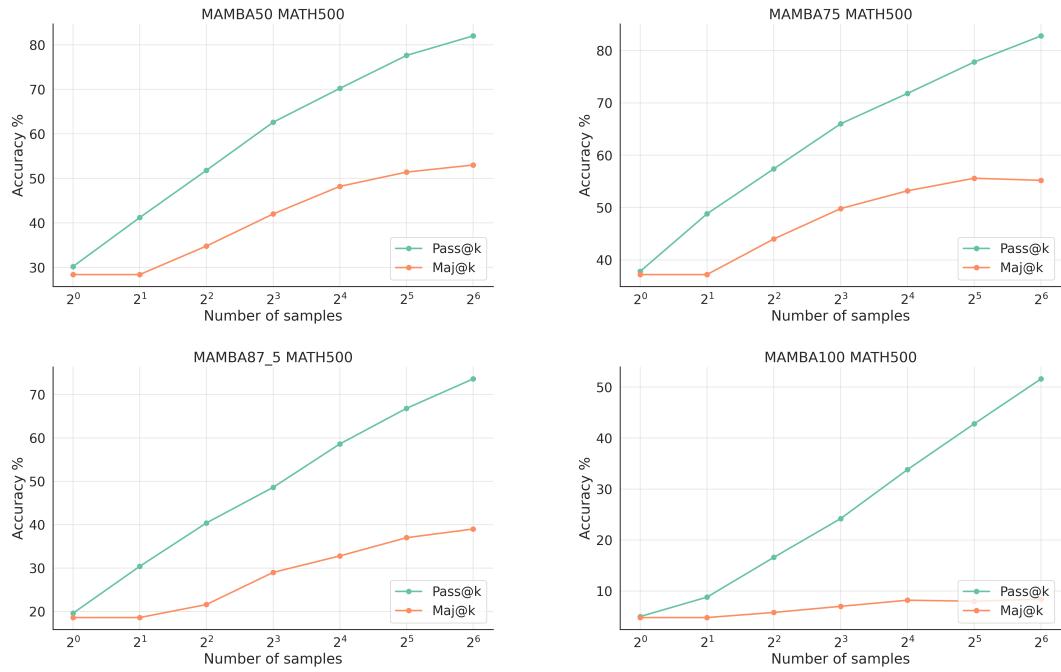


Figure 4.10 Mamba Hybrids on MATH500 with Pass@k and Maj@k

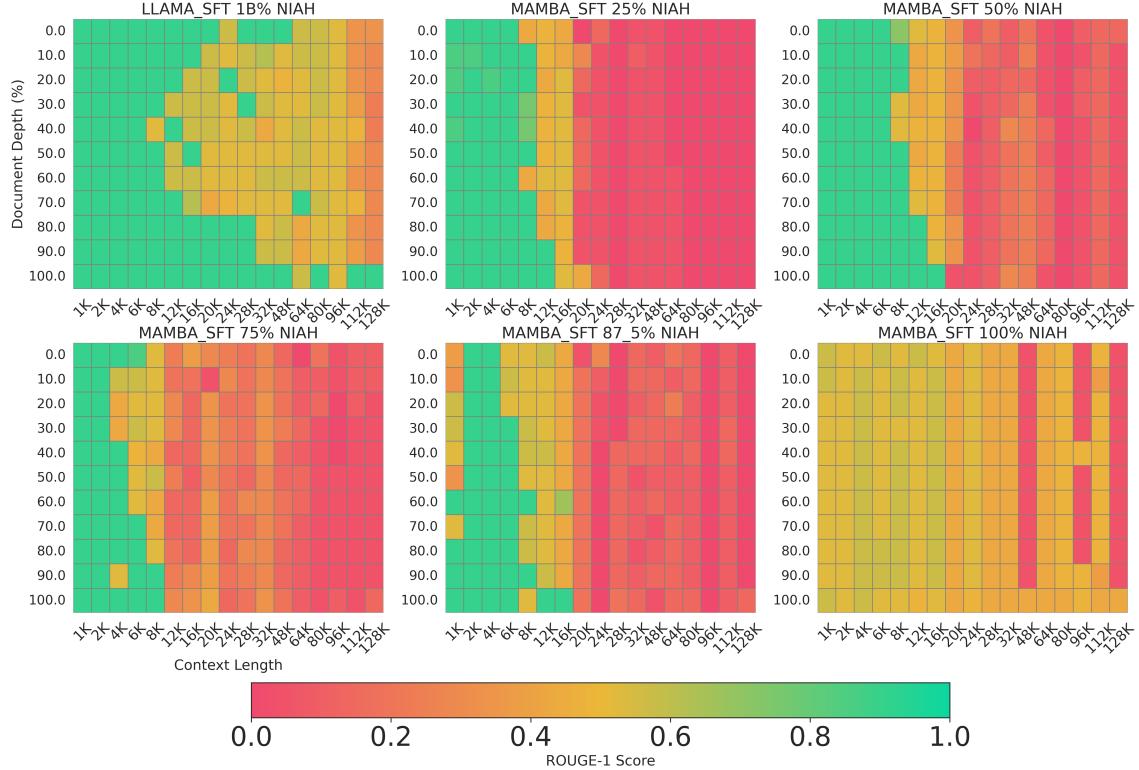


Figure 4.11 Needle in a Haystack plots for Mamba hybrids and Llama-1B. Similar to Figure 4.9 we see a sudden drop off in long context retrieval outside the training context length for models with any amount of attention.

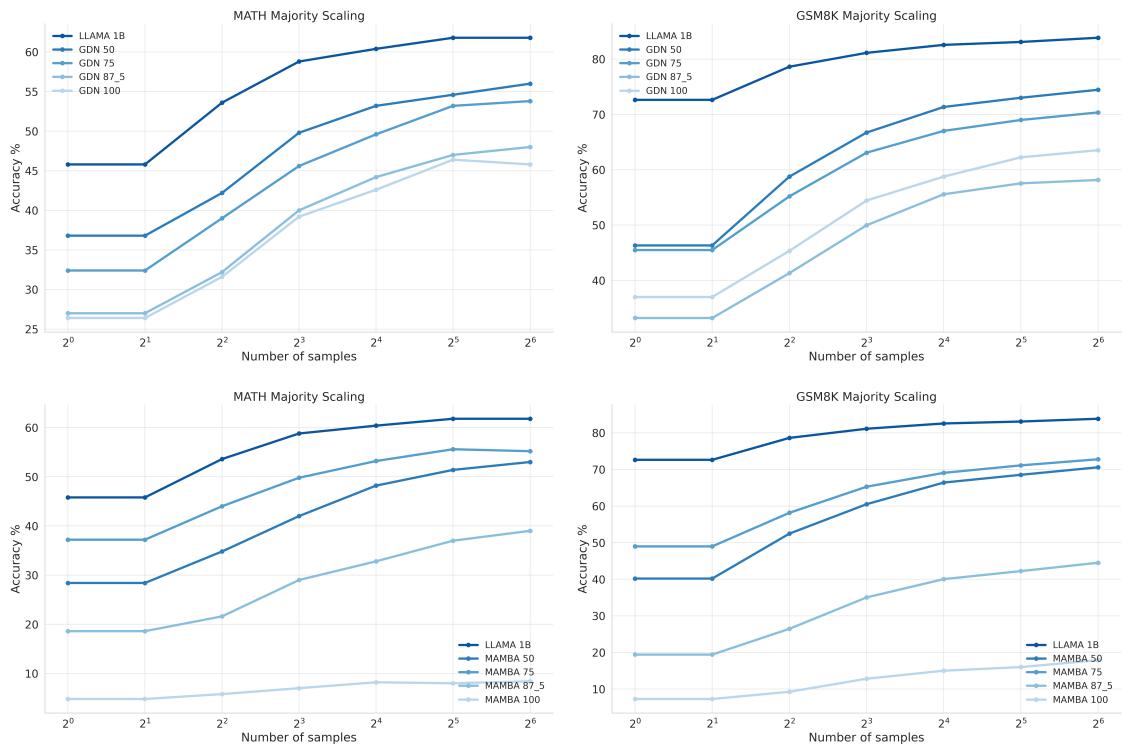


Figure 4.12 Combined Scaling plots for MATH500 and GSM8K on all cross-architecture distilled hybrid models. (**Top**) Gated DeltaNet and (**Bottom**) Mamba.

4.3 Understanding Representations

The theoretical framework of this thesis posits that different sequence mixing layers correspond to distinct associative memory update algorithms. While we have evaluated these architectures on downstream task performance, this tells us what they can do, but not how they do it. A deeper understanding requires peering inside the models to observe how they dynamically organize information in response to context. To investigate this, we use the in-context graph tracing task from Park et al. (2025) to probe and visualize the formation of internal representations.

4.3.1 In-context Graph Tracing

We randomly select a set of concepts (tokens) and form a grid that is unrelated to the semantics between the tokens. For eg, a set of common nouns (e.g., "apple", "bird", "car") are chosen as nodes. The model is then fed a long context consisting of examples of random walk on this graph. Park et al. (2025) showed the emergence of representations mirroring the grid-like structure in the internal activations of the models when visualized using PCA. The paper posits that standard transformer models exhibit a emergent property where the representations undergo a sudden phase transition and allows the model to build novel "world models" from in-context examples. We apply the same methodology to our distilled hybrid models to investigate how the balance between attention and mamba layers affect this capability. Specifically to calculate mean token representations \mathbf{h}_i from the last layer. For every token $i \in \mathcal{T}$ in our graph, we grab the activation per token in the most recent context window of $N_w = 50$ tokens. We then compute the mean activation per token and stack the together to obtain $\mathbf{H}^\ell(\mathcal{T}) \in \mathbb{R}^{n \times d}$. We then run PCA on this matrix and use the first two principle components to visualize model activations.

We find that models with more attention layers is able to form in-context representations but they appear less crisp as we reduce the amount of attention. We hypothesize that self-attention is the primary mechanism for the in-context forming of novel relations due to its ability to perform all-to-all comparisons. The fixed-state recurrent layers, while efficient and performant, lack the ability to construct these de-novo representations from scratch. This is not to say that recurrent layers cannot learn structure, but that their mode of learning may be different. This provides a strong, mechanistically grounded argument for why hybrid architectures are so

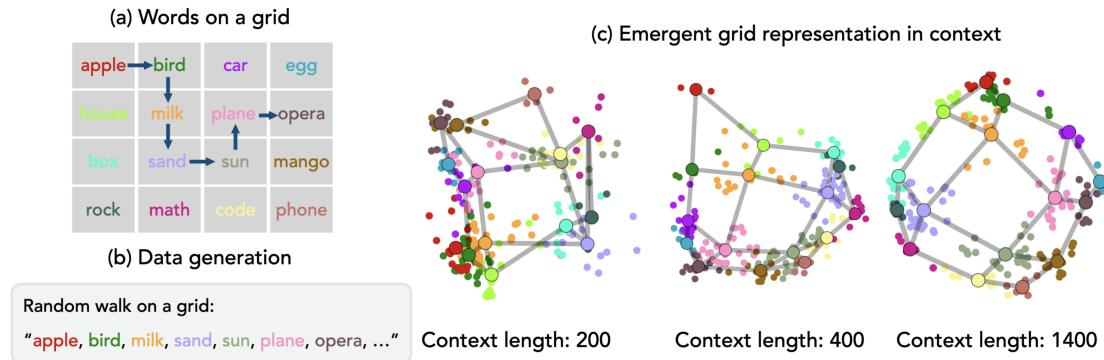


Figure 4.13 Fig 1 from Park et al. (2025) explaining the task and visualizations for Llama 3.1-8B. We use the same methodology of plotting the model’s mean token representation onto the top two principle components.

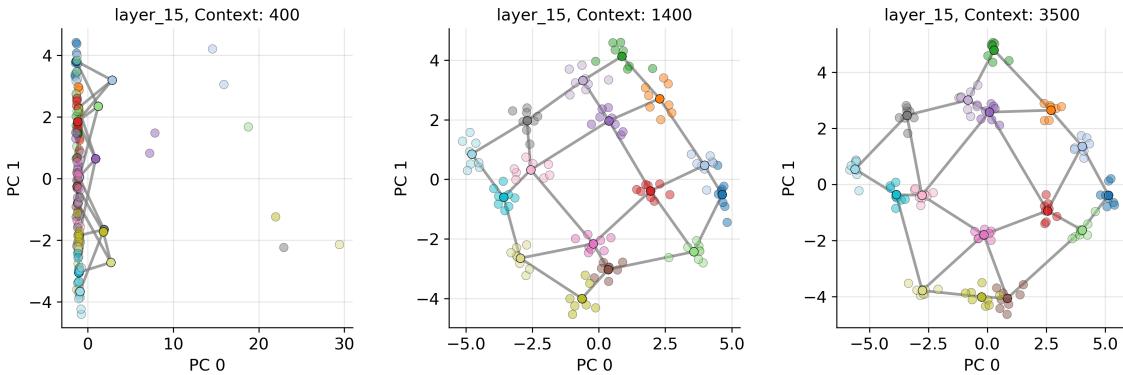


Figure 4.14 PCA visualization of last layer of Llama 1B with different context length presented.

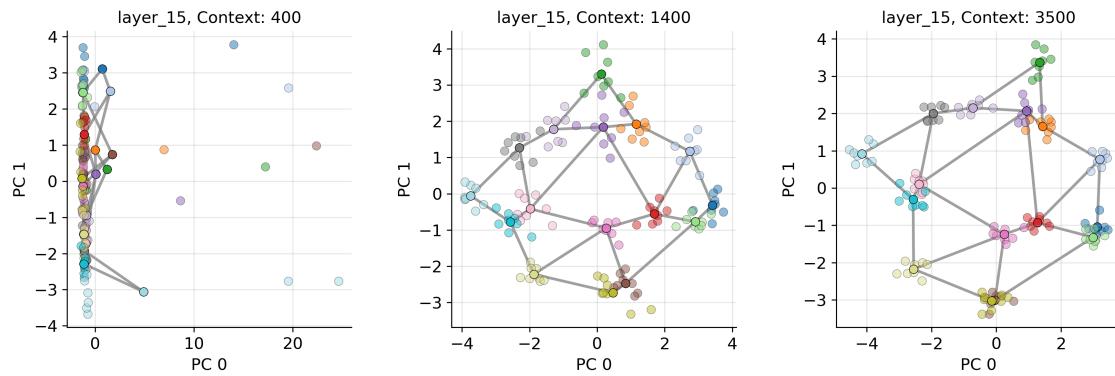


Figure 4.15 PCA visualization of 25% Mamba: 75% Attention

compelling: they offer the potential to combine the rapid, in-context reasoning and representation-building capabilities of attention with the computational efficiency

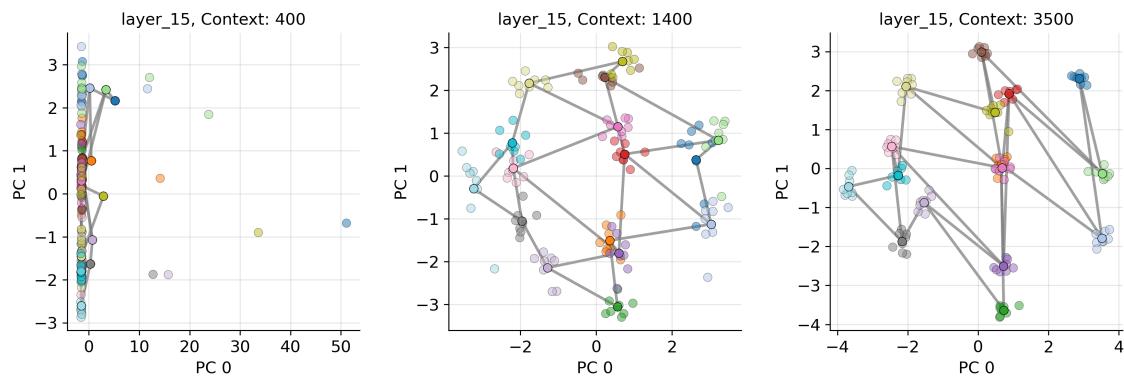


Figure 4.16 PCA visualization of 50% Mamba: 50% Attention

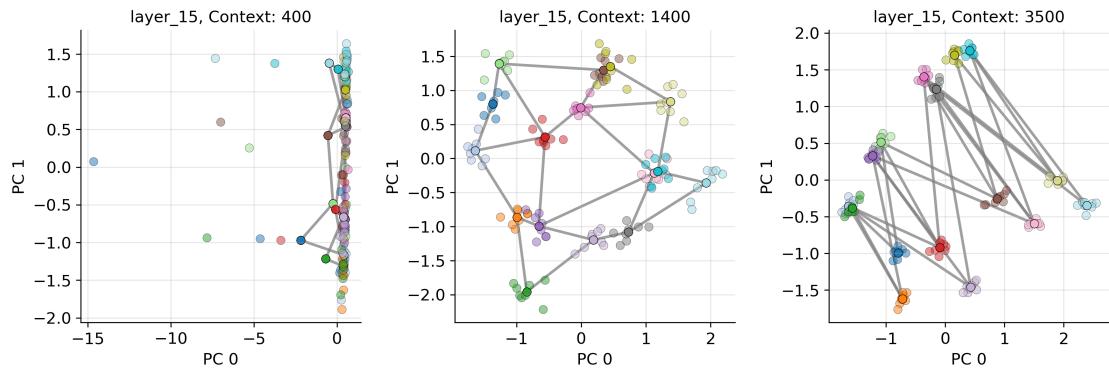


Figure 4.17 PCA visualization of 75% Mamba: 25% Attention

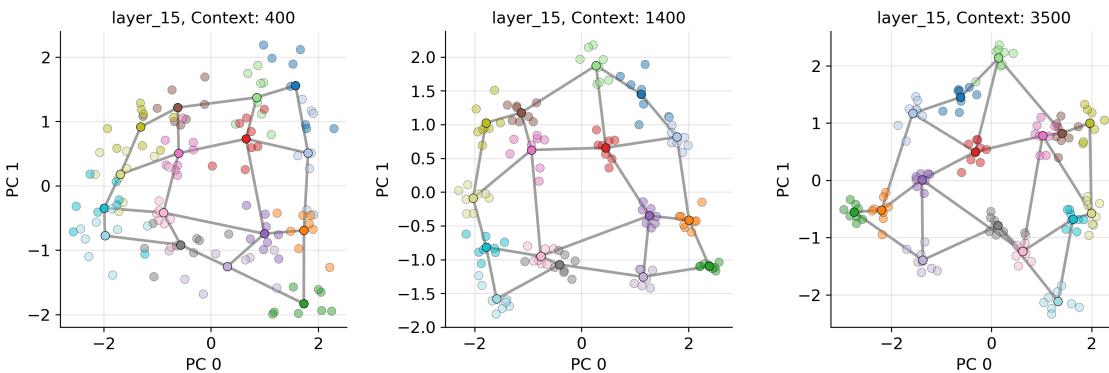


Figure 4.18 PCA visualization of 87.5% Mamba: 12.5% Attention

of recurrent updates. The ultimate challenge, which remains an open and fascinating area for research, is to discover the optimal way to blend these two powerful associative memory paradigms.

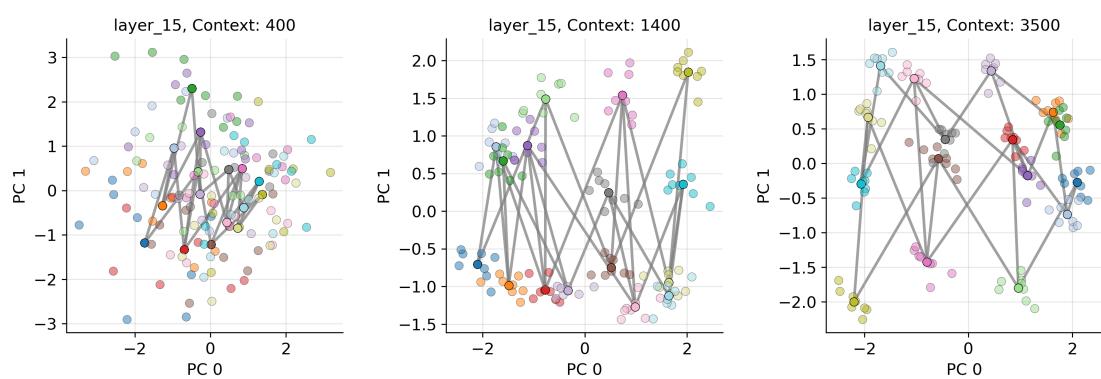


Figure 4.19 PCA visualization of 100% Mamba: 0% Attention

Chapter 5

Discussion

5.1 Do we really need long context?

The model's capacity for complex reasoning is fundamentally linked to the amount of information it can process in a single context window. We have seen models with larger context windows demonstrate remarkable "needle-in-a-haystack" retrieval capabilities and generate more coherent, long-form text, leading to an architectural arms race towards processing millions of tokens. The implicit assumption is that the primary barrier to more general intelligence is the size of the model's working memory, raising a provocative and fundamental question: if we could build a Transformer with a truly infinite context window, would it become infinitely intelligent? As explored by [Zhong et al. \(2025\)](#), associative memory models, when viewed as the online optimization systems we have described, face a theoretical risk of memory convergence during in-context learning. Their analysis show that for both DeltaNet and softmax attention, the memory state M_t will in the limit of infinitely long context, converge to a fixed point. This theoretical risk is compounded by a well-documented empirical phenomenon: the "Lost in the Middle" problem from [Liu et al. \(2023\)](#). Multiple studies have robustly demonstrated that even state-of-the-art long-context Transformers do not utilize their context window uniformly. Performance on retrieval and question-answering tasks is highest when the relevant information is placed at the very beginning or the very end of the context, and it degrades significantly when the crucial facts are buried in the middle. This U-shaped performance curve reveals that having information within the context window is not the same as being able to access and reason over it effectively. A model with a million-token context may not have a million tokens of usable context. This suggests that the brute-force memory of the KV Cache is not a perfect associative memory but a

noisy and biased one, where positional biases and attentional saturation limit its practical utility long before its theoretical capacity is reached.

This leads to a more nuanced vision for future architectures, one that moves beyond the monolithic pursuit of a single, ever-larger memory system. Intelligence is likely not just a function of memory size, but of the interplay between different types of memory and computation. The efficient recurrent models at the heart of this thesis, which are forced by their $\mathcal{O}(1)$ state size to learn compressed representations, are a natural candidate for an intelligent "working memory". The true challenge, therefore, may not be building a memory of infinite size, but designing the optimal interface between a fast, intelligent reasoning engine and a vast but passive knowledge repository.

5.2 Recurrent Depth and Latent Reasoning

If scaling horizontally along the axis of context length is not the only path forward, we must consider the alternative: scaling vertically by increasing the computational depth applied at each timestep. Instead of a model that reads more, we can envision a model that thinks more about what it has already read. The standard recurrent update $\mathbf{M}_t = f(\mathbf{M}_{t-1}, \mathbf{x}_t)$, performs a single transformation per input token. The idea of recurrent depth is to iterate this update multiple times per timestep before producing an output. The model would perform a sequence of internal state transitions:

$$\mathbf{M}_t^{k+1} = f(\mathbf{M}_t^k, \mathbf{x}_t) \quad \text{for } k = 0, 1, \dots, K - 1 \quad (5.1)$$

and $\mathbf{M}_t^0 = \mathbf{M}_{t-1}$. This iterative process draws a direct and powerful analogy to the dynamics of a Hopfield network. A standard RNN is like a single, one-step update of a Hopfield network's state. Introducing recurrent depth is equivalent to allowing the network to iterate for multiple steps, letting its internal state settle into a more stable energy minimum.

We can frame this iterative process as latent reasoning. The sequence of intermediate states, $\mathbf{M}_t^1, \dots, \mathbf{M}_t^K$ represents an internal chain-of-thought that occurs within the model's associative memory before it commits to emitting the next token. The crucial advantage of the efficient recurrent models studied in this thesis is that iterating their fixed-size state update is computationally cheap. In contrast, iterating a full Transformer block, with its $\mathcal{O}(T^2)$ attention mechanism, would be prohibitively expensive. Linear-time recurrent architectures are therefore the ideal substrate for exploring this new dimension of computational depth.

5.3 Test time training

The concept of latent reasoning, which can be viewed as an implicit, fixed-point optimization, can be generalized to its most powerful and explicit form: Test-Time Training (TTT). This paradigm takes the central thesis of our online optimization framework. If the forward pass of a recurrent model is an approximate solution to a per-step optimization problem, TTT proposes that we can achieve better performance by investing more compute to find a better solution at inference time. TTT involves performing several explicit steps of gradient descent on a subset of the model’s parameters (or a dedicated set of adapter parameters) to minimize a self-supervised loss on the current test input. For instance, given a complex prompt, the model could be tasked with minimizing its own prediction error on different parts of that prompt, thereby adapting its internal memory state to become a better one on that specific context before generating a final response.

This is the ultimate practical realization of the online optimization framework. The forward pass is no longer a single, fixed update rule but a dynamic, adaptive optimization process. This suggests a path towards models which can learn and adapt in-context at inference time. While challenges remain in defining the right self-supervised objectives and ensuring computational tractability, TTT represents a paradigm shift from static, pretrained models to dynamic, continuously adapting agents.

5.4 Outlook on architectural design

The design choices made in a sequence model are instantiations of specific learning and memory management style. This perspective compels us to move beyond the current paradigm, which has been dominated by the monolithic scaling of a single architectural primitive, the Transformer. The fundamental question for the next generation of AI should not be "What is the successor to the Transformer?" but rather, "How do we design systems that intelligently compose different forms of memory and computation?"

The future of advanced AI may not lie in a single, unified architecture, but in a more modular and cognitively inspired approach that embraces architectural specialization. Human cognition does not rely on one giant, uniform memory system; it leverages a sophisticated interplay between a fast, flexible, but capacity-limited "working memory" for active reasoning, and a vast, slower, and more compressed

"long-term memory" for storing knowledge. The architectures explored in this thesis point towards a digital analogue of this system. The efficient, fixed-state recurrent models, with their $\mathcal{O}(1)$ update cost are the ideal candidates for the role of computational working memory.

This suggests that hybrid models are not a compromise, but can be the central design pattern. One can envision an architecture where a powerful recurrent core acts as the central processing unit, performing the heavy lifting of reasoning and problem-solving by iterating its internal state.

Chapter 6

Conclusion

This thesis embarked on a journey to rethink sequence models, motivated by two converging trends: the rise of test-time compute as a new scaling frontier and the computational bottlenecks of the Transformer architecture. We began by stepping back from modern engineering heuristics and grounding our investigations in the rich literature surrounding associative memory. By tracing the evolution of sequence models through this theoretical lens, we presented a unifying framework wherein architectures are understood as different algorithmic solutions to a continuous online optimization problem. This perspective allowed us to move beyond simply describing architecture and towards a more fundamental understanding of the principles governing their behavior.

Our work has sought to bridge the gap between this theoretical understanding and practical applications. We have demonstrated that the future of scalable reasoning may not lie in a monolithic architectural choice but in a union of paradigms. The hybrid models at the core of this thesis, which interleave the non-parametric, global memory of attention with the compressed, fixed-state memory of recurrent layers, represent a concrete step in this direction. Through our empirical investigations, we have not only validated their efficacy but have also begun to chart the complex trade-offs between architectural design, reasoning capability, and representational learning. This concluding chapter will reason about the primary contributions and findings of this thesis, reflect upon their broader implications, acknowledge the limitations of our work, and offer an outlook on the future of intelligent architectural design.

Contributions

First, on a theoretical level, we have presented a comprehensive associative memory viewpoint that unifies a wide range of modern sequence architectures. By framing the forward pass as an online optimization process, we have shown that the update rules of models from Linear Attention to Gated DeltaNet can be rigorously derived from first principles. The architectural diversity we observe is a direct consequence of different choices for the core components of this optimization: the attentional bias, which defines the learning objective, and the retention regularizer, which governs the forgetting mechanism. This framework provides a common language and a structured design space that demystifies the connections between seemingly disparate models and provides a principled basis for future innovation.

Second, on a methodological level, we have developed and validated a practical pathway for creating large-scale, inference-efficient hybrid models. Recognizing the prohibitive cost of pretraining novel architectures from scratch, we adapted a multi-stage, cross-architecture distillation framework. This procedure leverages the immense investment in open-source Transformers by seeding a recurrent hybrid "student" with the parametric knowledge of a pretrained "teacher," followed by knowledge distillation and supervised fine-tuning. This methodology serves as a valuable tool for the research community, lowering the barrier to entry for exploring the vast design space of hybrid models and enabling the creation of powerful, specialized reasoners with a fraction of the computational resources.

Third, through our empirical investigations, we have provided the first systematic exploration of these hybrid architectures in the context of mathematical reasoning and test-time scaling. Our pretraining experiments at the 150M scale revealed a non-linear relationship between the proportion of attention layers and performance, suggesting that a small amount of attention provides a disproportionate reasoning benefit. Furthermore, we put forward a glimpse into the representational differences between architectures.

Main Findings

Perhaps the main finding is that peak performance and architectural purity are not synonymous. Our experiments consistently show that hybrid models composed of only 15-25% attention layers can achieve a substantial fraction of the reasoning performance of a pure Transformer. It suggests that the relationship between different layers is not homogeneous and some layers may be more critical for global information routing, while others can effectively operate with a more compressed, local view

of the sequence. This has profound implications for architectural design, suggesting a future where models are not uniform stacks of identical blocks but are instead hybrid systems with layers specialized for different computational functions.

Furthermore, our work highlights both the promise and the issues of cross-architecture distillation. While our three-stage procedure successfully produced performant hybrid models, fully recovering the teacher’s capabilities proved challenging. This finding implies that while distillation is a powerful tool for rapid prototyping and research, achieving state-of-the-art performance with novel architectures will likely still require the immense investment of direct pretraining. Disentangling the inherent limitations of a student architecture from the procedural artifacts of the distillation process itself remains a critical and open area of research.

Limitations and Avenues for Further Research

This thesis, while comprehensive in its scope, is subject to several limitations that naturally give rise to avenues for future work. First, our pretraining experiments were conducted at the 150M parameter scale. While this provides a controlled environment for architectural comparison, it remains an open question whether the specific trade-offs and scaling trends we observed hold true for models at larger sizes. Verifying these findings at a larger scale is a crucial next step. Second, our investigation focused on a specific class of mathematical reasoning tasks. The optimal balance between attention and recurrence is likely to be task-dependent. Future work should explore the performance of these hybrid architectures on a wider range of tasks, to develop a more holistic understanding of their capabilities. As outlined in our discussion, exploring recurrent depth and latent reasoning might prove to be a powerful and useful way of scaling reasoning capabilities. Our results show that reasoning capabilities is not merely a function of memory size, but might be an interplay between different types of memory and computation.

Appendix A

Appendix

A.1 Chunkwise training of algorithms

Here we present a short explanation of the chunkwise training algorithm behind the efficiency of linear attention variants. storing the full hidden state matrices, typically require expensive memory transfer from the GPU memory. Matrix multiplications on modern GPUs, on the other hand, are highly optimized using tensor cores and we can leverage this to only store the hidden states once every chunk of size C .

We will particularly look at the chunkwise parallel form of linear attention. This can be extended to other variants and we encourage the reader to check the beautiful FlashLinearAttention Library ([Yang and Zhang, 2024](#)). The basic idea behind this form is that the outer product can be written as matrix multiplications,

$$\sum_{i=1}^T \mathbf{v}_i \mathbf{k}_i^T = \mathbf{V}_T \mathbf{K}_T^T \quad (\text{A.1})$$

$$\text{where } \mathbf{V}_T = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_t] \quad (\text{A.2})$$

$$\mathbf{K}_T = [\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_t] \quad (\text{A.3})$$

The goal is to materialize states at every C chunksize $\mathbf{M}_0, \mathbf{M}_C, \mathbf{M}_{2C}, \dots, \mathbf{M}_{(n-1)C}$ where $n = \lceil L/C \rceil$.

Denote by $\mathbf{M}_{[i]} = \mathbf{M}_{iC} \in \mathbb{R}^{d \times d}$,

$$\mathbf{M}_{[i]}^r = \mathbf{M}_{iC+r} \in \mathbb{R}^{d \times d},$$

$$\mathbf{Q}_{[i]} = \mathbf{Q}_{iC+1:(i+1)C} \in \mathbb{R}^{C \times d},$$

$$\mathbf{K}_{[i]} = \mathbf{K}_{iC+1:(i+1)C} \in \mathbb{R}^{C \times d},$$

$$\mathbf{V}_{[i]} = \mathbf{V}_{iC+1:(i+1)C} \in \mathbb{R}^{C \times d},$$

$$\mathbf{O}_{[i]} = \mathbf{O}_{iC+1:(i+1)C} \in \mathbb{R}^{C \times d},$$

$$\mathbf{q}_{[i]}^r = \mathbf{q}_{iC+r} \in \mathbb{R}^{1 \times d},$$

$$\mathbf{k}_{[i]}^r = \mathbf{k}_{iC+r} \in \mathbb{R}^{1 \times d},$$

$$\mathbf{v}_{[i]}^r = \mathbf{v}_{iC+r} \in \mathbb{R}^{1 \times d},$$

$$\mathbf{o}_{[i]}^r = \mathbf{o}_{iC+r} \in \mathbb{R}^{1 \times d},$$

For any position r within chunk i, we can compute

$$\mathbf{M}_{[i]}^r = \mathbf{M}_{[i]} + \sum_{t=1}^r \mathbf{v}_{[i]}^t \mathbf{k}_{[i]}^{t\top} \quad (\text{A.4})$$

$$\mathbf{o}_{[i]}^r = \mathbf{M}_{[i]}^r \mathbf{q}_{[i]}^r = \mathbf{M}_{[i]} \mathbf{q}_{[i]}^r + \sum_{t=1}^r \mathbf{v}_{[i]}^t (\mathbf{k}_{[i]}^{t\top} \mathbf{q}_{[i]}^r) \quad (\text{A.5})$$

and in matrix form,

$$\mathbf{M}_{[t+1]} = \mathbf{M}_{[t]} + \mathbf{V}_{[t]}^\top \mathbf{K}_{[t]} \in \mathbb{R}^{d \times d} \quad (\text{A.6})$$

$$\mathbf{O}_{[t]} = \mathbf{Q}_{[t]} \mathbf{S}_{[t]}^\top + (\mathbf{Q}_{[t]} \mathbf{K}_{[t]}^\top \odot \Gamma) \mathbf{V}_{[t]} \in \mathbb{R}^{C \times d} \quad (\text{A.7})$$

This chunkwise formulation enables efficient hardware utilization and enables scaling, an important requisite for intelligence.

```

1 import numpy as np
2
3 def linear_attention_naive(Q, K, V):
4     """
5         Q, K, V : (T, d)  Query, Key, Value tensors
6         returns O : (T, d)
7     """
8     T, d = Q.shape
9     S = np.zeros((d, d))      # cumulative prefix state
10    O = np.empty_like(Q)
11
12    for t in range(T):        # left-to-right along the sequence
13        q, k, v = Q[t], K[t], V[t]
14
15        O[t] = q @ S.T        # prefix contribution
16        O[t] += (q @ k) * v    # current token
17        S += np.outer(v, k)    # update state
18
19    return O

```

Listing A.1 Naive causal linear attention (single head)

```

1 import numpy as np
2

```

```

3 def linear_attention_chunkwise(Q, K, V, C):
4     """
5     Q, K, V : (T, d)
6     C         : chunk length
7     returns O : (T, d)
8     """
9     T, d     = Q.shape
10    S        = np.zeros((d, d))           # carries prefix state
11    O        = np.empty_like(Q)
12    n_chunk = (T + C - 1) // C          # ceil division
13
14    for c in range(n_chunk):
15        s = c * C
16        e = min((c + 1) * C, L)          # slice [s:e]
17        Qc, Kc, Vc = Q[s:e], K[s:e], V[s:e]
18        m = e - s
19
20        mask = np.tril(np.ones((m, m)))   # causal mask
21
22        Oc = Qc @ S.T
23        Oc += ((Qc @ Kc.T) * mask) @ Vc  # in-chunk causal part
24        O[s:e] = Oc
25
26        S += Vc.T @ Kc                  # update prefix state
27
28    return O

```

Listing A.2 Chunkwise causal linear attention (chunk size C)

A.2 Dataset Examples

As mentioned in Section 3.2.1 we provide more details for the dataset here.

A.2.1 OpenMathInstruct2

OpenMathInstruct2

Question:

Ben throws four identical darts. Each hits one of four identical dartboards on the wall. After throwing the four darts, he lists the number of darts that hit each board, from greatest to least. How many different lists are possible?

Answer:

Step 1: Understand the constraints of the problem

There are four identical darts and four identical dartboards. The task is to list the number of darts that hit each board from greatest to least.

Step 2: Enumerate the possibilities. Given the constraints, we can start by listing the possibilities systematically, starting with the highest number of darts on one board and then distributing the remaining darts across the other boards in decreasing order.

Step 3: List all possible distributions

1. All four darts hit one board: 4, 0, 0, 0
2. Three darts hit one board, and one dart hits another: 3, 1, 0, 0
3. Two darts hit one board, and two darts hit another: 2, 2, 0, 0
4. Two darts hit one board, one dart hits another, and one dart hits yet another: 2, 1, 1, 0
5. One dart hits each board: 1, 1, 1, 1

Step 4: Count the different possibilities

There are 5 different ways to distribute the darts as listed above.

The final answer is:

A.2.2 MetaMathQA

MetaMathQA

Question:

Quantities r and s vary inversely. When r is 1200, s is 0.35. What is the value of s when r is 2400? Express your answer as a decimal to the nearest thousandths.

Answer:

If r and s vary inversely, then we know that $r \cdot s = k$ for some constant k .

We are given that when r is 1200, s is 0.35. So we can set up the equation:

$$1200 \cdot 0.35 = k$$

Simplifying, we find that $k = 420$.

Now we can use this value of k to solve for s when r is 2400:

$$2400 \cdot s = 420$$

Dividing both sides by 2400, we find that $s = 0.175$ to the nearest thousandths.

The answer is: 0.175

A.2.3 GSM8K

GSM8K

Question:

Julie is reading a 120-page book. Yesterday, she was able to read 12 pages and today, she read twice as many pages as yesterday. If she wants to read half of the remaining pages tomorrow, how many pages should she read?

Answer:

Maila read $12 \times 2 = 24$ pages today.

So she was able to read a total of $12 + 24 = 36$ pages since yesterday.

There are $120 - 36 = 84$ pages left to be read.

Since she wants to read half of the remaining pages tomorrow, then she should read $84/2 = 42$ pages.

42

A.2.4 MATH500

MATH500

Question:

Let x, y and z be positive real numbers that satisfy the following system of equations:

$$\log_2 \left(\frac{x}{yz} \right) = \frac{1}{2}$$

$$\log_2 \left(\frac{y}{xz} \right) = \frac{1}{3}$$

$$\log_2 \left(\frac{z}{xy} \right) = \frac{1}{4}$$

Then the value of $|\log_2(x^4y^3z^2)|$ is $\frac{m}{n}$ where m and n are relatively prime positive integers. Find $m + n$.

Answer:

Denote $\log_2(x) = a$, $\log_2(y) = b$, and $\log_2(z) = c$.

Then, we have:

$$a - b - c = \frac{1}{2}$$

$$-a + b - c = \frac{1}{3}$$

$$-a - b + c = \frac{1}{4}$$

Now, we can solve to get $a = \frac{-7}{24}, b = \frac{-9}{24}, c = \frac{-5}{12}$. Plugging these values in, we obtain $|4a + 3b + 2c| = \frac{25}{8} \implies \boxed{033}$. akliu

$$\log_2(y/xz) + \log_2(z/xy) = \log_2(1/x^2) = -2\log_2(x) = \frac{7}{12}$$

$$\log_2(x/yz) + \log_2(z/xy) = \log_2(1/y^2) = -2\log_2(y) = \frac{3}{4}$$

$$\log_2(x/yz) + \log_2(y/xz) = \log_2(1/z^2) = -2\log_2(z) = \frac{5}{6}$$

$$\log_2(x) = -\frac{7}{24}$$

$$\log_2(y) = -\frac{3}{8}$$

$$\log_2(z) = -\frac{5}{12}$$

$$4\log_2(x) + 3\log_2(y) + 2\log_2(z) = -25/8$$

$$25 + 8 = \boxed{033}$$
 Callisto531

Adding all three equations, $\log_2(\frac{1}{xyz}) = \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{13}{12}$. Subtracting this from every equation, we have:

$$2\log_2 x = -\frac{7}{12},$$

$$2\log_2 y = -\frac{3}{4},$$

$$2\log_2 z = -\frac{5}{6}$$

Our desired quantity is the absolute value of $4\log_2 x + 3\log_2 y + 2\log_2 z = 2(\frac{7}{12}) + 3/2(\frac{3}{4}) + \frac{5}{6} = \frac{25}{8}$, so our answer is $25 + 8 = \boxed{033}$. Spoirvfimidf

A.2.5 OpenR1-Math-220k

OpenR1-Math-220k

Question:

Let h_n and k_n be the unique relatively prime positive integers such that

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \frac{h_n}{k_n}.$$

Let L_n denote the least common multiple of the numbers $1, 2, 3, \dots, n$. For how many integers with $1 \leq n \leq 22$ is $k_n < L_n$?

- (A) 0 (B) 3 (C) 7 (D) 8 (E) 10

Answer:

<think>

Okay, so I need to figure out for how many integers n between 1 and 22 inclusive, the denominator k_n of the harmonic number $H_n = 1 + 1/2 + 1/3 + \dots + 1/n$ (when written in lowest terms) is less than the least common multiple L_n of the numbers 1 through n . The answer choices are given, and the correct answer is among A to E, with E being 10. First, let me recall that the harmonic number H_n is the sum of reciprocals from 1 to n . When you sum these fractions, the denominator in reduced form (k_n) is not necessarily equal to the LCM of the denominators, which is L_n . The question is asking for how many n from 1 to 22 does $k_n < L_n$ hold true. That is, in how many cases does the denominator of the reduced fraction H_n not equal the LCM of 1 through n .

I remember that when you add fractions, the denominator of the sum in reduced form is the LCM of the individual denominators divided by some common factors that might exist in the numerators. But in this case, since we're adding $1 + 1/2 + \dots + 1/n$, each term is a reciprocal of an integer. When adding these up, the denominators can sometimes share factors, so when you sum them, the reduced denominator might have factors canceled out. . . .

</think>

...

****Manual Calculation and Known Results****

For $n = 6, 7, 8$, the denominators k_n are 20, 140, and 280 respectively, which are less than L_n . For higher n , primes such as 3, 5, 7, etc., can divide the numerator of H_n , leading to $k_n < L_n$.

****Conclusion****

The values of n where $k_n < L_n$ are found to be 6, 7, 8, 18, 20, 21, and 22. However, considering known results and systematic checks, the total count is 8.

Thus, the number of integers n with $1 \leq n \leq 22$ such that $k_n < L_n$ is D.

A.2.6 OpenThoughts-114k-math

MATH500

Question:

Return your final response within `\boxed{}`. Let a, b, c be positive real numbers. Prove that

$$\frac{1}{a(1+b)} + \frac{1}{b(1+c)} + \frac{1}{c(1+a)} \geq \frac{3}{1+abc},$$

and that equality occurs if and only if $a = b = c = 1$

Answer:

<|begin_of_thought|> Okay, so I need to prove this inequality: $\frac{1}{a(1+b)} + \frac{1}{b(1+c)} + \frac{1}{c(1+a)} \geq \frac{3}{1+abc}$, where a, b, c are positive real numbers, and equality holds if and only if $a = b = c = 1$. Hmm, let me try to think through this step by step. First, I remember that inequalities often involve techniques like AM-GM, Cauchy-Schwarz, or other classical inequalities. Maybe I can start by looking at each term on the left side and see if I can manipulate them to relate to the right side. The left side has three terms: $\frac{1}{a(1+b)}$, $\frac{1}{b(1+c)}$, and $\frac{1}{c(1+a)}$. Each denominator has a variable multiplied by 1 plus another variable. The right side is $\frac{3}{1+abc}$, which is symmetric in a, b, c . Since the equality case is when all variables are 1, maybe substituting $a = b = c = 1$ gives both sides equal to $3/(1 + 1) = 1.5$, and the left side becomes $1/(1 * (1 + 1))$ three times, which is $0.5 * 3 = 1.5$.

...

<|end_of_thought|>

<|begin_of_solution|>

To prove the inequality $\frac{1}{a(1+b)} + \frac{1}{b(1+c)} + \frac{1}{c(1+a)} \geq \frac{3}{1+abc}$ for positive real numbers a, b, c with equality if and only if $a = b = c = 1$, we proceed as follows:

Key Steps and Proof.1. **Substitution for $abc = 1$:** - Assume $abc = 1$.

Let $a = \frac{x}{y}$, $b = \frac{y}{z}$, $c = \frac{z}{x}$. This transforms the left-hand side of the inequality into: $\frac{yz}{x(y+z)} + \frac{zx}{y(z+x)} + \frac{xy}{z(x+y)}$ - Using the AM-GM inequality on each term, we show: $\frac{yz}{x(y+z)} \geq \frac{\sqrt{yz}}{2x}$ - Summing up these inequalities and applying AM-GM again, we get: $\sum \frac{yz}{x(y+z)} \geq \frac{1}{2} \left(\frac{\sqrt{yz}}{x} + \frac{\sqrt{zx}}{y} + \frac{\sqrt{xy}}{z} \right) \geq \frac{3}{2}$

General Case via Scaling: - For any positive real numbers a, b, c , let $k = abc$. Scale the variables to $a' = ak^{-1/3}$, $b' = bk^{-1/3}$, $c' = ck^{-1/3}$ such that $a'b'c' = 1$. - Applying the proven inequality for $abc = 1$, we have: $\sum \frac{1}{a'(1+b')} \geq \frac{3}{2}$ - Reverting the scaling confirms the original inequality holds for general a, b, c .

Equality Condition: - Equality in the AM-GM steps occurs when all terms are equal, leading to $a = b = c = 1$.

Final Answer
$$\boxed{\frac{1}{a(1+b)} + \frac{1}{b(1+c)} + \frac{1}{c(1+a)} \geq \frac{3}{1+abc}}$$
 Equality holds if and only if $a = b = c = 1$. <|end_of_solution|>

A.3 Further Pretraining Plots

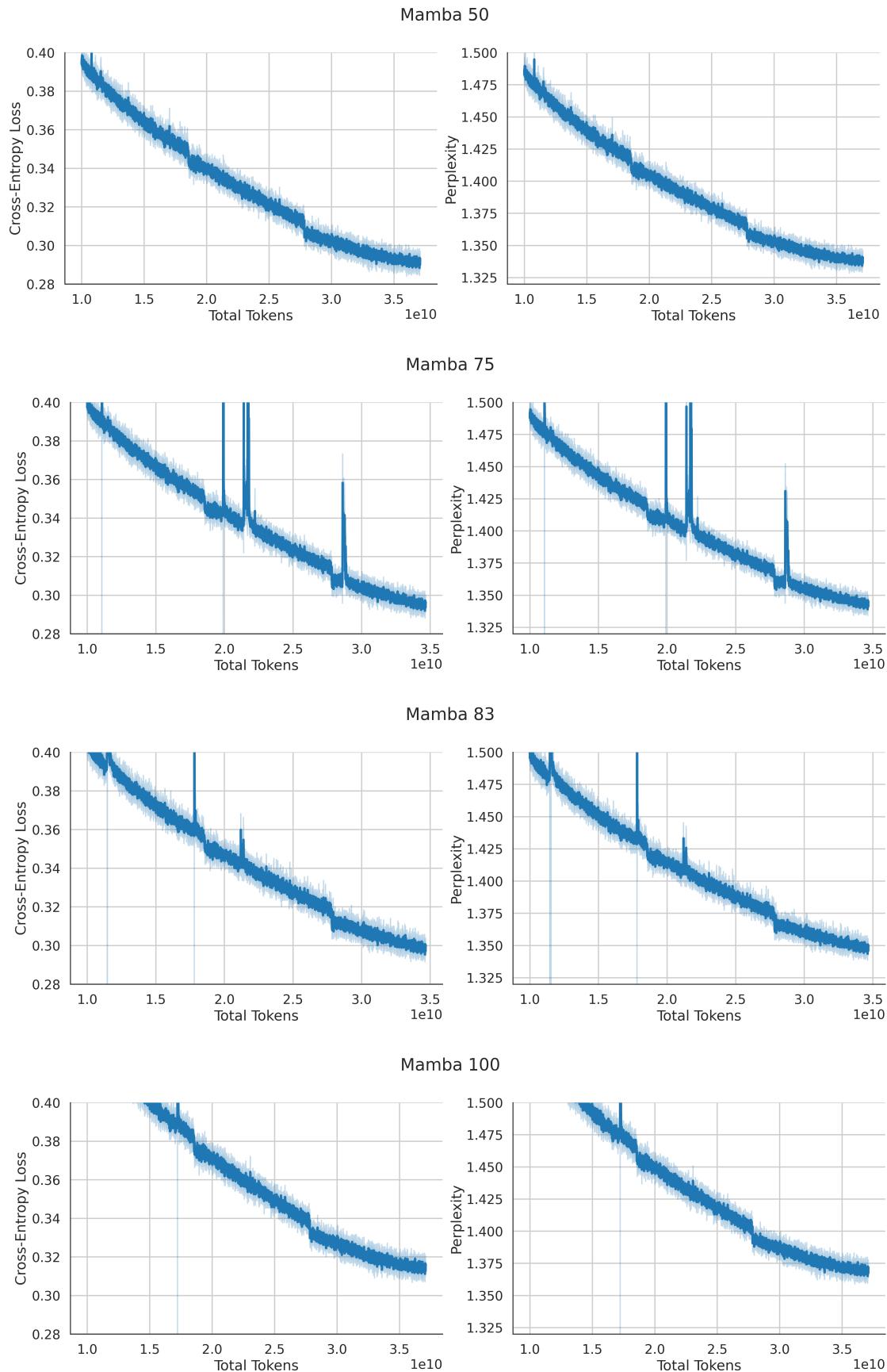


Figure A.1 Cross-entropy loss and perplexity for the Mamba Hybrid models. For each percentage of hybrids, we choose the best one from the hyperparameter sweeps.

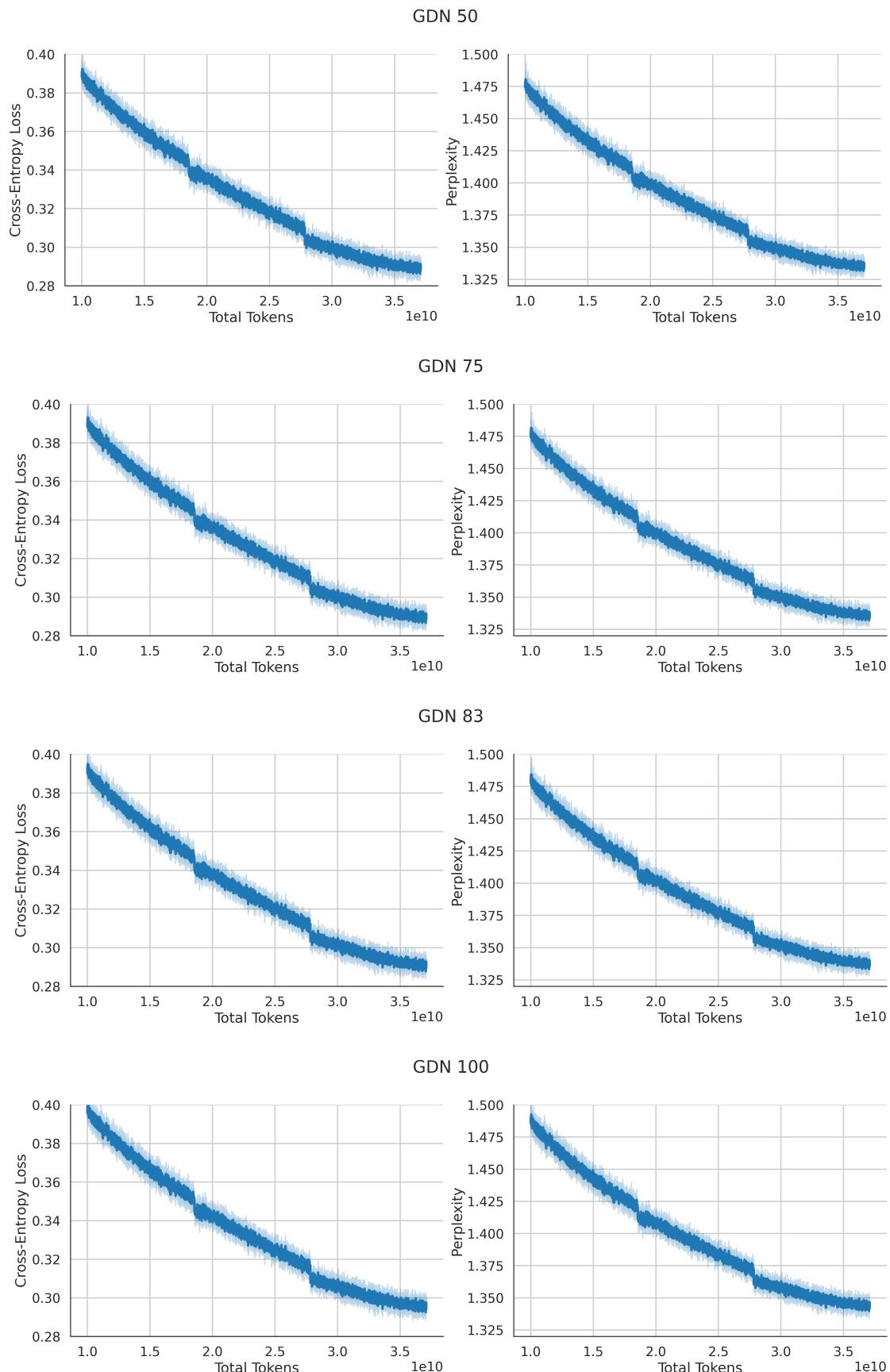


Figure A.2 Cross-entropy loss and perplexity for the GDN series.

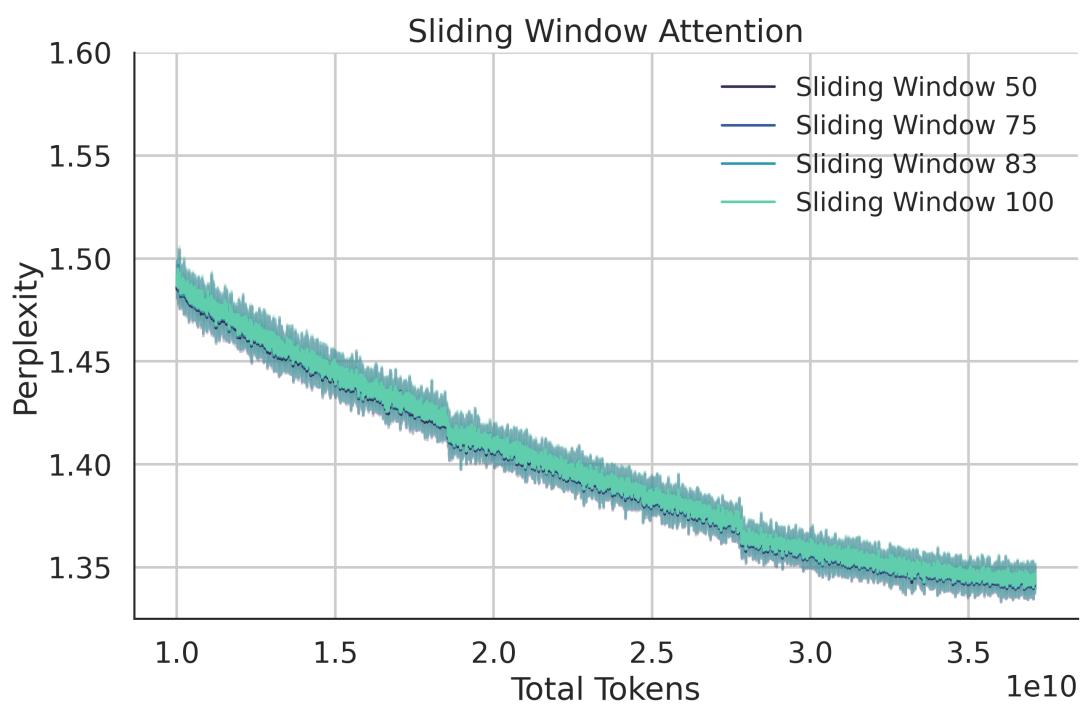


Figure A.3 Sliding Window attention hybrid models. We use a sliding window of 1024 for a training sequence length of 2048.

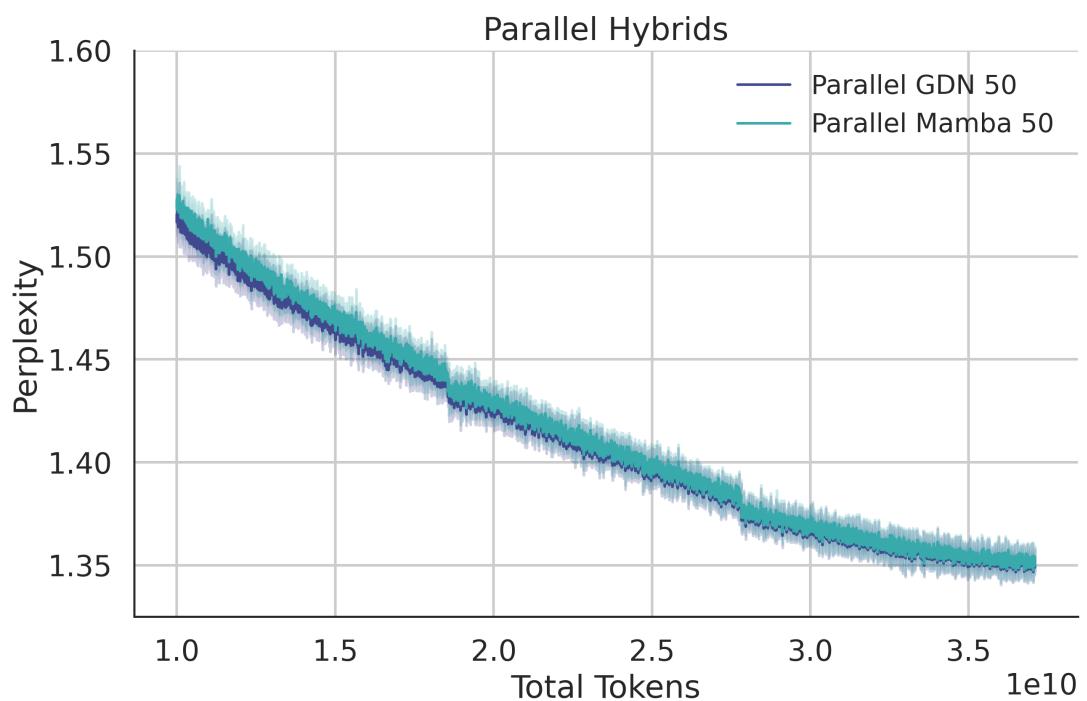


Figure A.4 Parallel Hybrids, i.e, For each layer, 50 percent of the heads are of linear attention variant. Introduced in Dong et al. (2024) and also used in Zuo et al. (2025) to train large scale models.

Bibliography

- Allen-Zhu, Z. and Alfarano, A. (2025). lt;divgt; physics of language models: Part 4.1, amp;nbsp;lt;spangt;architecture design and the magic of canon layerslt;/spangt; lt;/divgt;.
- Amit, D. J., Gutfreund, H., and Sompolinsky, H. (1985). Storing infinite numbers of patterns in a spin-glass model of neural networks. *Physical Review Letters*, 55(14):1530â1533.
- Anderson, J. A. (1972). A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14(3â4):197â220.
- Arora, S., Eyuboglu, S., Timalsina, A., Johnson, I., Poli, M., Zou, J., Rudra, A., and Ré, C. (2023). Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*.
- Arora, S., Eyuboglu, S., Zhang, M., Timalsina, A., Alberti, S., Zinsley, D., Zou, J., Rudra, A., and Ré, C. (2024). Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*.
- Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.
- Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate.
- Beck, M., Pöppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. (2024). xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*.
- Behrouz, A., Li, Z., Kacham, P., Daliri, M., Deng, Y., Zhong, P., Razaviyayn, M., and Mirrokni, V. (2025). Atlas: Learning to optimally memorize the context at test time.
- Behrouz, A., Zhong, P., and Mirrokni, V. (2024). Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*.
- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157â166.
- Bick, A., Li, K., Xing, E., Kolter, J. Z., and Gu, A. (2025). Transformers to ssms: Distilling quadratic knowledge to subquadratic models. *Advances in Neural Information Processing Systems*, 37:31788â31812.

- Bietti, A., Cabannes, V., Bouchacourt, D., Jegou, H., and Bottou, L. (2023). Birth of a transformer: A memory viewpoint.
- Brandon, W., Mishra, M., Nrusimha, A., Panda, R., and Kelly, J. R. (2024). Reducing transformer key-value cache size with cross-layer attention.
- Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. (2024). Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.
- Bühlmann, P. and van de Geer, S. (2011). *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer Berlin Heidelberg.
- Cabannes, V., Dohmatob, E., and Bietti, A. (2024a). Scaling laws for associative memories.
- Cabannes, V., Simsek, B., and Bietti, A. (2024b). Learning associative memories with gradient descent.
- Chaudhry, H. T., Kulkarni, M., and Pehlevan, C. (2025). Test-time scaling meets associative memory: Challenges in subquadratic models. In *New Frontiers in Associative Memories*.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation.
- Chollet, F. (2019). On the measure of intelligence. *arXiv preprint arXiv:1911.01547*.
- Choromanski, K., Likhoshesterov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., and Weller, A. (2022). Rethinking attention with performers.
- Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.
- Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021). Training verifiers to solve math word problems.
- Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning.

Dao, T. and Gu, A. (2024). Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

DeepSeek-AI, Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Xu, H., Yang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Chen, J., Yuan, J., Qiu, J., Song, J., Dong, K., Gao, K., Guan, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Pan, R., Xu, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Zheng, S., Wang, T., Pei, T., Yuan, T., Sun, T., Xiao, W. L., Zeng, W., An, W., Liu, W., Liang, W., Gao, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Chen, X., Nie, X., Sun, X., Wang, X., Liu, X., Xie, X., Yu, X., Song, X., Zhou, X., Yang, X., Lu, X., Su, X., Wu, Y., Li, Y. K., Wei, Y. X., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Zheng, Y., Zhang, Y., Xiong, Y., Zhao, Y., He, Y., Tang, Y., Piao, Y., Dong, Y., Tan, Y., Liu, Y., Wang, Y., Guo, Y., Zhu, Y., Wang, Y., Zou, Y., Zha, Y., Ma, Y., Yan, Y., You, Y., Liu, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Huang, Z., Zhang, Z., Xie, Z., Hao, Z., Shao, Z., Wen, Z., Xu, Z., Zhang, Z., Li, Z., Wang, Z., Gu, Z., Li, Z., and Xie, Z. (2024). Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model.

Dong, X., Fu, Y., Diao, S., Byeon, W., Chen, Z., Mahabaleshwarkar, A. S., Liu, S.-Y., Keirsbilck,

- M. V., Chen, M.-H., Suhara, Y., Lin, Y., Kautz, J., and Molchanov, P. (2024). Hymba: A hybrid-head architecture for small language models.
- Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179â211.
- Glorioso, P., Anthony, Q., Tokpanov, Y., Whittington, J., Pilault, J., Ibrahim, A., and Millidge, B. (2024). Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*.
- Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., GuzmÃ¡n, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhota, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Äelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenber, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A.,

Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Caggioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Khanelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., and Ma, Z. (2024). The llama 3 herd of models.

Gu, A. and Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Guha, E., Marten, R., Keh, S., Raoof, N., Smyrnis, G., Bansal, H., Nezhurina, M., Mercat, J., Vu, T., Sprague, Z., Suvarna, A., Feuer, B., Chen, L., Khan, Z., Frankel, E., Grover, S., Choi, C., Muennighoff, N., Su, S., Zhao, W., Yang, J., Pimpalgaonkar, S., Sharma, K., Ji, C. C.-J., Deng,

- Y., Pratt, S., Ramanujan, V., Saad-Falcon, J., Li, J., Dave, A., Albalak, A., Arora, K., Wulf, B., Hegde, C., Durrett, G., Oh, S., Bansal, M., Gabriel, S., Grover, A., Chang, K.-W., Shankar, V., Gokaslan, A., Merrill, M. A., Hashimoto, T., Choi, Y., Jitsev, J., Heckel, R., Sathiamoorthy, M., Dimakis, A. G., and Schmidt, L. (2025). Openthoughts: Data recipes for reasoning models.
- Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., and Tian, Y. (2024). Training large language models to reason in a continuous latent space.
- Hebb, D. (1950). Hebb, d. o. the organization of behavior: A neuropsychological theory. new york: John wiley and sons, inc., 1949. 335 p. \$4.00. *Science Education*, 34(5):336–337.
- Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. (2021). Measuring mathematical problem solving with the math dataset.
- Hendrycks, D. and Gimpel, K. (2023). Gaussian error linear units (gelus).
- Henighan, T., Kaplan, J., Katz, M., Chen, M., Hesse, C., Jackson, J., Jun, H., Brown, T. B., Dhariwal, P., Gray, S., et al. (2020). Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*.
- Henry, A., Dachapally, P. R., Pawar, S., and Chen, Y. (2020). Query-key normalization for transformers.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. (2022). Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.
- Hu, J. Y.-C., Chen, B.-Y., Wu, D., Ruan, F., and Liu, H. (2024). Nonparametric modern hopfield models.
- Irie, K., Schlag, I., Csordás, R., and Schmidhuber, J. (2021). Going beyond linear transformers with recurrent fast weight programmers.
- James, G., Witten, D., Hastie, T., Tibshirani, R., and Taylor, J. (2023). *An Introduction to Statistical Learning: with Applications in Python*. Springer Texts in Statistics. Springer International Publishing.
- Jelassi, S., Brandfonbrener, D., Kakade, S. M., and Malach, E. (2024). Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*.
- Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., ÅlÅddek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S.,

- Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.
- [Kamradt, G. (2023). Needle in a haystack - pressure testing llms.
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.
- Kasai, J., Peng, H., Zhang, Y., Yogatama, D., Ilharco, G., Pappas, N., Mao, Y., Chen, W., and Smith, N. A. (2021). Finetuning pretrained transformers into rnns.
- Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention.
- Kohonen, T. (1972). Correlation matrix memories. *IEEE Transactions on Computers*, C-21(4):353–359.
- Krotov, D. and Hopfield, J. J. (2016). Dense associative memory for pattern recognition. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.
- Kumar, A., Zhuang, V., Agarwal, R., Su, Y., Co-Reyes, J. D., Singh, A., Baumli, K., Iqbal, S., Bishop, C., Roelofs, R., Zhang, L. M., McKinney, K., Shrivastava, D., Paduraru, C., Tucker, G., Precup, D., Behbahani, F., and Faust, A. (2024). Training language models to self-correct via reinforcement learning.
- Lieber, O., Lenz, B., Bata, H., Cohen, G., Osin, J., Dalmedigos, I., Safahi, E., Meirom, S., Belinkov, Y., Shalev-Shwartz, S., et al. (2024). Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*.
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2023). Lost in the middle: How language models use long contexts.
- Liu, Y., Tian, Y., Zhao, Y., Yu, H., Xie, L., Wang, Y., Ye, Q., Jiao, J., and Liu, Y. (2025). Vmamba: Visual state space model. *Advances in neural information processing systems*, 37:103031–103063.
- Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization.
- Lu, E., Jiang, Z., Liu, J., Du, Y., Jiang, T., Hong, C., Liu, S., He, W., Yuan, E., Wang, Y., Huang, Z., Yuan, H., Xu, S., Xu, X., Lai, G., Chen, Y., Zheng, H., Yan, J., Su, J., Wu, Y., Zhang, N. Y., Yang, Z., Zhou, X., Zhang, M., and Qiu, J. (2025). Moba: Mixture of block attention for long-context llms.
- Ma, J., Li, F., and Wang, B. (2024). U-mamba: Enhancing long-range dependency for biomedical image segmentation. *arXiv preprint arXiv:2401.04722*.

- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115â133.
- Mercat, J., Vasiljevic, I., Keh, S., Arora, K., Dave, A., Gaidon, A., and Kollar, T. (2024). Linearizing large language models.
- Mezard, M., Parisi, G., Virasoro, M., and Hopfield, J. (1987). *Spin Glass Theory And Beyond: An Introduction To The Replica Method And Its Applications*. World Scientific Lecture Notes In Physics. World Scientific Publishing Company.
- Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., CandÃ“s, E., and Hashimoto, T. (2025). s1: Simple test-time scaling.
- Nguyen, E., Poli, M., Durrant, M. G., Kang, B., Katrekar, D., Li, D. B., Bartie, L. J., Thomas, A. W., King, S. H., Brixi, G., et al. (2024). Sequence modeling and design from molecular to genome scale with evo. *Science*, 386(6723):ead09336.
- OLMo, T., Walsh, P., Soldaini, L., Groeneveld, D., Lo, K., Arora, S., Bhagia, A., Gu, Y., Huang, S., Jordan, M., Lambert, N., Schwenk, D., Tafjord, O., Anderson, T., Atkinson, D., Brahman, F., Clark, C., Dasigi, P., Dziri, N., Guerquin, M., Ivison, H., Koh, P. W., Liu, J., Malik, S., Merrill, W., Miranda, L. J. V., Morrison, J., Murray, T., Nam, C., Pyatkin, V., Rangapur, A., Schmitz, M., Skjonsberg, S., Wadden, D., Wilhelm, C., Wilson, M., Zettlemoyer, L., Farhadi, A., Smith, N. A., and Hajishirzi, H. (2025). 2 olmo 2 furious.
- Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. (2022). In-context learning and induction heads.
- OpenAI, :, Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., Iftimie, A., Karpenko, A., Passos, A. T., Neitz, A., Prokofiev, A., Wei, A., Tam, A., Bennett, A., Kumar, A., Saraiva, A., Vallone, A., Duberstein, A., Kondrich, A., Mishchenko, A., Applebaum, A., Jiang, A., Nair, A., Zoph, B., Ghorbani, B., Rossen, B., Sokolowsky, B., Barak, B., McGrew, B., Minaiev, B., Hao, B., Baker, B., Houghton, B., McKinzie, B., Eastman, B., Lugaresi, C., Bassin, C., Hudson, C., Li, C. M., de Bourcy, C., Voss, C., Shen, C., Zhang, C., Koch, C., Orsinger, C., Hesse, C., Fischer, C., Chan, C., Roberts, D., Kappler, D., Levy, D., Selsam, D., Dohan, D., Farhi, D., Mely, D., Robinson, D., Tsipras, D., Li, D., Oprica, D., Freeman, E., Zhang, E., Wong, E., Proehl, E., Cheung, E., Mitchell, E., Wallace, E., Ritter, E., Mays, E., Wang, F., Such, F. P., Raso, F., Leoni, F., Tsimpourlas, F., Song, F., von Lohmann, F., Sulit, F., Salmon, G., Parascandolo, G., Chabot, G., Zhao, G., Brockman, G., Leclerc, G., Salman, H., Bao, H., Sheng, H., Andrin, H., Bagherinezhad, H., Ren, H., Lightman, H., Chung, H. W., Kivlichan, I., O'Connell, I., Osband, I., Gilaberte, I. C., Akkaya, I., Kostrikov, I., Sutskever, I., Kofman, I., Pachocki, J., Lennon, J., Wei, J., Harb, J., Twore, J., Feng, J., Yu, J., Weng, J., Tang, J., Yu, J., Candela, J. Q., Palermo, J., Parish, J., Heidecke, J., Hallman, J., Rizzo, J., Gordon, J., Uesato, J., Ward, J., Huizinga, J., Wang, J., Chen, K., Xiao, K., Singhal, K., Nguyen, K., Cobbe, K., Shi, K., Wood, K., Rimbach, K.,

Gu-Lemberg, K., Liu, K., Lu, K., Stone, K., Yu, K., Ahmad, L., Yang, L., Liu, L., Maksin, L., Ho, L., Fedus, L., Weng, L., Li, L., McCallum, L., Held, L., Kuhn, L., Kondraciuk, L., Kaiser, L., Metz, L., Boyd, M., Trebacz, M., Joglekar, M., Chen, M., Tintor, M., Meyer, M., Jones, M., Kaufer, M., Schwarzer, M., Shah, M., Yatbaz, M., Guan, M. Y., Xu, M., Yan, M., Glaese, M., Chen, M., Lampe, M., Malek, M., Wang, M., Fradin, M., McClay, M., Pavlov, M., Wang, M., Wang, M., Murati, M., Bavarian, M., Rohaninejad, M., McAleese, N., Chowdhury, N., Chowdhury, N., Ryder, N., Tezak, N., Brown, N., Nachum, O., Boiko, O., Murk, O., Watkins, O., Chao, P., Ashbourne, P., Izmailov, P., Zhokhov, P., Dias, R., Arora, R., Lin, R., Lopes, R. G., Gaon, R., Miyara, R., Leike, R., Hwang, R., Garg, R., Brown, R., James, R., Shu, R., Cheu, R., Greene, R., Jain, S., Altman, S., Toizer, S., Toyer, S., Miserendino, S., Agarwal, S., Hernandez, S., Baker, S., McKinney, S., Yan, S., Zhao, S., Hu, S., Santurkar, S., Chaudhuri, S. R., Zhang, S., Fu, S., Papay, S., Lin, S., Balaji, S., Sanjeev, S., Sidor, S., Broda, T., Clark, A., Wang, T., Gordon, T., Sanders, T., Patwardhan, T., Sottiaux, T., Degry, T., Dimson, T., Zheng, T., Garipov, T., Stasi, T., Bansal, T., Creech, T., Peterson, T., Eloundou, T., Qi, V., Kosaraju, V., Monaco, V., Pong, V., Fomenko, V., Zheng, W., Zhou, W., McCabe, W., Zaremba, W., Dubois, Y., Lu, Y., Chen, Y., Cha, Y., Bai, Y., He, Y., Zhang, Y., Wang, Y., Shao, Z., and Li, Z. (2024). Openai o1 system card.

OpenAI (2024). o3 model family technical report. Technical report, OpenAI. Retrieved from OpenAI's official model documentation.

OpenAI (2025). GPT-OSS-120B and GPT-OSS-20B: Open-weight language models. Version 1.0, Apache 2.0 license.

Park, C. F., Lee, A., Lubana, E. S., Yang, Y., Okawa, M., Nishi, K., Wattenberg, M., and Tanaka, H. (2025). Iclr: In-context learning of representations.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Grella, M., et al. (2023). Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*.

Poli, M., Thomas, A. W., Nguyen, E., Ponnusamy, P., Deisereth, B., Kersting, K., Suzuki, T., Hie, B., Ermon, S., Ré, C., et al. (2024). Mechanistic design and scaling of hybrid architectures. *arXiv preprint arXiv:2403.17844*.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., and Dean, J. (2022). Efficiently scaling transformer inference.

Press, O., Smith, N. A., and Lewis, M. (2022). Train short, test long: Attention with linear biases enables input length extrapolation.

Qin, Z., Han, X., Sun, W., Li, D., Kong, L., Barnes, N., and Zhong, Y. (2022). The devil in linear transformer.

- Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., et al. (2020). Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*.
- Ren, L., Liu, Y., Lu, Y., Shen, Y., Liang, C., and Chen, W. (2024). Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*.
- Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Spartan Books.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.
- Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization.
- Schlag, I., Irie, K., and Schmidhuber, J. (2021). Linear transformers are secretly fast weight programmers.
- Schmidhuber, J. (1992). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.
- Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Ådekk, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., and Hassabis, D. (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710.
- Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. (2024). Deepseekmath: Pushing the limits of mathematical reasoning in open language models.
- Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need.
- Shazeer, N. (2020). Glu variants improve transformer.
- Sherrington, D. and Kirkpatrick, S. (1975). Solvable model of a spin-glass. *Physical Review Letters*, 35(26):1792–1796.
- Snell, C., Lee, J., Xu, K., and Kumar, A. (2024). Scaling llm test-time compute optimally can be more effective than scaling model parameters.
- Stent, G. S. (1973). A physiological mechanism for hebbâs postulate of learning. *Proceedings of the National Academy of Sciences*, 70(4):997–1001.
- Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. (2023). Roformer: Enhanced transformer with rotary position embedding.
- Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. (2023a). Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*.

- Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. (2023b). Retentive network: A successor to transformer for large language models.
- Sun, Y., Li, X., Dalal, K., Xu, J., Vikram, A., Zhang, G., Dubois, Y., Chen, X., Wang, X., Koyejo, S., Hashimoto, T., and Guestrin, C. (2024). Learning to (learn at test time): Rnns with expressive hidden states.
- Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks.
- Toshniwal, S., Du, W., Moshkov, I., Kisacanin, B., Ayrapetyan, A., and Gitman, I. (2024). Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data.
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.
- von Oswald, J., Scherrer, N., Kobayashi, S., Versari, L., Yang, S., Schlegel, M., Maile, K., Schimpf, Y., Sieberling, O., Meulemans, A., Saurous, R. A., Lajoie, G., Frenkel, C., Pascanu, R., y Arcas, B. A., and Sacramento, J. (2025). Mesanet: Sequence modeling by locally optimal test-time training.
- Wang, D., Zhu, R.-J., Abreu, S., Shan, Y., Kergan, T., Pan, Y., Chou, Y., Li, Z., Zhang, G., Huang, W., and Eshraghian, J. (2025a). A systematic analysis of hybrid linear attention.
- Wang, J., Li, W.-D., Paliotta, D., Ritter, D., Rush, A. M., and Dao, T. (2025b). M1: Towards scalable test-time compute with mamba reasoning models.
- Wang, J., Paliotta, D., May, A., Rush, A., and Dao, T. (2024). The mamba in the llama: Distilling and accelerating hybrid models. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C., editors, *Advances in Neural Information Processing Systems*, volume 37, pages 62432–62457. Curran Associates, Inc.
- Wang, J., Paliotta, D., May, A., Rush, A. M., and Dao, T. (2025c). The mamba in the llama: Distilling and accelerating hybrid models.
- Wang, K. A., Shi, J., and Fox, E. B. (2025d). Test-time regression: a unifying framework for designing sequence models with associative memory.

- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Werbos, P. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.
- Willshaw, D. J., Buneman, O. P., and LONGUET-HIGGINS, H. C. (1969). Non-holographic associative memory. *Nature*, 222(5197):960–962.
- Xie, Y., Goyal, A., Zheng, W., Kan, M.-Y., Lillicrap, T. P., Kawaguchi, K., and Shieh, M. (2024). Monte carlo tree search boosts reasoning via iterative preference learning.
- Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., and Singh, V. (2021). Nyström-former: A nyström-based algorithm for approximating self-attention.
- Yan, J. N., Gu, J., and Rush, A. M. (2024). Diffusion models without attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8239–8249.
- Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. (2025). Qwen3 technical report.
- Yang, S., Kautz, J., and Hatamizadeh, A. (2024a). Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*.
- Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. (2023). Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*.
- Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. (2024b). Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*.
- Yang, S. and Zhang, Y. (2024). Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism.
- Yu, L., Jiang, W., Shi, H., Yu, J., Liu, Z., Zhang, Y., Kwok, J. T., Li, Z., Weller, A., and Liu, W. (2024). Metamath: Bootstrap your own mathematical questions for large language models.
- Yuan, J., Gao, H., Dai, D., Luo, J., Zhao, L., Zhang, Z., Xie, Z., Wei, Y. X., Wang, L., Xiao, Z., Wang, Y., Ruan, C., Zhang, M., Liang, W., and Zeng, W. (2025). Native sparse attention: Hardware-aligned and natively trainable sparse attention.
- Zhang, M., Arora, S., Chalamala, R., Wu, A., Spector, B., Singhal, A., Ramesh, K., and Ré, C. (2024a). Lollcats: On low-rank linearizing of large language models. *arXiv preprint arXiv:2410.10254*.

- Zhang, M., Arora, S., Chalamala, R., Wu, A., Spector, B., Singhal, A., Ramesh, K., and RĂ©, C. (2025). Lolcats: On low-rank linearizing of large language models.
- Zhang, M., Bhatia, K., Kumbong, H., and RĂ©, C. (2024b). The hedgehog—the porcupine: Expressive linear attentions with softmax mimicry.
- Zheng, L., Yuan, J., Wang, C., and Kong, L. (2023). Efficient attention via control variates.
- Zhong, S., Xu, M., Ao, T., and Shi, G. (2025). Understanding transformer from the perspective of associative memory.
- Zuo, J., Velikanov, M., Chahed, I., Belkada, Y., Rhayem, D. E., Kunsch, G., Hacid, H., Yous, H., Farhat, B., Khadraoui, I., Farooq, M., Campesan, G., Cojocaru, R., Djilali, Y., Hu, S., Chaabane, I., Khanna, P., Seddik, M. E. A., Huynh, N. D., Khac, P. L., AlQadi, L., Mokeddem, B., Chami, M., Abubaker, A., Lubinets, M., Piskorski, K., and Frikha, S. (2025). Falcon-h1: A family of hybrid-head language models redefining efficiency and performance.

Title of work:

Rethinking Sequence Models: Towards Scalable Test-Time Compute with Hybrid Reasoners

Thesis type and date:

Master's Thesis, August 10th, 2025

Supervision:

Prof. Dr. Cengiz Pehlevan, Harvard University
Prof. Dr. Valerio Mante, University of Zurich.

Student:

Name: Mohit Kulkarni
E-mail: mkulkarni@ethz.ch
Legi-Nr.:

Statement regarding plagiarism:

By signing this statement, I affirm that I have read and signed the Declaration of Originality, independently produced this paper, and adhered to the general practice of source citation in this subject-area.