LONDON'S GLOBAL UNIVERSITY
Department of Computer Science

Neural Memory Augmentation for Large Language Models

Ryan Pégoud[1]

Computational Statistics and Machine Learning

Supervised by Prof. Tim Rocktäschel and Davide Paglieri

Submission date: 15^{th} September 2025

[1]**Disclaimer:** This report is submitted as part requirement for the MSc Computational Statistics and Machine Learning at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Acknowledgements

I wish to thank my family, especially my mother and grandpa for their unwavering support during this year at UCL and for encouraging me to dedicate myself fully to all my pursuits. I also wish to remember my father, whose encouragements and dedication set me on this path.

Thank you Elisa for your invaluable companionship, patience and kindness during this year. You brought colours to London I didn't even know existed.

To Chris, thank you for always believing in me, supporting my growth as a person and a scientist. You have been a second brother to me and I couldn't have done this alone. I would also like to thank my other CSML classmates, particularly Nick and Panos. Our trip to Italy and the 10-hour study sessions in Panos' kitchen are memories I won't ever forget.

To my flatmates Divya and Jiali, thank you for making our flat feel like home and listening to my daily venting about how difficult and stressful this program was.

Particular thanks to my friends back home, Justin, Quentin, Mathieu, Alexandra and Hanaé for refilling my tank every time I visit.

Finally, I would like to thank my advisors. To Prof. Tim Rocktäschel for welcoming me as part of the UCL DARK Lab throughout my Masters, supporting my research ideas and applications. To Davide Paglieri for his insightful supervision and actionable feedback on this project.

Abstract

A fundamental constraint of Large Language Models (LLMs) is their fixed context window, limiting their ability to perform tasks requiring long-range memory and reasoning. While various methods attempt to scale the context window by making the attention mechanism more efficient, equipping LLMs with dedicated, online-learning memory systems represents a more scalable and cognitively-plausible paradigm. This thesis introduces MemoryLlama, a novel architecture that integrates a test-time trainable memory module, inspired by the Titans model, into a pre-trained Llama-3 foundation model. We investigate several architectural injection strategies and adapt Parameter-Efficient Fine-Tuning (PEFT) methods to co-train the memory and backbone components. Empirical evaluation reveals that memory injection is highly sensitive to training dynamics and hyperparameters. Despite displaying promising initial use of the memory modules, both architectural variants ultimately suffered catastrophic forgetting. Analysis of training metrics and ablation studies suggest that this instability arises when our LLM is jointly fine-tuned with the memory modules, while at the same time being a requirement for performance. Beyond MemoryLlama itself, this thesis contributes a rigorous characterisation of the challenges of training such hybrid models. By providing a diagnostic framework, a detailed analysis of failure modes, and directions for future research, it offers insights for building more robust and capable memory-augmented LLMs.

All experiments are fully reproducible, with code publicly available at https://github.com/RPegoud/Hercules.

Contents

List of Figures

List of Tables

0.1 Acronyms

0.1.1 Memory Augmentation

MAL Memory as a Layer (Titans architecture)

MAG Memory as a Gate (Titans architecture)

MAC Memory as a Context (Titans architecture)

MANN Memory-Augmented Neural Networks

MCP Memory CoProcessor

ELMA Embedding-Level Memory Augmentation

ALMA Attention-Level Memory Augmentation

GeMA Generalised Memory Augmentation

0.1.2 Deep Learning Architectures

RNN Recurrent Neural Network

LSTM Long Short-Term Memory Network

MLP Multi-Layer Perceptron

HRM Hierarchical Reasoning Model

0.1.3 Algorithms and Methods

SGD Stochastic Gradient Descent

PEFT Parameter-Efficient Fine-Tuning

LoRA Low-Rank Adapatation

ReLU Rectified Linear Unit

LayerNorm Layer Normalisation

RMSNorm Root-Mean-Square Normalisation

0.1.4 Hardware, Libraries and Benchmarks

FLOPS Floating-point Operations

GPU Graphics Processing Unit

bf16 Brain Floating Point format (occupying 16 bits in computer memory)

XLA Accelerated Linear Algebra

ARC Abstraction and Reasoning Corpus

0.2 Notations

Symbol	Description
τ	Sequence length
θ	Model parameters
$\varphi(\cdot)$	Arbitrary activation function or non-linearity
$\sigma(\cdot)$	Sigmoid activation function
x	Input sequence
$\tilde{x}$	Memory-augmented input sequence
h	Hidden or recurrent state of a sequence model
Q	Query vector
K	Key vector
V	Value vector
a	Attention scors (vector or matrix)
$\mathcal{O}(\cdot)$	Computational or memory complexity
$\mathcal{M}$	Memory module
Attn	Attention (operation)
SW-Attn	Sliding-Window Attention
$\otimes$	Non-linear gating
$\square$	Concatenation

Chapter 1

Introduction

1.1 Motivation

Large Language Models (LLMs) have transformed the field of artificial intelligence, exhibiting advanced capabilities in language understanding, reasoning, and generation (Devlin et al., 2019, Radford et al., 2018, Liu et al., 2019). Despite these advances, their effectiveness is constrained by a fundamental architectural bottleneck: the self-attention mechanism, whose computational and memory requirements scale quadratically with input sequence length. This scaling property limits the ability of LLMs to process long documents, sustain coherent dialogue, and learn from extended interactions, thereby restricting their applicability in complex, real-world scenarios.

To address this challenge, the research community has pursued two principal directions. The first seeks to enlarge the context window by designing more efficient attention mechanisms. Approaches such as sparse attention (Beltagy et al., 2020), linear attention (Katharopoulos et al., 2020), and hardware-aware optimisations such as FlashAttention (Dao et al., 2022) have considerably extended the feasible sequence length. While effective, these methods remain constrained by the requirement that the model processes and retains the entire sequence within its working memory.

This thesis pursues a complementary direction: enhancing the model's capacity to selectively retain and utilise information through an external, long-term memory system. Rather than continuously extending the context window, this approach augments LLMs with a dedicated memory coprocessor. Such an architecture not only addresses the challenge of processing long sequences, but also represents a foundational step towards more capable and autonomous AI systems. Higher-order cognitive abilities, including multistep

reasoning, strategic planning, and continual learning, rely fundamentally on mechanisms for storing, retrieving, and synthesising information over extended timescales. Memory, in this sense, is a prerequisite for genuine agency.

This work investigates a promising memory architecture introduced in the Titans model (Behrouz et al., 2024), which incorporates a memory module capable of updating itself online during test time. By adapting and integrating this specialised coprocessor into a large pre-trained model such as Llama-3 (Grattafiori et al., 2024), we aim to combine the broad knowledge base of a foundation model with the dynamic, adaptive properties of an external memory module. This integration establishes a practical and efficient paradigm for augmenting the cognitive capabilities of existing LLMs without incurring the prohibitive computational cost of training models from scratch.

1.2 Research Questions

The overarching goal of this thesis is to investigate the viability and dynamics of integrating an online, test-time trainable memory module into a pre-trained, decoder-only Transformer. We break dowm this goal into the following core research questions:

1. **Performance Impact:** Can a pre-trained LLM's long-term recall and performance on memory-intensive tasks be measurably improved by injecting a Titans-style memory module?

2. **Architectural Integration:** What are the performance and learning trade-offs associated with different memory injection strategies, particularly concerning the placement of memory-augmented blocks within the LLM's decoder stack?

3. **Training Dynamics:** What are the dynamics of training a randomly-initialised memory module within a pre-trained LLM? Is it necessary to co-adapt the base model's parameters (e.g., via PEFT), and what are the associated stability challenges?

4. **Scalability:** Do these hybrid memory-augmented architectures exhibit predictable scaling properties with respect to the size of the base LLM and the memory module itself?

1.3 Contributions

We propose a novel architecture for memory-augmentation of pre-trained LLMs called MemoryLlama. This architecture integrates and adapts Titans memory modules to the Llama-3 framework. These modules are injected within the LLM's decoder block to imbue the attention mechanism with long-past, out-of-context information. The placement of these memory-augmented decoder blocks represents a crucial research question. We therefore decline MemoryLlama in three variants: Embedding-Level Memory Augmentation (ELMA), Attention-Level Memory Augmentation (ALMA) and Generalised Memory Augmentation (GeMA) which represent distinct conceptual approaches to the placement of memory-augmented decoder blocks within the LLM's stack of decoder blocks. ELMA and ALMA are fine-tuned on a standard language modeling task and tested on memory-intensive tasks while GeMA is left to future work due to computational constraints. We conduct ablations studies to gain further insights on the training dynamics of our models and the requirements for succesful integration of memory modules in pre-trained LLMs. Our results suggest that fine-tuning the LLM jointly with the memory modules benefits performance but eventually becomes a source of instability. While this specific version of our achitecture is tailored for Llama-3 models, the methodology described in this thesis can be easily applied to any open-source LLM.

1.4 Thesis Structure

This thesis is structured as follows: Chapter 2 reviews background on sequence models, starting from vanilla RNNs and LSTMs which introduce the concepts of recurrent updates, gating mechanisms and context, which are essential to understand the design of Titans. Then, the attention mechanism, its use in LLMs, main limitations and variants are discussed before diving deeper into the Llama architecture, which we use as foundation for MemoryLlama. The concepts introduced so far culminate in a review of Titans and its neural memory networks, which we adopt as our main memory coprocessor. Finally, we briefly introduce Parameter Efficient Finetuning methods since our proposed approach borrows some key design choices from Low-Rank Adaptation (LoRA) and is conceptually aligned with the PEFT framework. Chapter 3 contextualises our approach and contribution within the broader litterature of memory-augmented models and attention-less sequence modeling architectures. Chapter 4 presents the MemoryLlama architecture and outlines the experimental framework while Chapter 5 reports the empirical results of our method. Finally,

Chapter 6 concludes this thesis by summarising the key findings, discussing future work and outlines future evolutions of the presented framework.

Chapter 2

Background

2.1 A History of Sequence Models

2.1.1 Recurrent Neural Networks

Historically, recurrent neural networks (RNNs) have emerged as the main architecture for language modeling tasks considered in this thesis. At their core, RNNs improve upon regular feedforward networks by using some form of parameter sharing. This enables the processing of sequences with varying lengths and of specific inputs that might appear at different positions within the sequence. Generally, such parameter sharing is achieved by making each member of the network's output a function of the previous outputs, resulting in a recursive formulation of the model. Specifically, RNNs operate on a sequence containing vectors $x^{(t)}$ with the time index t ranging from one to τ, the sequence length. Note that we neglected the batch index for simplicity, but the sequence length is usually different across members of the batch. A general formulation of recurrent neural networks is the following:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{2.1}$$

Where h refers to the state of the hidden units and θ to the network's parameters. In tasks such as language modeling that require predicting the next word in a sequence based on previous words, the network typically learns to use $h^{(t)}$ as an approximate summary of the task-relevant information contained in the past sequence of inputs up to t. *Unfolding* the computational graph for a short sequence highlights how parameters are shared across time. For $\tau = 3$, we have:

$$h^{(3)} = f(h^{(2)}, x^{(3)}; \theta) \tag{2.2}$$
$$= f(f(h^{(1)}, x^{(2)}; \theta), x^{(3)}; \theta) \tag{2.3}$$
$$= f(f(f(h^{(0)}, x^{(1)}; \theta), x^{(2)}; \theta), x^{(3)}; \theta) \tag{2.4}$$

Note that h_0 is typically initialised as a zero vector or treated as a learned parameter. A general formulation of recurrent neural networks can be represented as a computational graph where the model maps as sequence of inputs x to a corresponding sequence of outputs o. A loss function $\mathcal{L}$ measures the difference between the outputs and the training targets y, for instance the cross-entropy loss. The model is parameterised by three weight matrices $\theta = \{U, V, W\}$ where U, V, W parameterise input-to-hidden, hidden-to-hidden and hidden-to-output connections respectively. Assuming discrete outputs (for instance tokens in language modeling) and an arbitrary activation function $\varphi(\cdot)$, the update equations can be written as:

$$h^{(t)} = \varphi(b + Wh^{(t-1)} + Ux^{(t)}) \tag{2.5}$$
$$o^{(t)} = c + Vh^{(t)} \tag{2.6}$$
$$\hat{y}^{(t)} = \text{softmax}(o^{(t)}) \tag{2.7}$$

Where b and c are bias vectors. Gradient computations in this setting require a forward and backward pass through the unrolled computational graph, this algorithm is called back-propagation through time (BPTT).

Importantly, the runtime of this gradient computation is $\mathcal{O}(\tau)$ per time step, for a total of $\mathcal{O}(\tau^2)$ and **can not be reduced by parallelisation** since the forward pass is inherently sequential. This limits the scaling potential of RNNs with modern GPUs, which will be a computational bottleneck later in this thesis. Additionally, states computed during the forward pass must be stored until they are required in the backward pass, hence the memory cost is $\mathcal{O}(\tau)$.

Another disadvantage of RNNs is their inability to represent long-term dependencies. As discussed previously, the computational graph of an RNN can be unrolled and interpreted as a deeply nested composite function where the number of nested operations depends on τ. This deep structure is the primary cause of the vanishing gradient problem,

where the error signal induced by inputs over long sequences fail to propagate back through the network.

Consider the gradient of the loss function $\mathcal{L}$ at time t with respect to a hidden state $h^{(k)}$, with $k < t \leq \tau$, using the chain rule, we obtain:

$$\frac{\partial \mathcal{L}^{(t)}}{\partial h^{(k)}} = \frac{\partial \mathcal{L}^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial h^{(k)}} \tag{2.8}$$

The critical component of this derivative is the Jacobian $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ which measures how a change in the hidden state a step k affects the hidden state at a future step t. We can further expand this term using the chain rule:

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial h^{(t-2)}} \cdots \frac{\partial h^{(k+1)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial h^{(i)}}{\partial h^{(i-1)}} \tag{2.9}$$

Where the Jacobian for a single time step is given by the update equation $h^{(i)} = \varphi(b + Wh^{(i-1)} + Ux^{(i)}) = a(z^{(i)})$. Hence the derivative:

$$\begin{aligned} \frac{\partial h^{(i)}}{\partial h^{(i-1)}} &= \frac{\partial}{\partial h^{(i-1)}} \varphi(b + Wh^{(i-1)} + Ux^{(i)}) && (2.10) \\ &= \operatorname{diag}\left(\varphi'(b + Wh^{(i-1)} + Ux^{(i)})\right) W && (2.11) \\ &= \operatorname{diag}\left(\varphi'(z^{(i)})\right) W && (2.12) \end{aligned}$$

Substituting back in Equation 2.9, we get:

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \left(\operatorname{diag}\left(\varphi'(z^{(i)})\right) W\right) \tag{2.13}$$

This equation illustrates the core problem behind vanishing gradients, namely that the gradient from step t to step k is a product of $t - k$ matrices. We may bound the norm of this matrix as:

$$\left|\frac{\partial h^{(t)}}{\partial h^{(k)}}\right| \leq \prod_{i=k+1}^{t} \left|\operatorname{diag}\left(\varphi'(z^{(i)})\right) W\right| \leq (\gamma_a \gamma_W)^{t-k} \tag{2.14}$$

Where $\gamma_W = ||W||_2$ is the spectral norm of W and γ_a is an upper bound on activation function's derivatives. If the term $\gamma_a \gamma_W$ is consistently less than 1, the magnitude of the gradient will suffer an exponential decay, resulting in the model being unable to model dependencies between inputs at position k and t since the error signal cannot propagate

back.

RNNs are particularly vulnerable to this problem since the repeated matrix multiplications always involve the same weight matrix W. Therefore, if γ_W is not exactly 1, then gradients are bound to vanish or explode over long time horizons. Conversely, RNNs often use activation functions such as the sigmoid or hyperbolic tangent. The derivatives of which are always less than or equal to 1, and more importantly, they approach 0 as the neuron saturates. This ensures the term $\text{diag}(\varphi'(z^{(i)}))$ often contains small values, causing the gradient to shrink and eventually vanish over long sequences. In practice, the experiments in Bengio et al. (1994) show that as we expand the horizon of the dependencies that need to be modeled, gradient-based optimisation becomes increasingly unstable, with the probability of successful training of a traditional RNN via SGD rapidly reaching 0 for sequences of only length 10 or 20.

2.1.2 Long Short-Term Memory Networks and Gating

Based on the limitations of traditional RNNs, significant research efforts were dedicated to creating architectures containing paths through time with derivatives that neither vanish nor explode. Such architectures include Long Short-Term Memory Networks (LSTMs, Hochreiter and Schmidhuber (1997)) and other kind of gated RNNs. The crucial innovation behind these architectures is the concept of **gating**, improving the management of the network's hidden state. To this day, gating is still a crucial compoment of state-of-the-art sequence modeling architectures, including Titans, the focus of this thesis.

Contrarily to traditional RNNs that propagate the full hidden state at each time step, LSTMs learn to set input-dependent connection weights and forget irrelevant information by setting state values to zero. In LSTMs, the hidden state $h^{(t)}$ is augmented by a cell state $c^{(t)}$, which is controlled by three individual gates, each affecting the flow of information in a specific way. They are implemented as a single neuron with a sigmoid activation $\sigma(\cdot)$ and individual parameters.

For a specific input x and hidden state h, the **forget gate** $f^{(t)}$ is responsible for clearing irrelevant information from the cell state whereas the **input gate** $i^{(t)}$ determines which parts of the cell state are to be updated. Finally, the **output gate** $o^{(t)}$ selects relevant elements of the cell state that constitue the output hidden state. The sigmoid activation enables these gates to filter information by returning weights between 0 and 1 determining the proportion of each input element that should be carried forward to the next time steps:

$$f^{(t)} = \sigma(b^f + U^f x^{(t)} + W^f h^{(t-1)}) \tag{2.15}$$
$$i^{(t)} = \sigma(b^i + U^i x^{(t)} + W^i h^{(t-1)}) \tag{2.16}$$
$$o^{(t)} = \sigma(b^o + U^o x^{(t)} + W^o h^{(t-1)}) \tag{2.17}$$

Where b, U and W refer to the biases, input weights and recurrent weights of a specific cell.

Another neuron $\tilde{c}^{(t)}$ is responsible for proposing an update to the cell state $c^{(t)}$. In contrast to the previous gates, this neuron uses a tanh activation, allowing the proposal to add and subtract information from the cell state.

$$\tilde{c}^{(t)} = \tanh(b^c + U^c x^{(t)} + W^c h^{(t-1)}) \tag{2.18}$$

The information processed by the gates is then aggregated to update the LSTM's cell state and hidden state. The cell state is updated by adding the previous cell state with the candidate state, respectively weighted by the forget and input gates. The new hidden state is obtained by passing the cell state through a tanh activation and multiplying it (element-wise) with the output gate.

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \tag{2.19}$$
$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)}) \tag{2.20}$$

Note that $h^{(t)}$ is used as the output of the unit as well as the hidden state for the next step, letting the model remember its last output. As such, the hidden state acts as a **short-term memory** whereas the cell state is used as **long-term memory**, hence the name of the architecture. This interplay between long and short-term memory is central to this thesis and will be expanded upon later on.

This design proposes an effective solution to the vanishing gradient problem. Indeed, LSTMs propagate gradient through time by relying on the cell state, which uses an additive update rule as shown in Equation 2.19. Once again, consider the Jacobian involved in the propagation of gradients through time[1], for a single time step, we get:

[1]While the full Jacobian is more complex because the gates themselves depend on $h^{(t-1)}$ (and thus on $c^{(t-1)}$), this direct path through the forget gate is the most critical for gradient flow.

$$\frac{\partial c^{(t)}}{\partial c^{(t-1)}} = \text{diag}(f^{(t)}) \tag{2.21}$$

Unrolling across time, the gradient becomes:

$$\frac{\partial c^{(t)}}{\partial c^{(k)}} = \prod_{i=k+1}^{t} \text{diag}(f^{(i)}) \tag{2.22}$$

In contrast to the RNN update, the cell state update does not involve repeated matrix multiplication or activation derivatives. Instead, the gradient flow through time is regulated by the forget gate. If $f^{(i)} = 1$, then the gradient is propagated to the previous time step without shrinking or distortion. Contrarily, setting $f^{(i)} = 0$ cuts off the gradient flow for a specific input.

The improved handling of the memory flow in LSTMs have enabled the processing of considerably longer sequences.

2.1.3 Encoder-Decoder Architectures and Context

Previously, we introduced recurrent models designed to capture long-term dependencies through a hidden state. These models, however, were restricted to producing output sequences of the same length as the input. Many tasks, such as translation or question answering, instead require variable-length outputs.

Encoder-decoder architectures address this by mapping an input sequence $X = (x^{(1)}, \ldots, x^{(n_x)})$ to a compressed **context** vector C, which a decoder then uses to generate an output sequence $Y = (y^{(1)}, \ldots, y^{(n_y)})$. This allows n_x and n_y to differ, unlike earlier recurrent models where $n_x = n_y = \tau$. In practice, both encoder and decoder can be implemented with RNNs or LSTMs.

A simple choice is to define the context as the encoder's last hidden state, $C = h^{(n_x)}$. Yet, as shown by Bahdanau et al. (2014), compressing an entire sequence into a fixed-size vector creates a severe bottleneck: a single hidden state cannot capture the nuances of long inputs, while making C very large is inefficient for shorter ones.

2.1.4 The Attention Mechanism

The attention mechanism provides an elegant solution to the information bottleneck of fixed-size context vectors introduced in the previous section. At its core, attention allows a

model to dynamically focus on the most relevant parts of an input sequence when producing an output, rather than relying on a single compressed representation.

An Associative Memory Perspective

Attention can be understood as a form of **differentiable, soft associative memory**. The goal is to retrieve information from a set of **values** (V) based on the similarity of a **query** (Q) to a corresponding set of **keys** (K). Instead of a hard lookup that returns a single value, attention performs a "soft" lookup by computing a weighted average of all values, where the weights are determined by the query-key similarity. By doing so, attention creates a new, context-aware representation of tokens in a sequence by learning to "attend" to other relevant tokens.

Scaled Dot-Product Attention

In modern architectures like the Transformer, this mechanism is implemented using **scaled dot-product attention**. Given an input sequence representation $X \in \mathbb{R}^{\tau \times d}$, where d is the embedding dimension, we first project X into three matrices using learned weight matrices (W_q, W_k, W_v):

- **Queries ($Q = XW_q$):** A set of vectors representing what each token is "looking for".
- **Keys ($K = XW_k$):** A set of vectors representing what information each token "offers".
- **Values ($V = XW_v$):** A set of vectors representing the actual content of each token to be passed forward.

The dimensions are typically $Q, K \in \mathbb{R}^{\tau \times d_k}$ and $V \in \mathbb{R}^{\tau \times d_v}$ where $d_k, d_v < d$ are hyperparameters. The attention output is then computed with the following formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.23}$$

The computation can be broken down into four steps:

1. **Score:** The similarity between every query and every key is computed via a matrix multiplication, QK^T.
2. **Scale:** The scores are divided by $\sqrt{d_k}$. This scaling factor prevents the dot products from becoming too large, which helps to stabilise gradients during training.

3. **Softmax:** A softmax function is applied to the scaled scores, converting them into positive attention weights that sum to 1.

4. **Output:** The attention weights are multiplied by the value matrix V to produce a weighted sum.

Multi-Head Attention

To enhance the model's representational power, the attention mechanism is distributed between multiple **attention heads**. Instead of performing one large attention calculation, multi-head attention learns separate projection matrices (W_q^i, W_k^i, W_v^i) for each head i. Aside from the computational advantage of having parallel attention heads, we could interpret attention as a kernel matrix and multi-head attention as a way to capture different notions of similarity. Thus, each head learns to focus on different types of information or relational patterns in different "representation subspaces". The outputs from each head are concatenated and passed through a final linear projection to produce the final output of the layer.

Complexity Analysis

The core limitation of attention lies in its cost. Specifically, the computational cost of scaled dot-product attention (in terms of FLOPS) scales with $\mathcal{O}(n^2 d)$ while the memory cost is $\mathcal{O}(n^2 + nd)$. These results are detailed in Table 2.1 and Table 2.2 respectively. Practically, this severely limits the size of the "context window" (the sequence length over which attention is computed) of attention-based models as discussed in subsection 2.2.3. Still, attention is the backbone of the most popular sequence modeling architectures for two main reasons:

- **In-context retrieval**: Attention enables each token in a sequence to attend to every other token, regardless of their relative position and distance in the sequence. Thus, information stored within the context window is precisely retrieved by attention-based models. In contrast, recurrent models such as RNNs rely on noisy hidden states to model past information.

- **Parallel computation**: A crucial consideration in the context of this thesis is that attention is inherently **non-sequential** and can therefore be **parallelised**, in contrast to recurrent networks. This enables attention-based models to scale to larger data regimes by leveraging modern GPUs in spite of the quadratic scaling.

Table 2.1: Computational Complexity of Scaled Dot-product Attention

Operation	**Input Dimensionalities**	**Complexity (FLOPS)**
QK^T	$(n \times d_k) \times (d_k \times n)$	$\mathcal{O}(n^2 d_k)$
softmax$(\cdot)$	$(n \times n)$	$\mathcal{O}(n^2)$
AttentionScores $\times V$	$(n \times n) \times (n \times d_v)$	$\mathcal{O}(n^2 d_v)$

Table 2.2: Memory Cost of Scaled Dot-product Attention

Matrix	**Symbol**	**Dimensionality**
Queries, Keys, Values	Q, K, V	$\mathcal{O}(nd_k)$, $\mathcal{O}(nd_v)$
Attention Scores	QK^T	$\mathcal{O}(n^2)$
Output	Attention(Q, K, V)	$\mathcal{O}(nd_v)$

2.1.5 Transformers

The Transformer model, first introduced in the seminal "Attention is All You Need" (Vaswani et al., 2017), is an attention-based encoder-decoder architecture that rapidly obtained state-of-the-art results in several sequence generation tasks, such as machine translation. The Transformer is a complex model that uses several building blocks, as introduced below.

2.1.6 Self-Attention and Cross-Attention

In a Transformer, the encoder processes the entire input sequence in parallel, while the decoder autoregressively generates the output sequence one token at a time. Each component is built from layers that include multi-head attention and feedforward layers. Specifically, the encoder uses **self-attention** (Figure 2.1), which computes dependencies between different positions within a single sequence. Given an input sequence, self-attention computes a contextualized representation for each token by using it as a query to attend to all other tokens in the sequence (including itself). **Cross-attention** (Figure 2.2), used in the decoder, allows the model to attend to the encoder's output. At each decoding step, the decoder uses its own hidden states as queries and the encoder's output as keys and values:

$$z_i^{\text{dec}} = \text{Attention}(q_i^{\text{dec}}, K^{\text{enc}}, V^{\text{enc}})$$

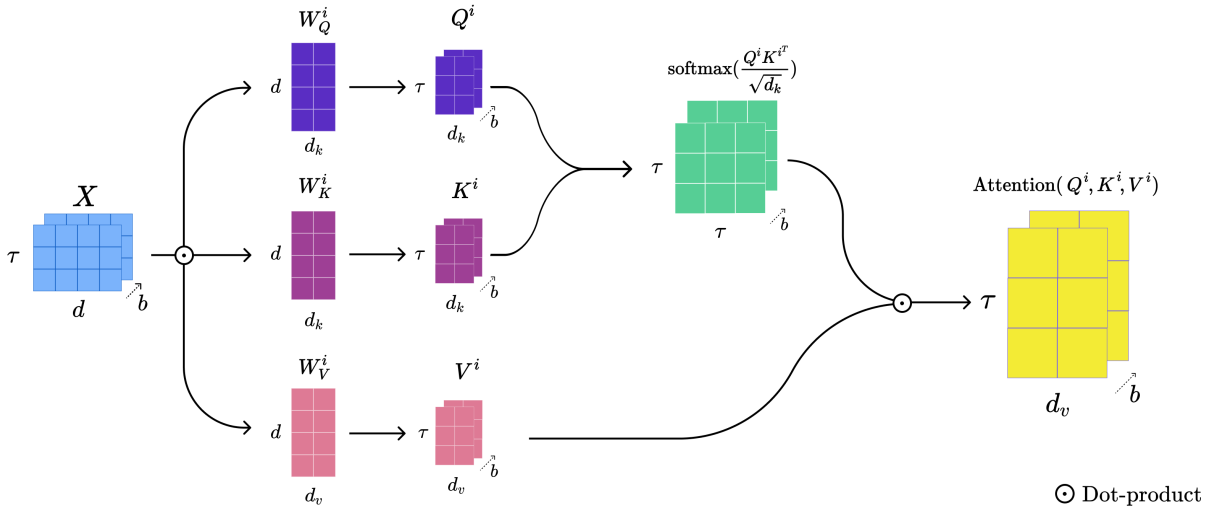


Figure 2.1: Representation of the self-attention mechanism for a single attention head, as typically encountered in the Transformer encoder. Here, b represents the batch size.

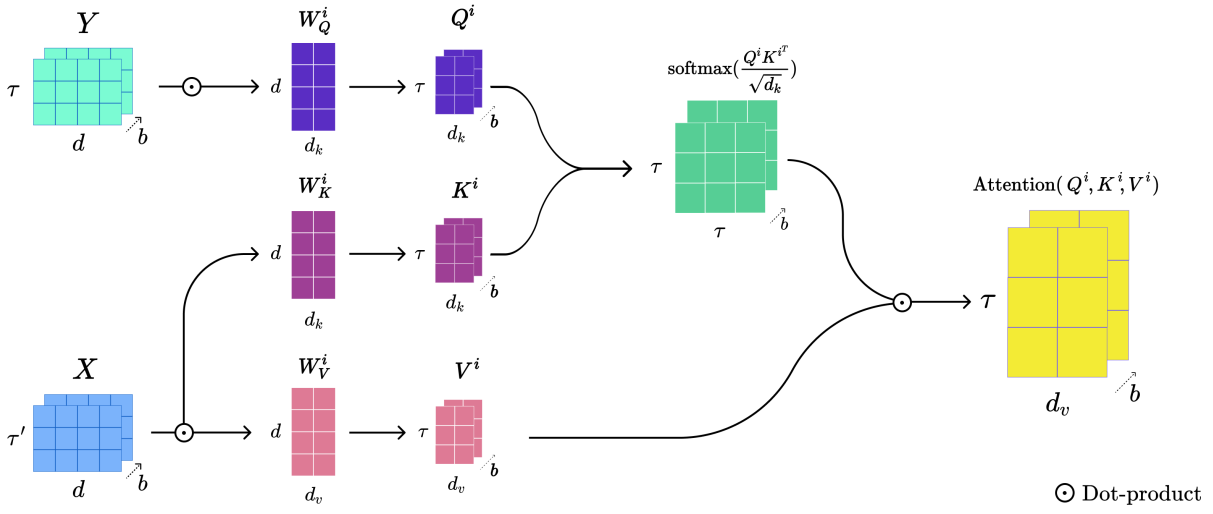


Figure 2.2: Representation of the cross-attention mechanism for a single attention head, as typically encountered in the Transformer decoder.

Where q_i^{dec} is derived from the decoder's current state, $K^{\text{enc}}, V^{\text{enc}}$ are the outputs of the encoder stack, and z_i is the decoder's output at position i. During training, the decoder receives the full target sequence, allowing all attention operations to be computed in parallel. However, to preserve the autoregressive property, a **causal mask** is applied to the decoder's self-attention layer to prevent any position from attending to future tokens.

2.1.7 Positional Embeddings

Importantly, attention is **permutation-invariant**, it does not natively process information about the order of tokens in a sequence. To circumvent this limitation, Transformers add positional encodings to the input embeddings. The original paper uses a sinusoidal basis:

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right) \tag{2.24}$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right) \tag{2.25}$$

where i is the position, j is the dimension index, and d is the model's embedding dimension. These encodings are deterministic and can generalize to sequence lengths not seen during training. Furthermore, the position vector for any location $t + \phi$ is a linear function of the vector at position t, which is a useful inductive bias for modeling relative positions.

2.1.8 Assembling the Transformer

The encoder and decoder are each composed of a stack of N identical blocks. Each block contains sublayers for multi-head attention and a position-wise feedforward network, with residual connections and layer normalisation applied around each sublayer. The decoder has a similar structure but includes a third sublayer for cross-attention over the encoder's output. The self-attention sublayer is also masked to preserve the autoregressive property.

Parallelism in Transformers

Despite their quadratic time and memory complexity, Transformers are the most popular approach in most sequence modeling tasks. This is mainly due to their non-sequential nature, which allows them to benefit from several layers of parallelisation:

- **Token-level parallelism**: During training, both the encoder and the decoder compute self-attention and cross-attention over all tokens in parallel. At test-time, the

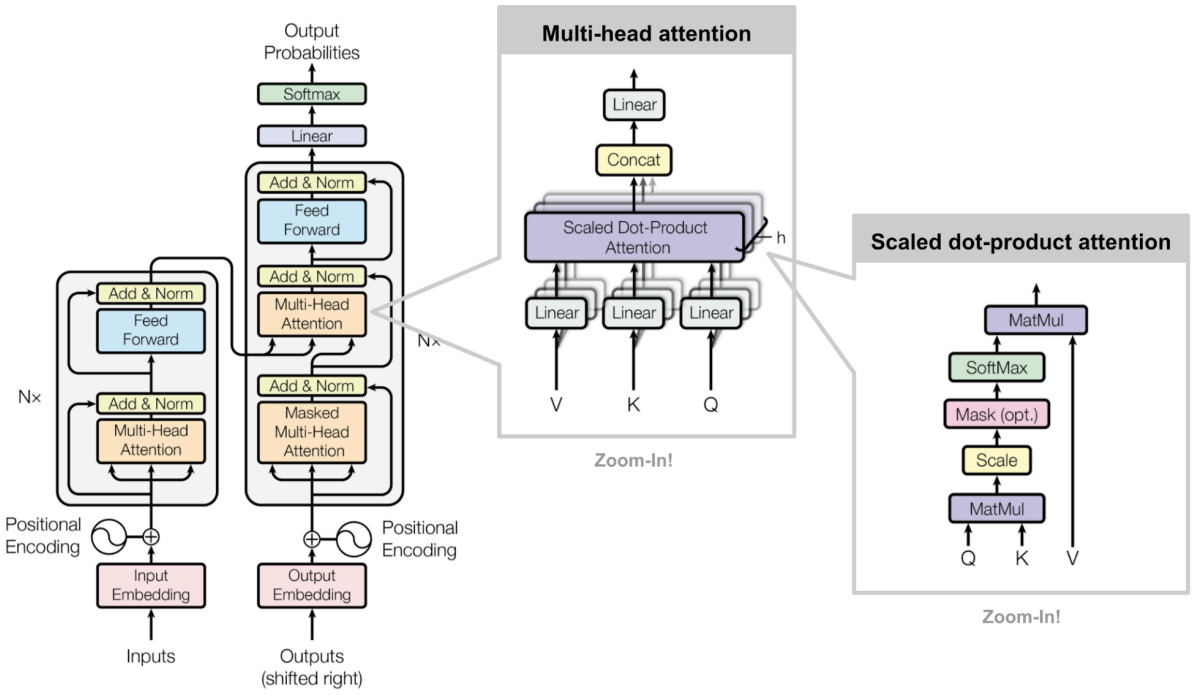


Figure 2.3: The full Transformer architecture, source: Weng (2018)

encoder still processes input sequences in parallel while the decoder performs autoregressive generation.

- **Head-level parallelism**: Multi-head attention breaks down the embedding dimension of the input sequences into smaller chunks, which are processed in parallel by independent attention heads.
- **Batch-level parallelism**: Multiple sequences are batched and processed at once.
- **Device parallelism**: For very large models, different parts of the model are split across several GPUs or nodes.

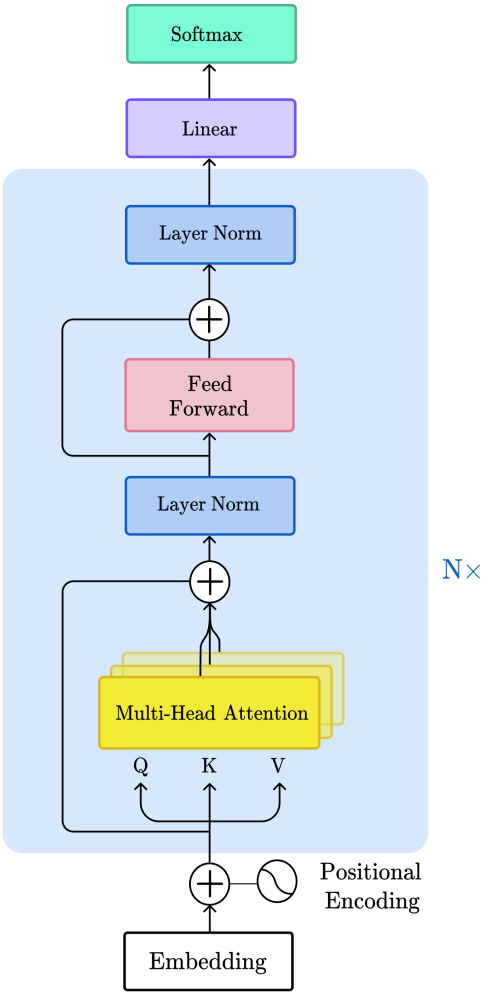
These levels of parallelism allowed Transformers to leverage advances in GPU architecture and scale efficiently across massive clusters, ultimately ushering in the era of large language models.

2.2 Large Language Models

2.2.1 GPT Models

Closely following the invention of Transformers, Radford et al. (2018) propose a new approach to language modeling and a large variety of downstream tasks. Namely, they introduce the GPT (Generative Pre-trained Transformer) model, a stack of N Transformer decoders as depicted in Figure 2.4, that is trained in two stages. First, GPT is pre-trained on a large unlabeled dataset $X = \{x^{(1)}, \ldots, x^{(n)}\}$ by maximising the log-likelihood of the next token in a sequence:

Figure 2.4: The GPT architecture.



$$\mathcal{L}_{\text{PT}}(X) = \sum_i \log P(x^{(i)}|x^{(i-k)}, \ldots, x^{(i-1)}; \theta) \tag{2.26}$$

Where k is the size of the context window, the conditional probability P is obtained by applying a softmax layer to the model outputs and θ represents the model parameters.

Following this pre-training stage, the model undergoes supervised fine-tuning on a downstream task such as text classification or question-answering. This time, the existence of a labeled dataset $\mathcal{C}$ composed of sequence of tokens $x^{(1)}, \ldots, x^{(m)}$ with an associated label y is assumed. The input sequences are passed through the pre-trained model and then fed into an added linear output layer with parameters W_y to predict y. The predictive distribution is thus given by:

$$P(y|x^{(1)}, \ldots, x^{(m)}; \theta) = \text{softmax}(hW_y) \tag{2.27}$$

Where h denotes the output of the last Transformer block. The objective is then to maximise the following log-likelihood:

$$\mathcal{L}_{\text{FT}}(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^{(1)}, \ldots, x^{(m)}; \theta) \tag{2.28}$$

Additionally, language modeling can serve as an auxiliary objective during the fine-tuning stage, thereby improving generalisation and accelerating convergence. The finetuning loss then becomes:

$$\mathcal{L}(\mathcal{C}) = \mathcal{L}_{\text{FT}} + \lambda \mathcal{L}_{\text{PT}} \tag{2.29}$$

Where λ controls the contribution of the auxiliary objective to the loss.

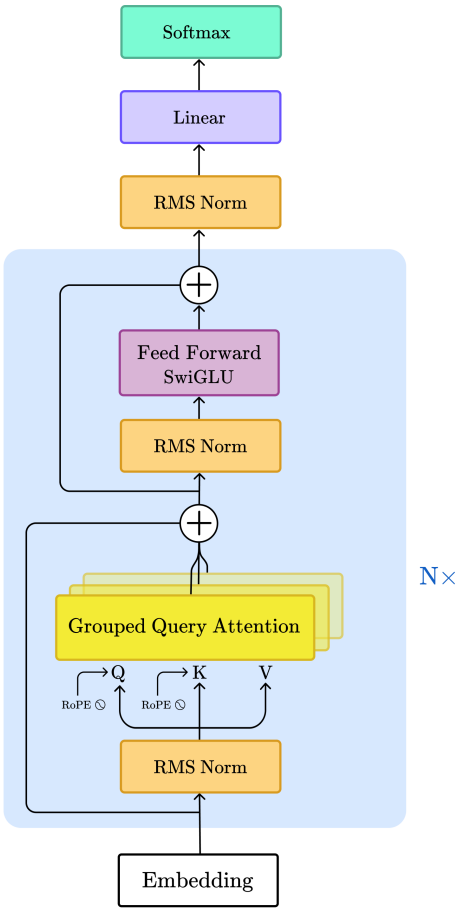
Subsequent models such as GPT3 (Brown et al., 2020) forego the fine-tuning phase with the intent of being general language models (often referred to as large language models or LLMs) that can perform zero-shot task transfer (solving a task without supervised fine-tuning) or be fine-tuned to improve their performance on a specific downstream task. The latter is called **transfer learning** and is a core component of this thesis.

Since the introduction of GPT3, large language model architectures have advanced considerably, with each component of the original design being independently optimised. These developments have culminated in the Llama-3 architecture, which serves as the backbone of our proposed architecture.

2.2.2 Llama Models

In this thesis, we focus primarily on open-source, pre-trained LLMs. A popular choice for such projects is the Llama (Large Language Model Meta AI, Touvron et al. (2023), Grattafiori et al. (2024)) family of models. These models come in different sizes, generally ranging from 1 to 90 billion parameters. While inheriting from the decoder-only architecture of GPT models, their implementation contains some key differences, as summarised in Figure 2.5.

Figure 2.5: The Llama architecture.



Rotary Positional Embeddings

Llama models use **rotary positional embeddings** (Su et al., 2021) as opposed to the absolute positonal encodings introduced in subsection 2.1.7. These rotary embeddings are applied directly to queries and keys to allow the attention mechanism itself to handle relative positions. Assuming d-dimensional embeddings where d is an even number, RoPE treats the embeddings as pairs of two dimensions (x_i, x_{i+1}) and applies a rotation matrix based on the token's position, m:

$$\begin{pmatrix} x'_{m,i} \\ x'_{m,i+1} \end{pmatrix} = \begin{pmatrix} \cos(m\theta_i) & -\sin(m\theta_i) \\ \sin(m\theta_i) & \cos(m\theta_i) \end{pmatrix} \begin{pmatrix} x_{m,i} \\ x_{m,i+1} \end{pmatrix} \tag{2.30}$$

Where $\theta_i = 10000^{-2i/d}$ is a predefined frequency that depends on the embedding dimensionality. Importantly, the dot product of a rotated query at position m and a rotated key at position n depends only on their relative position $m - n$:

$$(R_m q)^T (R_n k) = q^T R_{m-n}^T k \tag{2.31}$$

This injects relative positional information directly into the attention score calculation without altering the token embeddings themselves.

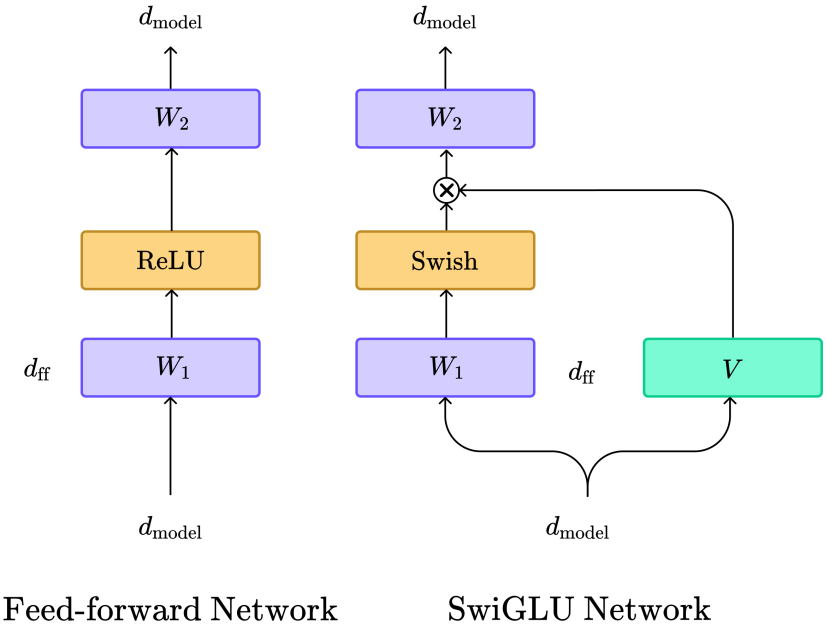


Figure 2.6: Comparison of a regular feed-forward layer with a SwiGLU layer.

SwiGLU Networks

Traditional feed-forward networks are replaced with SwiGLU networks, gated linear units (GLUs, Dauphin et al. (2017)) with a swish activation (Ramachandran et al., 2017). Essentially, SwiGLU modifies the feed-forward network by introducing a gating mechanism to control the information flow. This gating is achieved by adding a linear transformation of the input Vx and performing element-wise multiplication with the output of the first feed-forward layer activated by the Swish function.

A forward pass through the SwiGLU network is summarised by:

$$\text{SwiGLU}(x) = \text{Swish}(xW_1) \odot (Vx) \tag{2.32}$$

$$\text{Swish}(x) = x\sigma(x) \tag{2.33}$$

RMS Norm

The vanilla Transformer architecture uses layer normalisation (LayerNorm) *after* attention and feed-forward layers, standardising the inputs across the feature dimension and applying

an affine transformation:

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta \tag{2.34}$$

Where μ and σ are the mean and standard deviation of the input across the feature dimension and γ and β are learned parameters called the gain and bias. Instead, Llama applies normalisation *before* attention and feed-forward layers. Additionally, it replaces LayerNorm by the root-mean-square norm (RMSNorm, Zhang and Sennrich (2019)), a more efficient normalisation procedure removing the mean-centering step and omitting the bias parameter. The input is now scaled by its root-mean-square:

$$y = \frac{x}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2 + \epsilon}} \cdot \gamma \tag{2.35}$$

RMSNorm has been shown to perform similarly to LayerNorm while being significantly faster.

Grouped-Query Attention

Standard multi-head decoder blocks employ the same number of heads for keys, queries and values. While this approach is highly performant, it also incurs a high computational cost. Grouped-Query Attention trades off performance for efficiency by dividing the N_q query heads into G groups, where each group shares a single key and value head and $1 < G < N_q$. This significantly reduces the size of the key-value cache at inference time, speeding up generation while retaining most of the performance of multi-head attention.

2.2.3 Limitations of Transformer-Based LLMs

As discussed previously, Transformers leverage attention to store key-value associations and retrieve them by computing pairwise similarity between queries (i.e. search signals) and keys (i.e. contexts). However, these associations are only conditioned on the *current* context window and attention has quadratic scaling with respect to the context length (the size of the context window). This implies that the context window is finite and prohibitively costly to expand to long sequences. Some tasks like language modeling or video understanding may require extremely large context windows, limiting the applicability of Transformers in these tasks and encouraging research on more efficient alternatives.

2.2.4 Optimising the Attention Mechanism

Approaches to improving attention efficiency can be broadly categorised into hardware-aware optimisations, algorithmic approximations using sparsity, and mathematical reformulations.

A prime example of hardware-centric optimisation is FlashAttention (Dao et al., 2022, Dao, 2023). By fusing all attention operations into a single CUDA kernel, it minimises the number of slow read/write operations to high-bandwidth memory. Techniques such as tiling and online softmax calculation ensure that memory usage scales linearly with sequence length, resulting in significant speedups for exact attention without any approximation.

A second, algorithmic approach is to introduce sparsity into the attention matrix, limiting the number of tokens to which any given token can attend. Early work introduced fixed attention patterns, such as combining a local window with a strided or dilated pattern (Child et al., 2019). A more common variant is the sliding-window attention used in models like LongFormer (Beltagy et al., 2020), where each token attends to a fixed-size window of neighbouring tokens. While computationally efficient, these fixed patterns risk missing important long-range dependencies. Nonetheless, the approach remains highly relevant, with modern models like Mistral 7B (Jiang et al., 2023) successfully employing large sliding windows($w = 4096$) covering half of the total context length.

A third strategy aims to eliminate the quadratic bottleneck entirely by reformulating the attention equation. Linear Attention methods (Katharopoulos et al., 2020) approximate the softmax kernel, $\text{sim}(Q_i, K_j) = \exp\left(\frac{Q_i^T K_j}{\sqrt{d}}\right)$, with a feature map $\phi(\cdot)$ such that $\text{sim}(Q_i, K_j) \approx \phi(Q_i)^T \phi(K_j)$. This decomposition allows for a reordering of operations:

$$A_i = \frac{\phi(Q_i)^T \sum_{j=1}^{N} (\phi(K_j) V_j^T)}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)} \tag{2.36}$$

Crucially, the summations over all keys and values can be computed once as a state and reused for every query, reducing the overall complexity to linear. Subsequent research has focused on developing more powerful kernel functions. For example, Arora et al. (2023) used kernels based on Taylor expansions of the exponential function, while Aksenov et al. (2024) introduced learnable parameters into the kernel. Other work has sought to combine these ideas, for instance by unifying sparse patterns with kernel feature maps to improve performance on vision tasks (Chen et al., 2021).

Despite these advances, each approach carries trade-offs. The performance of linearised Transformers often lags behind that of standard Transformers on complex language tasks,

while sparse methods can suffer from suboptimal hardware utilisation due to the difficulty of creating optimised kernels for irregular memory access patterns. These persistent challenges motivate the exploration of entirely different paradigms. One of them aims to extend the context window *implicitely* by enabling LLMs to retrieve and exploit out-of-context information.

2.3 Titans: Learning to Memorize at Test Time

Contrarily to efficient attentions discussed in the previous section, the Titans model (Behrouz et al., 2024) attempts to reduce the cost of Transformers by scaling their memory capabilities at test time. Instead of explicitely extending the context window or modifying the attention mechanism, the authors introduce neural memory modules tasked to store and retrieve information at inference time. Historical (potentially out-of-context) information is then blended with attention representations, implicitely extending the effective context window. These memory modules have a recurrent update rule (subsection 2.1.1) and use gating mechanisms (subsection 2.1.2) to control the information flow. As such they are reminiscent of RNNs and LSTMs and have linear complexity in time, making them a cheaper alternative to attention.

Motivation

As discussed earlier, attention acts as an associative memory mechanism over a fixed context window. Consequently, it can be thought of as **short-term** memory. To understand the limitations of attention-based models, consider processing a stream of tokens, which eventually saturates the context window. This implies that the earlier tokens in the sequence will no longer be processed by attention, and their contextual information will not be exploited by tokens within the context window.

To circumvent this issue, Titans incorporate **task-related** (referred to as "meta" memory) and **long-term** memory.

Meta-Memory

In addition to short and long-term memory, the authors define meta-memory, a set of task-related, input-independent parameters. In the context of an agentic task such as playing a game, meta-memory could be seen as a compressed representation of relevant information

about the game such as the rules, how to play the game and general knowledge that does not depend on the current context.

Meta-memory is implemented as a vector of N_p learnable parameters $P = \begin{bmatrix} p_1 & p_2 & \dots & p_{N_p} \end{bmatrix}$ that are prepended to the inputs:

$$\tilde{x} = \text{Concat}\left(\begin{bmatrix} p_1 & p_2 & \dots & p_{N_p} \end{bmatrix}; x\right) \tag{2.37}$$

This kind of input-augmentation can be seen as a form of learned prompt-tuning that remains static at test-time.

Long-Term Memory

The core contribution of this work is the introduction of neural memory modules[2], described as *online meta-models* that learn how to memorise and forget data at **test time**. These memory networks are designed to store information in their weight matrix and minimise an associative retrieval loss $\ell(\cdot;\cdot)$. They are defined as simple MLPs with residual connections and use two computational paths to store and retrieve memories, both drawing inspiration from attention.

To store memories, the memory module uses linear layers to project inputs into queries, keys and values. The keys and values are then computed to measure the retrieval loss:

$$\ell(\mathcal{M}_{t-1}; x_t) = ||\mathcal{M}_{t-1}(k_t) - v_t||_2^2 \tag{2.38}$$

Where $\mathcal{M}$ refers to the parameters of the memory network. Optimising this loss enables the network to learn to memorise key-value mappings at test-time.

When approaching the problem of online memorisation, it is important to consider the concept of **surprise**. Consider processing a "surprising" input, in the sense that it is not yet stored in memory, or was previously associated with different information. From a human's perspective, a surprising event might be followed by some less surprising, but still relevant events (this event "catches our attention"). This implies that surprise has some notion of **momentum**.

In the context of memory modules, surprise is broken down into *momentary* and *past* surprise. Momentary surprise is measured as the gradient of the associative loss. On the other hand, past surprise is defined recursively as the scaled momentary surprise of the

[2]In the following sections, "memory network" refers to the MLP whereas "memory module" refers to the ensemble of the memory network, the learned gates and linear projections.

previous input, thus implementing momentum. The surprise update is implemented as:

$$S_t = \eta_t \underbrace{S_{t-1}}_{\text{past surprise}} - \theta_t \underbrace{\nabla \ell(\mathcal{M}_{t-1}; x_t)}_{\text{momentary surprise}} \tag{2.39}$$

Similarly to gated networks, memory modules use gates (here linear layers) to control the flow of information in an input dependent fashion. Here, η_t and θ_t scale the past surprise and the gradient of the loss respectively. These mechanisms are useful to avoid writing irrelevant data to memory. Indeed, an input might cause high associative loss because it is particularly noisy. While the gradient would be large, we may want to avoid writing this information to memory, which can be achieved by setting the input-dependent learning rate θ_t to zero. Similarly, an input sequence x_t might be judged unsurprising or unrelated to x_{t-1}, which would cause the adaptive momentum η_t to downscale the past surprise.

Finally, the parameters of the model itself are updated using the following equation:

$$\mathcal{M}_t = (1 - \alpha_t)\mathcal{M}_{t-1} + S_t \tag{2.40}$$

Where $\alpha_t \in [0, 1]$ is the forget gate responsible to erase information that is not needed anymore, improving the management of the memory's limited capacity and limiting memory overflow. Note that the memory is continuously updated as the module processes new inputs, including at test time. Therefore, the training of a Titans model is reminiscent of meta-learning approaches where updating the memory constitutes the inner-loop and other parameters are updated in the outer-loop.

Retrieving the memories associated with a query $q_t = x_t W_q$ is achieved by using a forward pass without weight update (referred to as *retrieval*):

$$y_t = \mathcal{M}^*(q_t) \tag{2.41}$$

Here, $\star$ denotes the update-free nature of the forward pass through the memory module. A visual representation of the memory module is available in Appendix A.7.

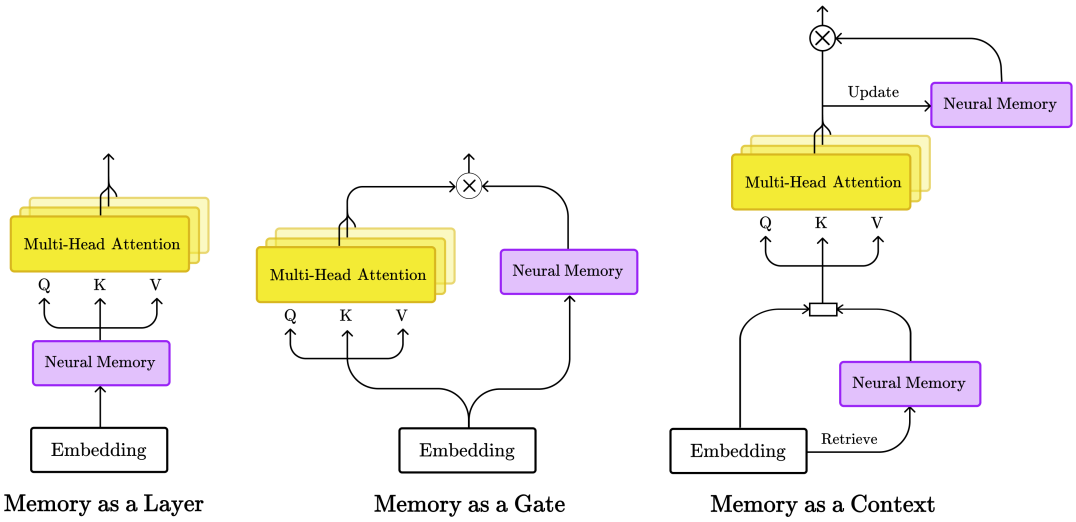


Figure 2.7: Titans architectures as described in Behrouz et al. (2024). The $\otimes$ symbol represents a non-linear gate and $\square$ the concatenation operation. "Embeddings" refers to the input sequence extended by the meta memory vector $\tilde{x}_t$, "retrieve" and "update" refer to a forward pass through the memory network with and without update respectively.

The authors introduce three novel architectures integrating neural memory networks:

1. Memory as a Layer (MAL): MAL is a traditional architecture stacking a recurrent model (the memory module) with attention. It simply adds meta-memory to an input to form $\tilde{x}_t$ and feeds it to the memory module before applying sliding-window attention (SW-Attn):

$$\tilde{x}_t = \text{Concat}\left(\begin{bmatrix} p_1 & p_2 & \dots & p_{N_p} \end{bmatrix}; x_t\right) \tag{2.42}$$

$$y_t = \mathcal{M}(\tilde{x}_t) \tag{2.43}$$

$$o_t = \text{SW-Attn}(y_t) \tag{2.44}$$

Here, the notation $\mathcal{M}(\tilde{x}_t)$ is overloaded to refer to the final output of the memory network after recursion and memory updates over all tokens in $\tilde{x}_t$.

2. Memory as a Gate (MAG): In contrast, MAG decouples the memory module from the sliding attention by introducing a non-linear gating. This enables the model to

trade-off short term memory (attention) with long-term memory (memory module):

$$\tilde{x}_t = \text{Concat}\left(\begin{bmatrix} p_1 & p_2 & \dots & p_{N_p} \end{bmatrix}; x\right) \tag{2.45}$$

$$y = \text{SW-Attn}^*(y) \tag{2.46}$$

$$o = y \otimes \mathcal{M}(\tilde{x}_t) \tag{2.47}$$

Where SW-Attn* is sliding-window attention with prefix. Specifically, the attention mask contains a fixed-sized window of tokens centered around the current token as well as the persistent memory vector as depicted in Appendix A.8.

3. Memory as a Context (MAC): Finally, MAC reuses the gated design of MAG, while integrating past information as a context for attention. At a given step t, the input sequence is projected into a query vector q_t used to perform an *update-free* retrieval as described in Equation 2.41. This produces an output vector h_t referred to as context. The context is concatenated to the original input sequence and passed through an attention layer.

$$h_t = \mathcal{M}^\star_{t-1}(q_t) \tag{2.48}$$

$$\tilde{x}_t = \text{Concat}\left(\begin{bmatrix} p_1 & p_2 & \dots & p_{N_p} \end{bmatrix}, h_t, x_t\right) \tag{2.49}$$

$$y_t = \text{Attn}(\tilde{x}_t) \tag{2.50}$$

The resulting attention vector y_t will be used to perform a memory update following Equation 2.40 and produce another context vector, which will be combined with attention outputs using a non-linear gate similarly to MAG:

$$o_t = y_t \otimes \mathcal{M}(y_t) \tag{2.51}$$

Limitations and Axes of Improvement

The Titans paper presents several limitations:

1. **Lack of architectural details:** This paper insists on the motivation behind neural memory networks and their implementation. However, specific architectural are omitted for the sake of simplicity, which limits reproducibility. Specifically, the authors

mention training models having between 170 and 760M parameters, however the way these parameters are distributed within the model remain unclear. Additionally, the authors do not mention whether or not a Titans model stacks multiple decoder blocks and memory modules like a general decoder-only architecture[3]. Other details such as the potential use of positional encodings are left out, even though such encodings could lead to duplicated information. Generally, it seems that the paper was written hastefuly, as many sentences have missing words and important implementation details are left out.

2. **Input-dependent retrieval:** The Titans retrieval mechanism relies on queries generated from tokens currently within the context window. This creates a critical limitation: retrieval is only effective for information related to tokens that either still exist in the context or have left a strong, persistent trace. To illustrate, consider a toy model with a context window of size 3 processing the sentence "Elisa bought some peanuts." Once the sequence is processed, the final context contains $\left[\text{bought, some, peanuts}\right]$, and the token "Elisa" is lost. If the model is subsequently prompted to recall who bought peanuts, it cannot form a query containing direct information about "Elisa" to probe the long-term memory. While one could argue that latent information about "Elisa" might persist in adjacent tokens via attention, this implicit mechanism is fragile at best. In summary, the quality of retrieval is fundamentally bottlenecked by the quality of queries, which are constrained by the contents of the short-term context window.

3. **Limited scaling:** As mentioned previously, the largest Titans model evaluated in the paper has 760M parameters. While Titans are not designed to be general-purpose LLMs, studying the scaling of memory-augmented architectures represents an important next step to evaluate the real-world usability of these models.

4. **Tabula rasa approach:** So far, Titans models are trained from scratch on language modeling tasks. A promising next step would be to investigate whether standard pre-trained LLMs can benefit from Titans layers, aligning with the transfer learning paradigm which has shown great success in natural language processing. This is the main goal of this thesis.

[3]A brief discussion with the authors revealed that Titans are indeed a stack of multiple decoder blocks, each containing a memory module, in contrast to our prior belief.

5. **Benchmark selection:** So far, Titans have been tested on standard language modeling, needle-in-the-haystack, timeseries forecasting and DNA modeling tasks. While these benchmark showcase the improved long-context memory capabilities of Titans, further work could focus on more complex and real-world tasks that require long-term memory as well as reasoning or planning. Indeed, from the current results alone, it remains unclear how improved memory impacts the reasoning and planing ability of Titans models.

6. **Lack of open-source code:** Despite promising a code release, the authors have not yet shared the code used for the implementation, training and evaluation of Titans models, further limiting the reproducibility of the experiments and adding to the general confusion about the architecture.

2.4 Adapting Pre-Trained Language Models

The immense scale of modern foundation models necessitates efficient adaptation techniques. Parameter-Efficient Fine-Tuning (PEFT, Xu et al. (2023)) is a family of methods designed to reduce the computational and memory costs of fine-tuning by updating only a small subset of the model's parameters. This allows for the cost-effective specialisation of large models for downstream tasks, often approaching the performance of full fine-tuning. Our work's methodology, which involves injecting a small, trainable module into a large, pre-trained model, is conceptually aligned with the PEFT paradigm. In addition, we want our LLM to constantly adapt to the evolution of the memory modules, which may be achieved by fine-tuning the LLM during the initial training phase of memory modules. To this end, we use Low-Rank Adaptation as a fine-tuning method.

2.4.1 Low-Rank Adaptation (LoRA)

A particularly prominent PEFT method is Low-Rank Adaptation (LoRA, Hu et al. (2021)). It is based on the intrinsic rank hypothesis, which posits that the weight update matrix learned during fine-tuning, ΔW, has a low intrinsic rank and can be effectively approximated by the product of two smaller matrices.

Specifically, for a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, the update is represented by a low-rank decomposition $\Delta W = W_B W_A$, where $W_A \in \mathbb{R}^{d\times r}$, $W_B \in \mathbb{R}^{r\times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 remains frozen, and only W_A and W_B are updated.

The forward pass is modified additively:

$$h = W_0 x + \Delta W x = W_0 x + (W_B W_A) x \tag{2.52}$$

This reduces the number of trainable parameters for the matrix from $d \times k$ to just $r(d+k)$. A critical component of LoRA is its initialisation strategy: W_A is typically initialised with a random Gaussian distribution, while W_B is initialised to zero. This ensures that ΔW is zero at the beginning of training, preserving the base model's initial state for improved stability. We draw inspiration from this method in our proposed architecture.

Chapter 3

Related Work

The work presented in this thesis sits at the intersection of two major research areas: memory-augmented Transformers and methods for adapting pre-trained models. This chapter situates the contribution of this thesis within the existing literature. We start by introducing memory-augmentation as a distinct paradigm for equiping models with long-term memory. This is generally achieved by training memory-centric architectures from scratch, or integrating memory coprocessors in frozen, pre-trained architectures. We then discuss concurrent architectures focused on long-term sequence processing that move beyond attention.

3.1 Memory-Augmented Transformers

Beyond optimising or replacing attention, a third paradigm for extending the effective context of Transformers draws inspiration from cognitive neuroscience. The theory of complementary learning systems posits that the brain uses distinct systems for rapid learning of specific events (the hippocampus) and slow learning of general structure (the neocortex) (McClelland et al., 1995). Analogously, Memory-Augmented Neural Networks (MANNs) decouple computation from memory by augmenting a base model (like the neocortex) with an external memory module (like the hippocampus). We refer to this module as a Memory Coprocessor (MCP).

Current memory augmentation methods for LLMs fall into two main categories: those that are trained from scratch to leverage memory, and those that inject memory modules into already pre-trained, frozen models.

3.1.1 Models Trained with Integrated Memory

The first approach involves training a language model from the ground up with a bespoke memory architecture, allowing the base model and the MCP to co-adapt fully.

A notable example is **Titans** (Behrouz et al., 2024), the primary inspiration for this thesis. In Titans, the MCP is an MLP with residual connections whose weights are updated at test time using a surprise-based online learning rule. This mechanism uses learnable gates, reminiscent of LSTMs, to control the flow of information into and out of the MCP's weights, which function as the long-term store.

Similarly, **LM2** (Kang et al., 2025) is a Llama-3-based model trained from scratch with an integrated memory system. It differs from Titans in two key ways: information is stored in explicit memory bank matrices rather than the weights of a network, and these banks are updated via a gating system without a surprise metric. The retrieved memory is then combined with the standard activations using cross-attention.

Compared to these methods, our approach attempts to obtain similar memory capabilities without training a large model from scratch, rather, we integrate MCPs in a pre-trained model. Furthermore, Titans and LM2 are not trained to be general-purpose LLMs, whereas our approach aims to preserve the vast pre-training knowledge of commercial models while improving their memory.

3.1.2 Augmenting Pre-trained Models

The second, more practical approach involves augmenting existing, pre-trained LLMs with an MCP, typically by freezing the base model's weights and training only the memory components.

LongMem (Wang et al., 2023) exemplifies this approach. It attaches an MCP to a frozen LLM, which consists of two parts: memory banks (a key-value cache) and a residual network to process retrieved information. The MCP operates only on the final layer of the LLM, using a retrieval score to pull relevant key-value pairs from the memory banks before the final language modeling head.

Following this work, **CAMELoT** (He et al., 2024) introduces a more sophisticated, training-free method. Instead of storing raw key-value pairs, CAMELoT's memory slots maintain consolidated representations of past tokens, akin to dynamic conceptual summaries. Each slot can be viewed as approximating a distribution over the key-value manifold of past information. A token's key-value pair is merged into the most similar memory

slot if their similarity exceeds a predefined threshold, updating the slot's running average. This allows for more efficient compression of the past context. Crucially, this update mechanism is a heuristic applied exclusively at test time, requiring no gradient-based training.

While efficient in terms of computational resources, these approaches rely on heuristics (CAMELoT) or MCPs frozen at test-time. Additionally, the underlying LLM remains frozen, limiting the potential adaptation to the new memory signal. In contrast, our approach lets the LLM co-adapt to the MCPs at train-time, while the MCPs perform online-learning at test-time.

3.1.3 Architectural Alternatives to the Transformer

Recently, popular alternatives to attention-based models emerged. These attention-free architectures aim to capture long-range dependencies with linear complexity and broadly fit in two categories: state-space models and linear recurrent models.

3.1.4 State Space Models (SSMs)

State Space Models (SSMs), inspired by control theory, map input sequences to outputs through a latent hidden state governed by linear dynamics. While originally designed for continuous signals, recent advances have made SSMs highly competitive for language modeling. The **H3** model (Fu et al., 2022) introduced gating mechanisms to better handle discrete data, paving the way for **Mamba** (Gu and Dao, 2023). Mamba extends SSMs by making their parameters input-dependent, allowing the model to selectively retain or discard information. This selective mechanism enables context-aware behavior similar to attention, while preserving linear complexity.

3.1.5 Linear Recurrent Models (LRMs)

A second family, Linear Recurrent Models (LRMs), combines the efficiency of convolutional training with the constant-memory cost of recurrence at inference. The **RWKV** architecture (Peng et al., 2023) exemplifies this approach, using time-mixing and channel-mixing blocks with a decay mechanism to regulate the influence of past tokens. Similarly, the **Retentive Network (RetNet)** (Sun et al., 2023) unifies recurrence, attention, and convolution, offering a parallel formulation for training and a recurrent one for inference, thereby achieving both scalability and efficiency.

In summary, attention-free architectures demonstrate that long-term dependencies can be handled implicitly within recurrent hidden states. By contrast, the approach taken in this thesis retains the Transformer backbone and augments it with an explicit, modular memory system reminiscent of LSTMs.

3.2 Synthesis and Research Gap

The literature presents a clear trade-off. Models trained from scratch, such as Titans, demonstrate the power of deeply integrated, online-learnable memory but require immense computational resources. Conversely, methods that augment pre-trained models, like LongMem and CAMELoT, are more practical but often rely on simpler memory mechanisms, such as first-in first-out caches or training-free heuristics. Furthermore, maintaining the LLM frozen potentially prevents further performance gains, since the adaptation to the additional memory signal is limited.

The architectures presented in this thesis attempt to bridge this divide. We propose a novel methodology that synthesises the strengths of both paradigms: the practicality of augmenting pre-trained LLMs with the power of an online, test-time trainable memory module inspired by Titans. To further enhance integration, we jointly fine-tune the LLM alongside the memory modules, enabling the base model to adapt to the added memory signal. This allows us to investigate how a sophisticated learning mechanism can be efficiently injected into powerful foundation models, creating an architecture that combines pre-training, fine-tuning and rapid test-time adaptation.

Chapter 4

Neural Memory Augmentation for Pre-Trained LLMs

4.1 Methodology

This section outlines the main design choices and research directions pursued in the development of memory-augmented models. We structure the discussion into four key aspects:

1. **Base model selection:** Choosing an open-source LLM as the foundation for memory augmentation.
2. **LLM adaptation strategy:** Motivating the need for adapting the base model while training the MCP.
3. **Memory-augmented decoder block design:** Proposing a novel decoder block architecture that integrates Titans memory modules within a standard Llama decoder block, with attention to implementation details and initialisation.
4. **Memory injection strategy:** Presenting three approaches for arranging memory-augmented decoder blocks within the LLM decoder stack. The first two are tailored to limited computational resources and evaluated in this thesis, while the third, more resource-intensive strategy is left for future work.

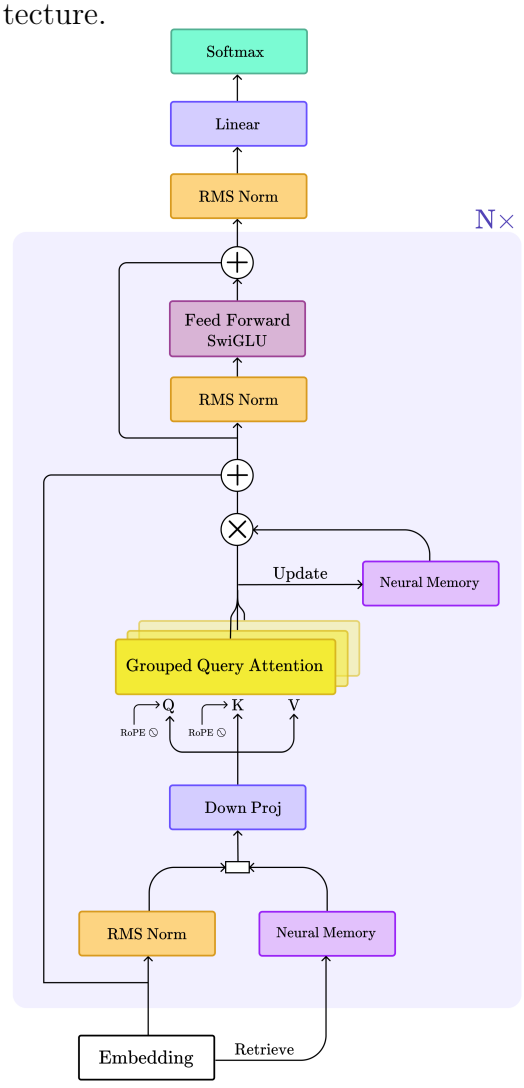
4.1.1 Base Model Selection

As a starting point, we consider the Llama-3 family of models (Grattafiori et al., 2024), focusing on the 1B variant to align with the computational resources available for this

project.

While the initial objective included studying scaling properties across the 8B and 70B models, these experiments were left to future work due to insufficient resources. Even though experiments are conducted with Llama models, the framework presented in this thesis remains model-agnostic, the only requirement being the access to model parameters. Therefore, other open-source model families such as Qwen (Yang et al., 2025) or Gemma (Team et al., 2025) could be considered in future work. This flexibility in the choice of framework could prove to be particularly relevant if practitioners aim to employ memory-augmented models in downstream applications or modalities dominated by a particular open-source model architecture (e.g., for a fixed computational budget, Llama models might have better coding performances but worse reasoning capabilities than their Gemma equivalent).

Figure 4.1: The MemoryLlama architecture.



4.1.2 LLM Adaptation Strategy

Our approach introduces randomly initialised, online-learning memory modules into a pre-trained LLM. Effective integration of these modules plausibly requires some degree of adaptation from the LLM itself to exploit and process the additional memory signal. A natural class of candidate solutions are PEFT methods which allow for cost-efficient fine-tuning. Among these, we adopt LoRA as our primary mechanism due to its wide adoption and the availability of optimised implementations in the `Transformers` library.

4.1.3 Memory-Augmented Decoder Blocks

In this section, we detail the architectural integration of the Titans memory module with the pre-trained Llama layers to create a MemoryLlama decoder block. Our design adheres to two guiding principles. First, to fully utilise the base model's pre-training, the integration should introduce minimal disruption at initialisation, an approach reminiscent of LoRA's initialisation. Second, the contribution of the memory module should be dynamically adjusted throughout training and inference.

To satisfy these principles, we adopt a gated architecture inspired by the MAC variant of Titans (Behrouz et al., 2024). This design allows a learnable gate to smoothly mix the memory output with the standard attention output. Additionally, this gate can be initialised to ignore memory, so that the addition of the memory module does not affect the base model before the start of training. In contrast, a sequential design like the MAL architecture where memory is processed before attention was found to be particularly unstable. Indeed, since the memory network is an MLP with non-linear activations, it cannot easily approximate an identity function at initialisation and would inevitably corrupt the finely-tuned inputs expected by the pre-trained attention layer.

The data flow within a MemoryLlama decoder block, depicted in Figure 4.1, can be understood in three stages:

1. Context Retrieval and Integration: The first stage enriches the raw input sequence x_t, with long-term context retrieved from memory. The input is used to form a query q_t, which retrieves a context vector h_t from the memory's state at the previous timestep $\mathcal{M}_{t-1}$. This context is then concatenated with the RMS-normalized input, $\text{RMSNorm}(x_t)$:

$$q_t = x_t W_q \tag{4.1}$$

$$h_t = \mathcal{M}^{\star}_{t-1}(q_t) \tag{4.2}$$

$$\tilde{x}_t = \text{Concat}\left(\text{RMSNorm}(x_t), h_t\right) \tag{4.3}$$

Since concatenation doubles the sequence length, a linear projection layer, W_{DP}, maps $\tilde{x}_t$ back to the original sequence dimension (DP stands for "down projection"). This creates the final memory-augmented input x_{DP}, which now contains contextual information while matching the input shape expected by the Llama attention layer.

$$x_{\text{DP}} = \tilde{x}_t W_{\text{DP}} \tag{4.4}$$

2. Attention and Memory Update: The memory-augmented input x_{DP} is processed by the pre-trained Llama attention layer to produce the attention output a_t. This output serves two purposes: it acts as the primary source of short-term context, and it provides the "surprise" signal needed to update the memory module from state $\mathcal{M}_{t-1}$ to $\mathcal{M}_t$ according to the update rule in Equation 2.40. A new long-term context vector m_t, is then retrieved from this updated memory state.

$$a_t = \text{Attn}(x_{\text{DP}}) \tag{4.5}$$
$$\mathcal{M}_t = \text{Update}(\mathcal{M}_{t-1}; a_t) \tag{4.6}$$
$$m_t = \mathcal{M}_t^\star(a_t) \tag{4.7}$$

3. Gated Output Combination: The final stage adaptively combines the short-term context from attention (a_t) with the long-term context from memory (m_t). As the specific gating mechanism was not detailed in the Titans paper, we implement a convex combination controlled by a learnable, per-token gate g_t. The gate values are predicted by passing the attention output through a linear layer W_g with bias b_g and applying a sigmoid activation. The final block output o_t is formed by adding a residual connection from x_{DP} to the gated output y_t.

$$g_t = \sigma(a_t W_g + b_g) \tag{4.8}$$
$$y_t = (1 - g_t) \odot a_t + g_t \odot m_t \tag{4.9}$$
$$o_t = y_t + x_{\text{DP}} \tag{4.10}$$

The output o_t is then passed to the rest of the standard Llama block, including an RMSNorm layer, a SwiGLU network and a residual connection, completing the MemoryLlama block.

Initialisation

As discussed previously, we aim to initialise our MemoryLlama block to behave identically to the original layer to improve the stability of finetuning. To do so, we need to ensure $x_{\text{DP}} = \text{RMSNorm}(x_t)$ and $y_t = a_t$.

First, we define W_{DP} as the concatenation of two matrices $W_x \in \mathbb{R}^{d \times d}$ and $W_h \in \mathbb{R}^{d \times d}$ such that $W_{\text{DP}} = [W_x, W_h]$. This allows us to write the output as $x_{\text{DP}} = W_x \cdot \text{RMSNorm}(x_t) + W_h \cdot h_t$. To achieve the identity mapping for $\text{RMSNorm}(x_t)$, we simply initialise W_x to the identity matrix $\mathbf{I} \in \mathbb{R}^{d \times d}$ and W_h to the zero matrix $\mathbf{0} \in \mathbb{R}^{d \times d}$.

Similarly, we set W_g to the zero matrix $\mathbf{0} \in \mathbb{R}^{d \times d}$ and b_g to a relatively large negative value, for instance -6. Thus, we obtain $g_t = \sigma(a_t \mathbf{0} + -6) \approx 2.5 \times 10^{-3}$. This is sufficient to recover:

$$y_t = (1 - g_t) \odot a_t + g_t \odot m_t \approx a_t \tag{4.11}$$

4.1.4 Memory Injection Strategy

In this section, we focus on the placement of the memory-augmented block within the stack of decoder blocks of the pre-trained LLM. Two main strategies are considered within this thesis, namely **embedding-level memory augmentation** (ELMA) and **attention-level memory augmentation** (ALMA).

In ELMA, the neural memory module processes the input embeddings directly, adding long-past contextual information. All the subsequent decoder blocks then benefit from this memory-augmented representation. This design could be seen as a generalisation of MAL, where the sliding window attention layer is replaced by a stack of pre-trained decoder blocks.

In contrast, ALMA integrates the memory module towards the end of the LLM's decoder stack, processing attention vectors instead of raw embeddings. In theory, this approach enables the memory module to benefit from the contextualised representation of the attention outputs, anchoring the stored information within context by the same occasion. Here, the memory module is injected within the second-to-last decoder block, so that the memory-augmented attention outputs can be processed by the last decoder block before predicting the output.

These designs, represented in Figure 4.2, trade off the number of decoder blocks processing memory-augmented embeddings (maximised in ELMA) for additional context-

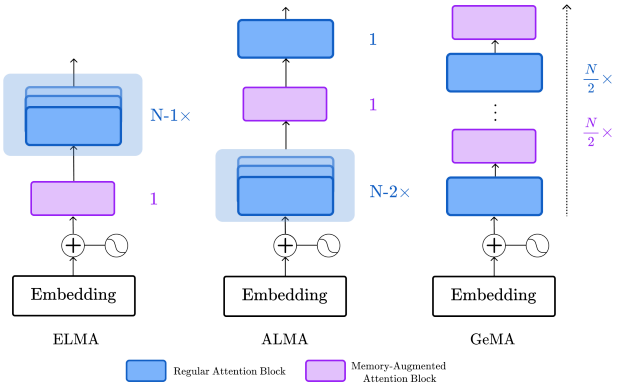


Figure 4.2: ELMA (left), ALMA (centre) and GeMA (right). N denotes the total number of decoder blocks in the LLM.

awareness in the information stored by the memory module (optimised by ALMA).

Finally, we consider a hybrid design which uses multiple memory modules injected at regular intervals, for instance once every two decoder blocks. We refer to this architecture as **Generalised Memory Augmentation** (GeMA), a trade-off between computational efficiency and performance. As explained in the following section, this design was left for future work in reason of the limited computational budget and the overhead created by memory modules.

4.2 Experimental Framework

4.2.1 Compute Environment and Hyperparameters

All experiments were conducted on a single NVIDIA GH100 GPU (AARCH64) using `bf16` mixed precision. The base model was Llama-3.2-1B, augmented with LoRA adapters of rank 16 applied to all attention and MLP modules. Memory modules were implemented as two-layer MLPs with residual connections and an expansion factor of two (input/output size 2048, hidden size 4096). Including its gating layers and q, k, v projections, each memory module comprised approximately 46 million parameters.

Training employed the `AdamW-8bit` optimiser with a constant learning rate of 1×10^{-3} and a weight decay of 0.1. The physical batch size was set to 4 due to memory constraints, but gradient accumulation over 4 steps yielded an effective batch size of 16, improving training stability.

Experiments were configured using Hydra (Yadan, 2019). Hyperparameters for ALMA and ELMA experiments are listed in Appendix B.1. The software environment was managed with UV, with dependencies specified in a `requirements.txt` file. All experiments conducted during this project can be easily replicated via `Make` commands, as detailed in the repository's `README`.

Computational Constraints and Experimental Scope

A primary consideration shaping the experimental design was the computational overhead of the Titans memory update rule (Equation 2.40). The rule's sequential nature necessitates an iterative update for each token in the sequence (2048 tokens in this study) on every forward pass. This iterative process, while parallelizable across the batch dimension (itself limited by the available memory), creates a serious computational bottleneck.

We investigated the parallel update rule proposed by Behrouz et al. (2024) as an alternative. However, this method's substantially higher memory requirements proved intractable given the available single-GPU setup. Consequently, to maintain feasibility, the experimental scope was reduced by limiting the number of memory modules and constraining the total number of training steps. We therefore define both ELMA and ALMA as MemoryLlama architectures with two memory modules injected in the first and second or the last and second-to-last decoder blocks respectively. Possible solutions to these issues are listed in section 6.2.

4.2.2 Memory Training

Objective and Dataset. To train the memory-augmented components, we adopt a causal language modeling objective (i.e., next-token prediction). The model is fine-tuned on the `fineweb-edu` dataset (Lozhkov et al., 2024), a large corpus of high-quality educational web text, following the training procedure of the Titans paper (Behrouz et al., 2024). While direct training on downstream memory tasks like BabiLong was explored, this approach proved unstable as the model quickly learned a trivial strategy of predicting the `<eos>`[1] token to minimise the causal loss. Training on a general language corpus ensures that the memory module learns to store and retrieve information that is broadly useful for language understanding, rather than overfitting to a narrow task.

[1]The "end of sentence" token ends the generation process prematuraly.

Table 4.1: BabiLong Tasks

Task	Name	Facts per task	Supporting facts per task
qa1	single supporting fact	2 - 10	1
qa2	two supporting facts	2 - 68	2
qa3	three supporting facts	4 - 32	3
qa4	two arg relations	2	1
qa5	three arg relations	2 - 126	1
qa6	yes-no questions	2 - 26	1
qa7	counting	2 - 52	1-10
qa8	lists-sets	2 - 50	1- 8
qa9	simple negation	2 - 10	1
qa10	indefinite knowledge	2 - 10	1

Training Process and Parameters. Training runs were conducted for 12,000 to 20,000 steps, with each step processing a sequence of 2048 tokens (a total of 24–41M tokens). Runs exhibiting early signs of overfitting or catastrophic forgetting were terminated. During training, three sets of parameters are updated concurrently:

1. The memory network's weights are updated at each step according to the online rule in Equation 2.40.

2. The memory gate parameters (η, θ, α, W_g, b_g and W_{DP}) are updated via standard backpropagation, optimising the causal language modeling loss.

3. The parameters of the underlying Llama-3.2-1B model are adapted using LoRA with gradient descent to allow the base model to adapt to the newly injected memory modules.

The resulting MemoryLlama models have a total of 1.53B parameters, of which the base 1.3B parameters are frozen. The trainable components consist of 92M (7%) parameters for the memory modules and 61M (4.5%) for the LoRA adapters.

Training Dynamics and Monitoring. To ensure the model learns to use its memory module as intended, we monitor several key statistics throughout the training process:

- **Gate Activation (g_t):** We track the distribution (mean, std, min, max) of the output gate activations. A healthy training run should show the average activation

increasing, with max values approaching 1 and min values approaching 0, indicating the model is learning to dynamically arbitrate between short-term (attention) and long-term (memory) information.

- **Gate Bias (b_g):** We monitor the bias of the gate's linear projection. An increase from its initial value of -6 provides a simple indicator that the memory pathway is being utilised more frequently.

- **Memory Projection Norm ($||W_h||_2$):** The weights in W_{DP} corresponding to the memory context (W_h) are zero-initialised. We track the L2 norm of this sub-matrix as a sanity check to confirm that the model is learning to incorporate information from the memory context vector h_t.

- **Relative Memory Contribution:** We measure the relative influence of the memory term versus the attention term in the final output combination (Equation 4.9). We compute both a global contribution per step and an average per-token contribution across the sequence length L:

$$\frac{\|m_t \odot g_t\|_2}{\|(1 - g_t) \odot a_t\|_2 + \epsilon} \qquad \text{(Global Contribution)} \tag{4.12}$$

$$\frac{1}{L} \sum_{i=1}^{L} \frac{|m_{t,i} \cdot g_{t,i}|}{|(1 - g_{t,i}) \cdot a_{t,i}| + \epsilon} \qquad \text{(Avg. Per-Token Contribution)} \tag{4.13}$$

- **Causal Loss:** Finally, we monitor the standard cross-entropy loss for next-token prediction. This allows us to track the model's core language modeling capability and verify that the addition of memory does not degrade its general performance.

4.2.3 Evaluation

Benchmark: BabiLong. Following the experimental framework of Behrouz et al. (2024), we evaluate the fine-tuned models on the Babilong benchmark (Kuratov et al., 2024). Babilong comprises 10 procedurally generated tasks designed to test elementary reasoning and memory recall. Each sample consists of a series of facts describing interactions between characters and objects (e.g., "Mary travelled to the office"), followed by a question (e.g., "Where is Mary?"). The core challenge lies in identifying the relevant facts, which are embedded within distractor sentences, and using them to answer the question. Task

difficulty is modulated by factors such as the number of facts, the length of the distractor text, and the complexity of the question. A list of the tasks is provided in Table 4.1.

Evaluation Protocol. To ensure a fair and reproducible evaluation, we adhere to a strict protocol for each of the 100 test samples per task (1000 total samples):

1. The memory network's weights ($\mathcal{M}$) and momentum vectors (S) are reset to a zero state to prevent information leakage from previous samples.
2. The model processes the test sample with a batch size of 1 to avoid information leakage or conflicting memories from other samples in the batch.
3. The model generates up to 25 tokens autoregressively with a temperature of 0, so that the outputs are deterministic.
4. The generation is marked as correct if the ground-truth answer (typically a single word) is present in the output.

Our primary metric is the per-task accuracy, reported as the percentage of correct answers.

Chapter 5

Results and Analysis

In this section, we present the empirical results of our investigation into the MemoryLlama architecture. We first analyse the training dynamics of the ALMA and ELMA models, followed by their performance on the Babilong benchmark. We conclude with a series of ablation studies designed to probe the behaviour of the memory components.

A central finding of this study is that both the ALMA and ELMA models, under the current training configuration, exhibit catastrophic forgetting after an initial phase of stable learning. This phenomenon occurred after approximately 2970 batches for ALMA and 980 batches for ELMA. This section therefore analyses both the promising initial learning dynamics and the subsequent onset of this instability. The potential causes of this behaviour and solutions to mitigate it are discussed in section 6.2. This chapter focuses on empirically characterising the phenomenon itself.

For visual clarity, the plots presented in the following sections and the appendix use a Gaussian smoothing filter with a standard deviation of 3, while the true lines are shaded.

5.1 Training Dynamics and the Onset of Instability

We compare and analyse the train-time metrics for the ALMA and ELMA models. To provide a complete picture of the training process, we first present the full training curves that capture the entire learning trajectory, including the point of collapse. Subsequently, we provide a more detailed analysis of the initial, stable learning phase that preceded this instability.

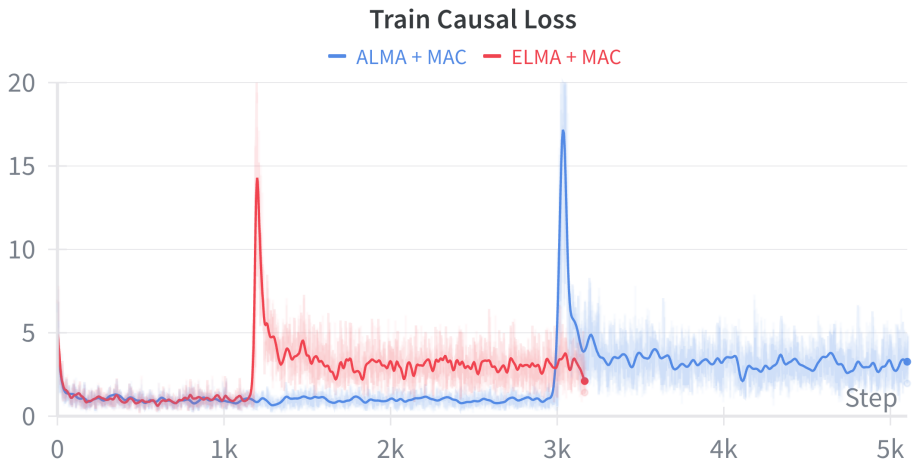


Figure 5.1: Causal loss of ALMA and ELMA during training, showing the onset of instability.

Characterising Catastrophic Forgetting. First, we analyse the training losses over the full duration of the runs. As depicted in Figure 5.1, both models exhibit a period of stable convergence, with the causal loss decreasing towards a value of approximately 1.0. This is followed by a sudden spike where the loss rapidly exceeds 20, before settling at a new, much higher baseline. This sharp degradation in performance indicates a catastrophic forgetting event, where the model abruptly deviates from its effective pre-trained state.

This instability manifests clearly in the memory-related metrics. We observe a polarization of the output gate (g_t), where the activations for individual tokens collapse towards either 0 or 1. This causes the mean gate activation to jump and the standard deviation to increase dramatically, while the memory contribution metrics fall to near-zero. This suggests the model adopts a dysfunctional strategy: for a small subset of tokens, it relies exclusively on the memory output (where $g_t = 1$), while for the majority, it relies exclusively on attention (where $g_t = 0$). Simultaneously, the L2 norm of the memory projection weights ($\|W_h\|_2$) explodes, rising from a stable plateau around 10 to over 40 for ALMA and over 20 for ELMA. This uncontrolled growth in the memory pathway parameters is a symptom of training instability. Complete training reports showing this collapse are available in Appendix A.1.1.

Analysis of the Stable Learning Phase. Focusing on the metrics prior to collapse reveals valuable insights into the initial learning dynamics. For the ALMA model, the gate

statistics suggest a healthy and nuanced adoption of the memory module. The minimum and maximum gate values quickly diverge towards 0 and 1 respectively, creating a wide dynamic range. The mean activation steadily increases to 0.008, indicating that the model is learning to put more emphasis on the memory pathway. Interestingly, we observe that the memory module injected in layer 14 consistently utilizes memory more than the one in layer 15. We hypothesise that the model learns a conservative gating strategy for the final layer due to its more direct impact on the output logits.

In contrast, the ELMA model (with memory in layers 1 and 2) struggles to meaningfully engage its memory modules. The maximum gate activation barely exceeds 0.005 before collapse, and the average memory contribution is an order of magnitude lower than ALMA's. This behaviour is likely attributable to vanishing gradients; modules placed in the earliest layers of a deep network have a less direct impact on the final loss, receive weaker learning signals, and therefore learn more slowly.

Given that the ALMA architecture demonstrated a more promising initial learning trajectory, we focus our subsequent ablation studies on it. Training reports from the stable phase for both models are available in Appendix A.1.2.

Hypotheses on the Cause of Instability. Based on our analysis, we hypothesise two primary factors contributing to the observed catastrophic forgetting. The first potential cause is the interaction between the frozen base model and the trainable LoRA adapters. During fine-tuning, if the LoRA updates cause the model's representations to stray too far from their pre-trained manifold, it can lead to a collapse in performance. To test this, we conduct an ablation study on ALMA without LoRA updates. The second factor is the aggressive learning rate. The use of a large, constant learning rate of 1×10^{-3} was a concession to the limited computational budget. The observed instability suggests that a more conservative approach, such as a lower learning rate, a cosine decay schedule, or differential learning rates for the MCP and LoRA components, is warranted for future work.

5.2 BabiLong Performance and Ablation Studies

We evaluate the performance of our memory-augmented models on the BabiLong benchmark. To provide a comprehensive analysis, we compare checkpoints of ALMA and ELMA taken from their stable training phase against two baselines: the raw Llama-3.2-1B model

and the same model fine-tuned on the `fineweb-edu` dataset for a comparable number of steps, note that this model didn't suffer catastrophic forgetting. We also include two ablations of ALMA to diagnose the effects of the memory module and the fine-tuning process. The results are presented in Table 5.1, while a visual representation is available in Appendix A.6.

Table 5.1: Performance comparison on BabiLong QA tasks. Best score per task is in bold. Checkpoints for ALMA and ELMA are taken from the stable training phase prior to catastrophic forgetting.

Method	**qa1**	**qa2**	**qa3**	**qa4**	**qa5**	**qa6**	**qa7**	**qa8**	**qa9**	**qa10**
Llama 3.2 1B (Baseline)	**55**	**29**	34	55	**76**	5	1	30	17	5
+ fine-tuning	55	27	**36**	**57**	75	6	4	**35**	13	4
ALMA (Ours)	44	16	35	41	72	**11**	**5**	26	7	7
- fine-tuning	46	25	27	47	72	7	2	25	16	**8**
- LoRA	37	20	18	41	67	4	3	31	11	4
ELMA (Ours)	47	24	30	50	61	8	1	22	**19**	**8**

Primary Finding: Performance Degradation Compared to Baselines. The primary result is that neither ALMA nor ELMA consistently outperform the fine-tuned Llama baseline. In most tasks, the introduction and subsequent training of the memory modules led to a notable degradation in performance (e.g., a drop from 55% to 44% on qa1 for ALMA). While the memory-augmented models do achieve the highest score on four of the ten tasks, these are exclusively tasks where baseline performance is already very low (under 20%). Furthermore, the margin of improvement in these cases is small (1-6 percentage points), making it difficult to conclude that this constitutes a meaningful improvement in memory capabilities[1].

Analysis of Ablation Results. The ablation studies provide further insight into the source of this performance degradation:

- **The Effect of Training:** The "ALMA - fine-tuning" model, which includes the zero-initialised memory modules without any training, performs slightly better than the trained ALMA model on several tasks (e.g., qa1, qa2, qa4). This suggests that

[1]A limitation of our study is the absence of confidence intervals. Given the experimental protocol requiring a temperature of 0, obtaining multiple samples would necessitate training the same model several times, which is infeasible with the available computational budget.

the initial phase of training is detrimental, as the randomly initialised modules and their subsequent unstable updates disrupt the base model's representations.

- **The Role of LoRA:** The "ALMA - LoRA" ablation was designed to test the hypothesis that adapting the base LLM's parameters is necessary for the effective integration of the memory modules. The experiment revealed a critical trade-off between performance and stability. On one hand, freezing the base LLM (removing LoRA) resulted in a significant performance decrease compared to the standard ALMA model on several tasks. This confirms that co-adapting the LLM is indeed a crucial component for achieving potential performance gains. On the other hand, this variant trained with remarkable stability, completing 28,000 samples without any sign of the catastrophic forgetting that affected the LoRA-enabled models (see Appendix A.5 for stable training metrics). These contrasting outcomes lead to a key insight: while LoRA-based adaptation appears to be a prerequisite for functionality, it is also the primary source of training instability. The instability therefore arises not from the memory module alone, but from the interaction between the online learning of the memory and the simultaneous fine-tuning of the base model. For a fair comparison, the performance reported in Table 5.1 is from an early checkpoint, consistent with the other models.

This diagnostic process suggests that the challenge lies not in the architectural concept, but in the difficulty of stably co-training the new memory modules alongside the sensitive parameters of a pre-trained LLM.

Chapter 6

Conclusion

6.1 Discussion and Principal Findings

This thesis introduced and evaluated MemoryLlama, a novel architecture that integrates a test-time trainable memory coprocessor, inspired by Titans, into a pre-trained Llama language model. Our primary research questions focused on the possibility for Titans memory modules to improve the recall accuracy on memory-intensive tasks, the impact of different injection strategies on the training dynamics and the need for LLM adaptation during training.

Our experiments revealed that this method of memory injection, in its current form, is highly sensitive to training dynamics. The principal finding of this work is the characterisation of a catastrophic forgetting phenomenon that prevented the models from completing a full training run. While the training metrics showed a promising initial phase where the models began to utilise the memory modules, this learning was not sufficient to yield a net performance benefit on the BabiLong benchmark before the onset of instability. The short training duration, constrained by this collapse, means our initial hypotheses remain unconfirmed.

The core insight from these results is that the interface between a large, static, pre-trained model and a small, dynamic, randomly-initialised online learning module is inherently volatile. The ablations suggest that this instability likely arises from a combination of aggressive LoRA updates pushing the model off its learned manifold and the choice of a large, constant learning rate. However, the absence of LoRA adapters, while improving training stability, degrades performances. This work, therefore, provides a valuable empirical demonstration of the challenges in co-training disparate components in a hybrid

model.

6.2 Future Work

The challenges identified in this thesis give rise to a clear and promising research agenda. We structure the next steps into three main categories: achieving training stability, analysing scalability and generalization, and pursuing architectural enhancements.

6.2.1 Stabilising the Training Paradigm

The most immediate priority is to mitigate the observed catastrophic forgetting. Several strategies are warranted:

- **Advanced Optimisation:** Future work should explore more conservative learning rate schedules, such as a cosine decay with a warm-up phase. Furthermore, employing differential learning rates, a smaller one for the sensitive LoRA adapters and a larger one for the from-scratch memory modules, could balance the learning dynamics.
- **Regularisation Techniques:** Implementing techniques like gradient clipping to prevent exploding gradients in the memory pathway, or exploring more stable PEFT methods such as Singular Value Fine-tuning (Sun et al., 2025), could enable longer, more stable training runs.

6.2.2 Scaling, Generalization, and Efficiency

Once a stable training regime is established, the core hypotheses of this thesis can be investigated further.

- **Systematic Scaling Analysis:** A key avenue is to conduct a systematic study of scaling laws. This involves applying the methodology to a range of model sizes, from smaller, more agile models (e.g., Qwen3-0.6B (Yang et al., 2025)) to larger ones (e.g., Llama-3.2 8B and 70B), to establish a scaling curve for memory augmentation. This would also involve exploring the impact of stacking a greater number of memory modules, as the two used in this study may not be sufficient to significantly alter the base model's capabilities.

- **Hardware-Aware Optimisation:** To make larger-scale experiments feasible, future work must address the computational bottlenecks of the memory update rule. Developing hardware-aware implementations using frameworks like Triton (Tillet et al., 2019) or JAX (Bradbury et al., 2018) to create optimised kernels for the memory update would drastically improve throughput, enabling the more rigorous and extensive experimentation required. Additionally, while incompatible with the hardware used in this thesis, libraries like LigerKernels (Hsu et al., 2025) offer optimised Llama-wrappers, significantly reducing the memory footprint while increasing the throughput of the base LLM.

6.2.3 Architectural Enhancements

Beyond stability, the architectural framework itself can be extended.

- **Memory-Register Tokens:** As discussed in section 2.3, a current limitation of Titans is its inability to query out-of-context tokens, limiting information retrieval to in-context tokens. Hence, a promising architectural evolution is the integration of dynamic memory-register tokens, inspired by Darcet et al. (2023). By adding a persistent `<|Memory|>` token to the input, the model could learn to aggregate a running summary of the context. This token's hidden state could then generate more informed queries for the long-term memory, overcoming the limitation of relying solely on the immediate context window.

- **Memory Interpretability:** A significant open question is what the model learns to store. Future work could apply interpretability techniques to analyse the contents of the memory module, providing insights into the emergent cognitive strategies of these hybrid models.

6.3 Outlook

This thesis presented a new paradigm for augmenting foundation models: the integration of a specialised, online-learning memory coprocessor within a pre-trained architecture, itself co-adapting with PEFT. While our initial experiments revealed significant stability challenges, they have charted a clear path for future research and provided valuable insights into the difficulties of creating such hybrid architectures.

From a broader perspective, this paradigm draws inspiration from the modular nature of cognition in the brain. In our analogy, the pre-trained LLM acts as a knowledge foundation (akin to the neocortex) while the memory module acts as a specialised system for rapid, online learning (akin to the hippocampus). While this thesis focused on memory, this "modular cognition" approach could be extended to other domains. For instance, one could imagine integrating highly specialised reasoning modules, such as the recent Hierarchical Reasoning Model (HRM, Wang et al. (2025)) into an LLM to synergise the generalist capabilities of language models with the specialist power of dedicated reasoning engines.

Ultimately, the path towards more capable and biologically-plausible artificial intelligence may not lie in creating ever-larger monolithic models, but in learning how to effectively compose and co-train a hierarchy of specialised, modular networks.

Bibliography

Yaroslav Aksenov, Nikita Balagansky, Sofia Maria Lo Cicero Vaina, Boris Shaposhnikov, Alexey Gorbatovski, and Daniil Gavrilov. Linear transformers with learnable kernel functions are better in-context models. *arXiv preprint arXiv:2402.10644*, 2024.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

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

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

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

Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 17413–17426. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/9185f3ec501c674c7c788464a36e7fb3-Paper.pdf.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers, 2023.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

Daniel Y Fu, Tri Dao, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

Zexue He, Leonid Karlinsky, Donghyun Kim, Julian McAuley, Dmitry Krotov, and Rogerio Feris. Camelot: Towards large language models with training-free consolidated associative memory. *arXiv preprint arXiv:2402.13449*, 2024.

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

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, Yanning Chen, and Zhipeng Wang. Liger-kernel: Efficient triton kernels for LLM training. In *Championing Open-source DEvelopment in ML Workshop @ ICML25*, 2025. URL https://openreview.net/forum?id=36SjAIT42G.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. arxiv 2021. *arXiv preprint arXiv:2106.09685*, 10, 2021.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

Jikun Kang, Wenqi Wu, Filippos Christianos, Alex J Chan, Fraser Greenlee, George Thomas, Marvin Purtorab, and Andy Toulis. Lm2: Large memory models. *arXiv preprint arXiv:2502.06049*, 2025.

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

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. In search of needles in a 10m haystack: Recurrent memory finds what llms miss, 2024.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: enhanced transformer with rotary position embedding. arxiv. *arXiv preprint arXiv:2104.09864*, 2021.

Qi Sun, Edoardo Cetin, and Yujin Tang. Transformer-squared: Self-adaptive llms. *arXiv preprint arXiv:2501.06252*, 2025.

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

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

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

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543, 2023.

Lilian Weng. Attention? attention! *lilianweng.github.io*, 2018. URL https://lilianweng.github.io/posts/2018-06-24-attention/.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *arXiv preprint arXiv:2312.12148*, 2023.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Appendix A

Additional Illustrations

A.1 Training Statistics Reports

A.1.1 Training Statistics with Catastrophic Forgetting

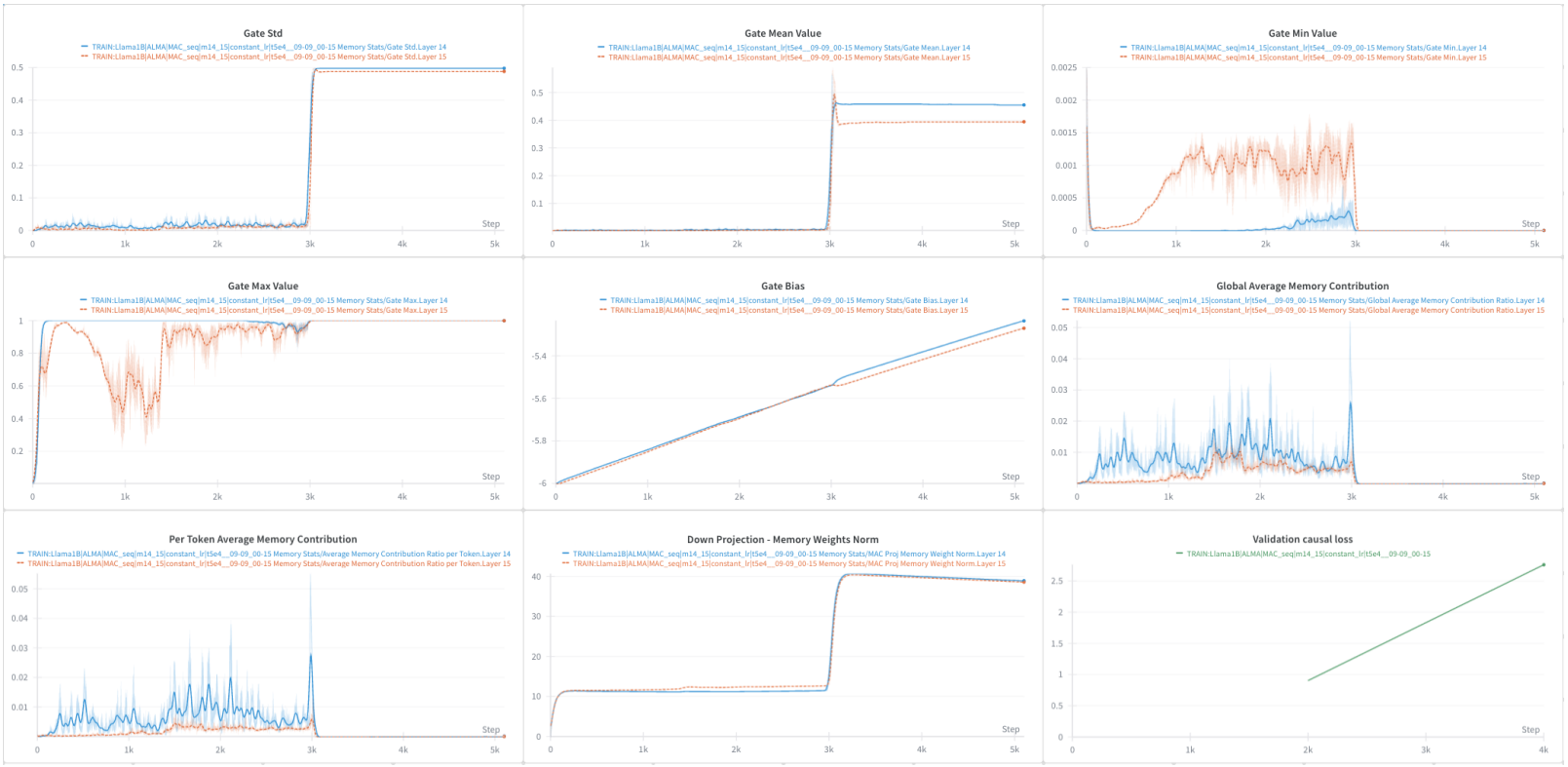


Figure A.1: Training Statistics for ALMA with Catastrophic Forgetting.

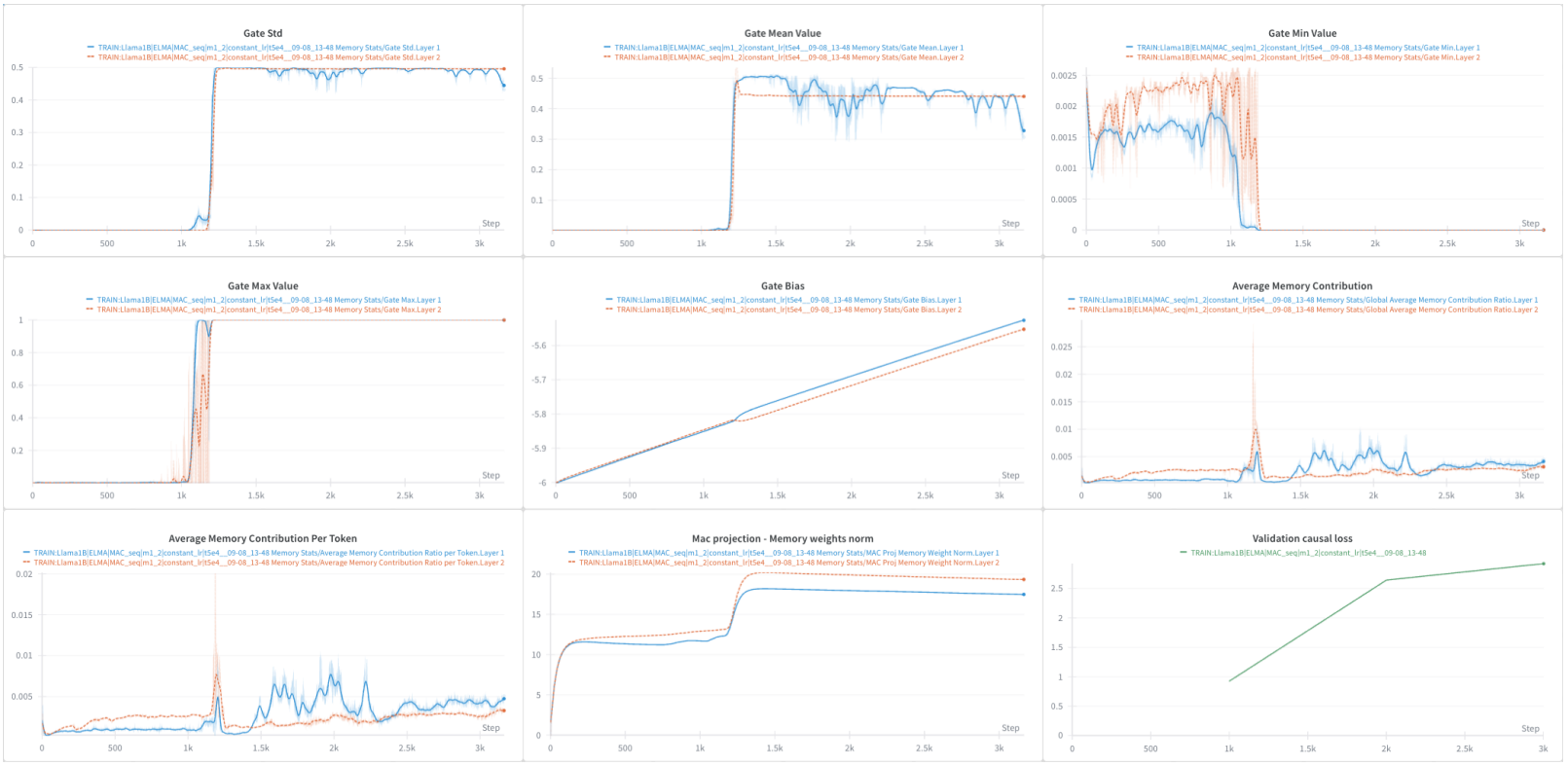


Figure A.2: Training Statistics for ELMA with Catastrophic Forgetting.

A.1.2 Training Statistics Prior to Catastrophic Forgetting

Figure A.3: Training Statistics for ALMA prior to Catastrophic Forgetting.

Figure A.4: Training Statistics for ELMA prior to Catastrophic Forgetting.

A.1.3 Training Statistics for ALMA without LoRA

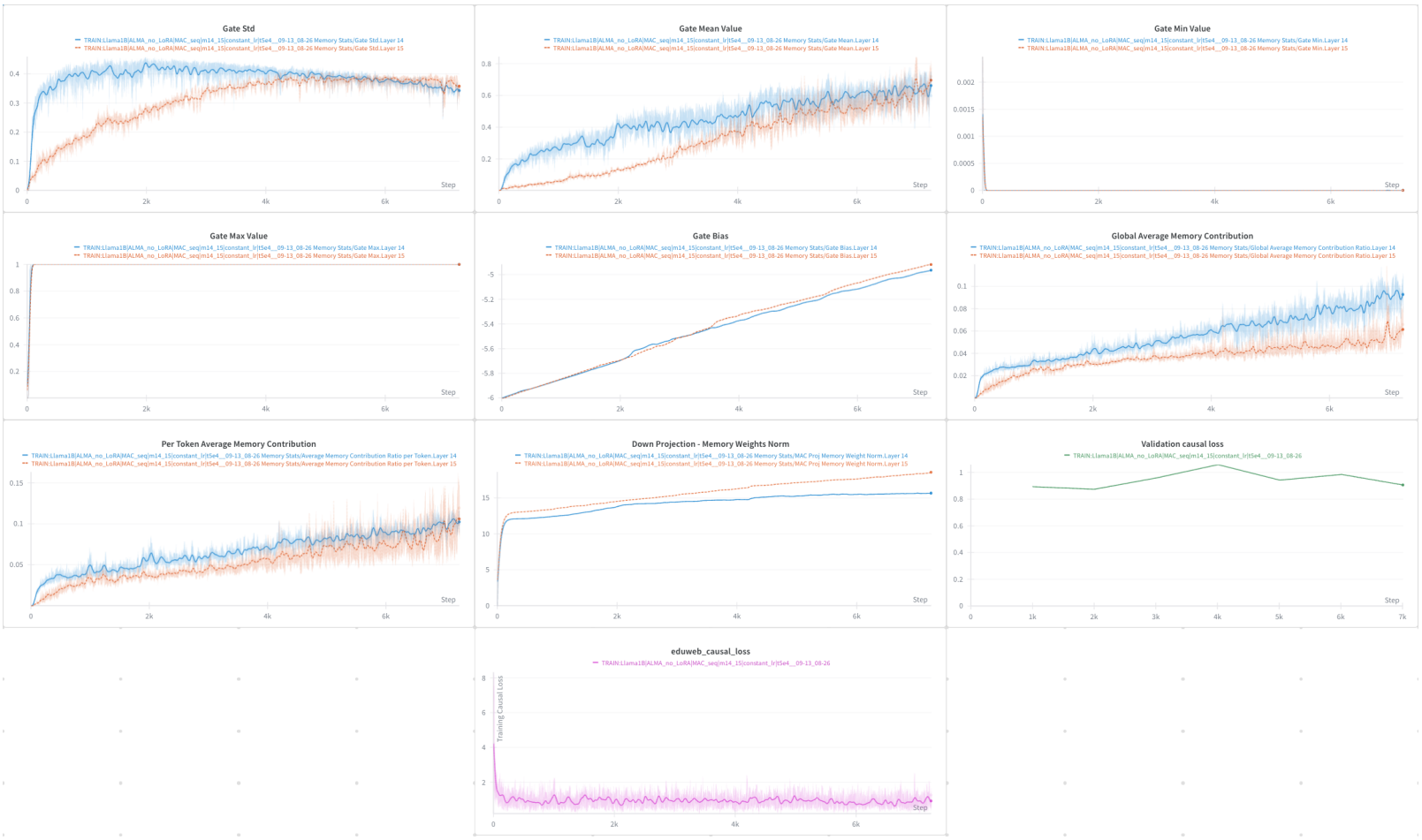


Figure A.5: Training Statistics for ALMA without LoRA.

A.2 Additional Figures

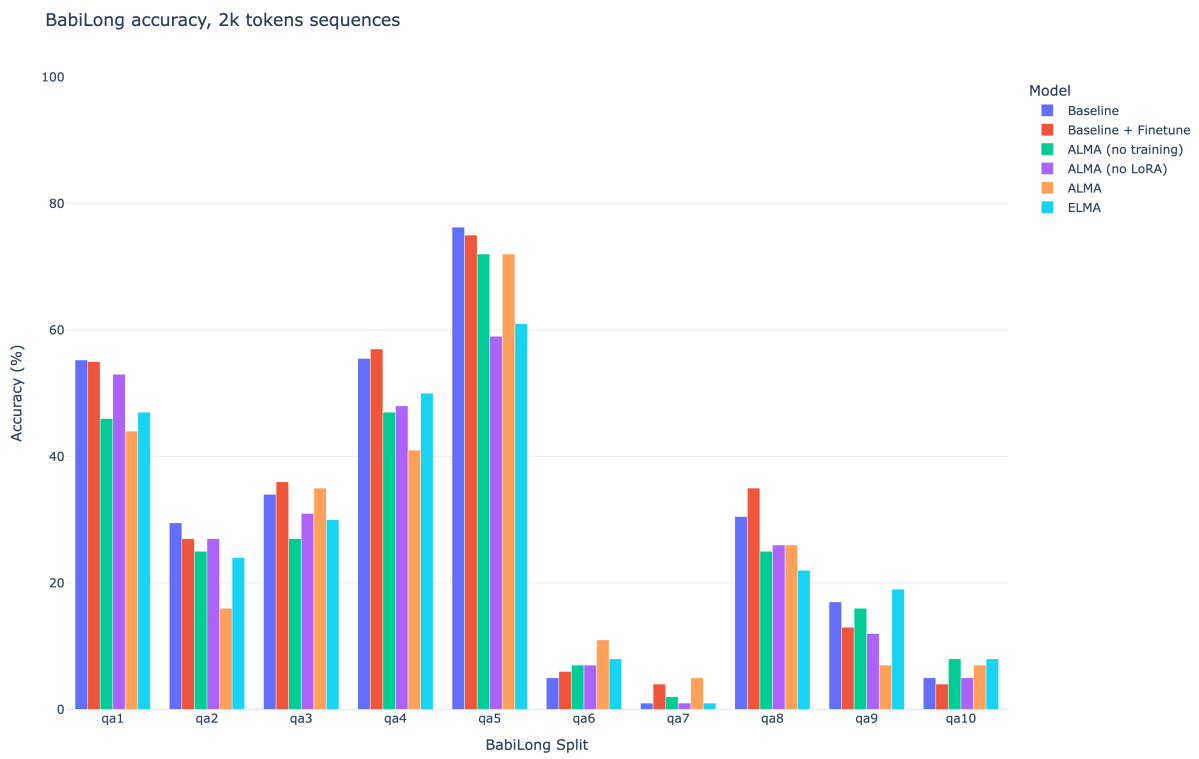


Figure A.6: Accuracy per model and task on the BabiLong benchmark for sequences of length 2000.

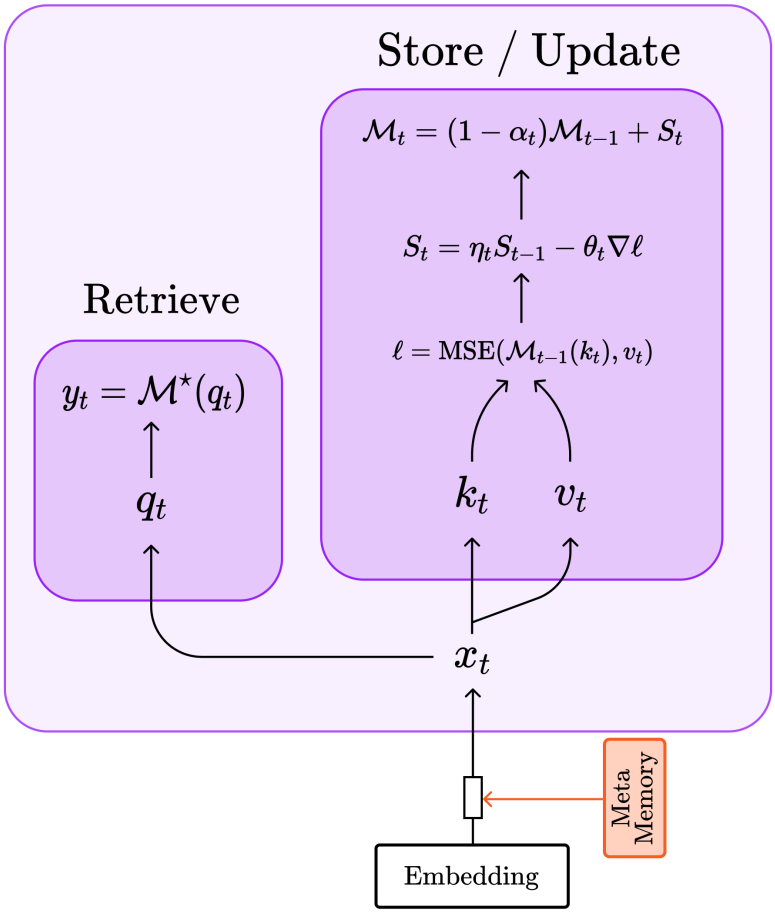


Figure A.7: Read and write operation in neural memory modules.

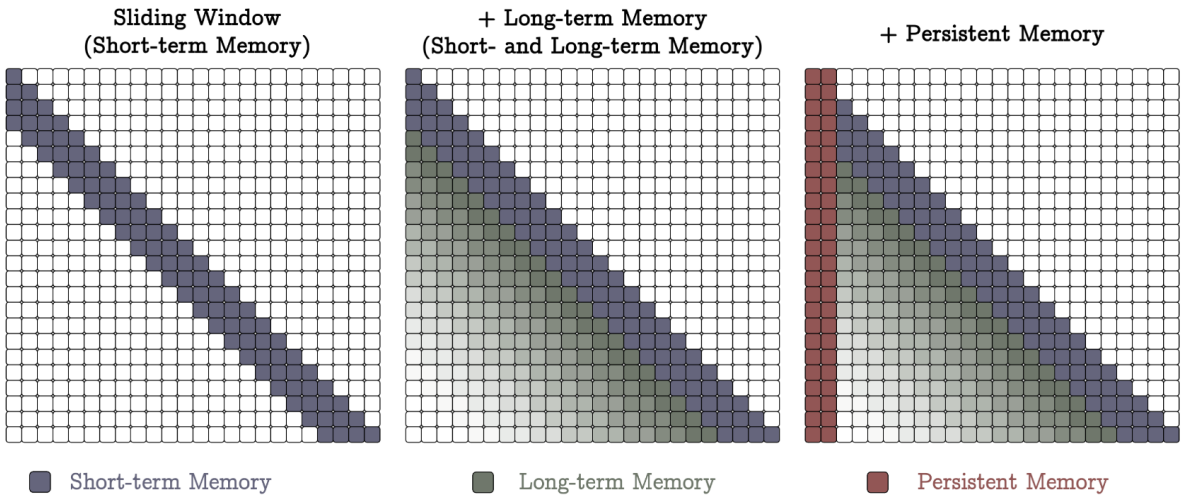


Figure A.8: Sliding-window attention mask used in MAG, source: Behrouz et al. (2024)

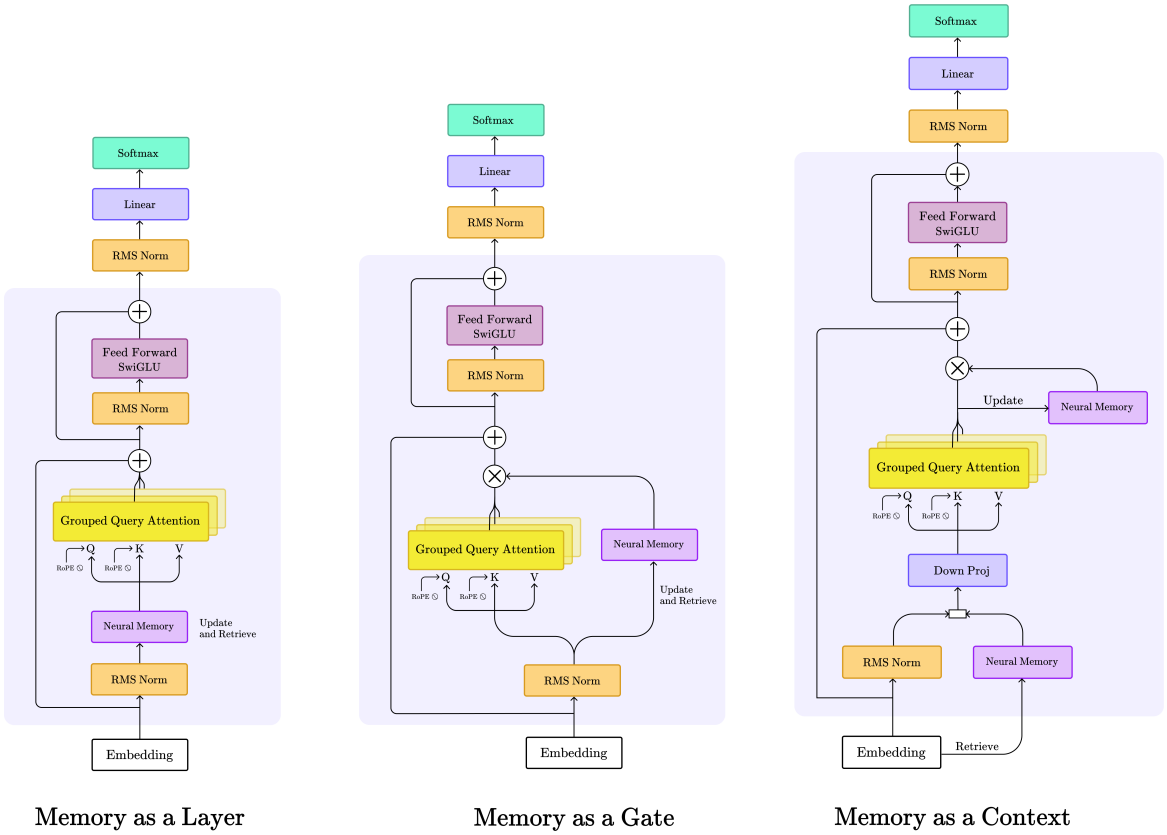


Figure A.9: MemoryLlama: Titans architectures adapted to the Llama framework.

Appendix B

Hyperparameters

B.1 MemoryLlama (ALMA & ELMA) Hyperparameters

Parameter	Value
Experiment	
name	eduweb_pt
seed	0
epochs	1
eduweb_train_batch_size	4
eduweb_val_batch_size	4
babilong_test_batch_size	1
gradient_accumulation_steps	4
learning_rate	1e-3
weight_decay	0.1
scheduler	constant
num_train_samples	5e4
num_val_samples	100
num_eval	50
num_test_samples	100
max_seq_len	2048
train_task_name	2k
test_task_name	2k
train_splits	~
test_splits	all
eval_with_generate	true
max_gen_tokens	25
save_every	50
save_recent_n	3
log_experiment	true
resume_from_checkpoint	false
checkpoint_name	~
resume_from_step	~
Neural Memory	
mlp_depth	3
mlp_expansion_factor	2
max_adaptive_lr	1e-2
Memory LLaMA	
memory_arch	context
llama_hf_path	meta-llama/Llama-3.2-1B
memory_layer_id	[14, 15]
use_lora	true
track_memory_statistics	true
LoRA	
lora_rank	16
lora_alpha	32
lora_dropout	0.05
lora_target_modules	q_proj, v_proj, k_proj, o_proj, gate_proj, down_proj, up_proj

Table B.1: Configuration for the ALMA experiment.

Parameter	Value
Experiment	
name	eduweb_pt
seed	0
epochs	1
eduweb_train_batch_size	4
eduweb_val_batch_size	4
babilong_test_batch_size	1
gradient_accumulation_steps	4
learning_rate	1e-3
weight_decay	0.1
scheduler	constant
num_train_samples	5e4
num_val_samples	100
num_eval	50
num_test_samples	100
max_seq_len	2048
train_task_name	2k
test_task_name	2k
train_splits	~
test_splits	all
eval_with_generate	true
max_gen_tokens	25
save_every	50
save_recent_n	3
log_experiment	true
resume_from_checkpoint	false
checkpoint_name	~
resume_from_step	~
Neural Memory	
mlp_depth	3
mlp_expansion_factor	2
max_adaptive_lr	1e-2
Memory LLaMA	
memory_arch	context
llama_hf_path	meta-llama/Llama-3.2-1B
memory_layer_id	[1, 2]
use_lora	true
track_memory_statistics	true
LoRA	
lora_rank	16
lora_alpha	32
lora_dropout	0.05
lora_target_modules	q_proj, v_proj, k_proj, o_proj, gate_proj, down_proj, up_proj

Table B.2: Configuration for the ELMA experiment.