
An Evolved Universal Transformer Memory

Edoardo Cetin, Qi Sun, Tianyu Zhao, Yujin Tang
Sakana AI, Japan
`{edo,qisun,tianyu,yujintang}@sakana.ai`

Abstract

We introduce Neural Attention Memory Models (NAMMs) to improve the performance and efficiency of transformer foundation models. NAMMs are *evolved* atop pre-trained transformers to provide different latent contexts containing the most relevant information for individual layers and attention heads. NAMMs are universally applicable to any model using self-attention as they condition exclusively on the attention matrices produced in each layer. NAMMs learned on a relatively small set of problems substantially improve performance across multiple unseen long-context language tasks while cutting the model’s input contexts up to a fraction of the original sizes, setting them apart from prior hand-designed KV cache eviction strategies that only aim to preserve model behavior. We show the generality of our conditioning enables zero-shot transfer of NAMMs trained *only* on language to entirely new transformer architectures even across input modalities, with their benefits carrying over to vision and reinforcement learning. Our source code is available at <https://github.com/SakanaAI/evo-memory>.

1 Introduction

Transformer architectures have become the golden standard in deep learning, with ubiquitous applications in the design of modern foundation models, exhibiting exceptional performance and scalability [1–7]. The outputs of a transformer are solely conditioned on a recent context of input tokens, which for language models (LMs) generally correspond to a window of preceding words. Thus, tractably extending this context window is critical for long-range tasks and has emerged as a focal area of research [8]. However, longer contexts also immediately introduce constraints and efficiency trade-offs, hindering efficiency and scalability during training and inference.

We start tackling this challenge by focusing on the inference stage of transformers. During this stage, the context of a transformer is maintained by what is referred to as the Key-Value (KV) cache, a buffer of the latent representations for the keys and values of the most recent input tokens. The KV cache can be seen as the memory of a transformer, allowing for tractable auto-regressive conditioning without incurring quadratic costs during self-attention for each newly generated token. However, even

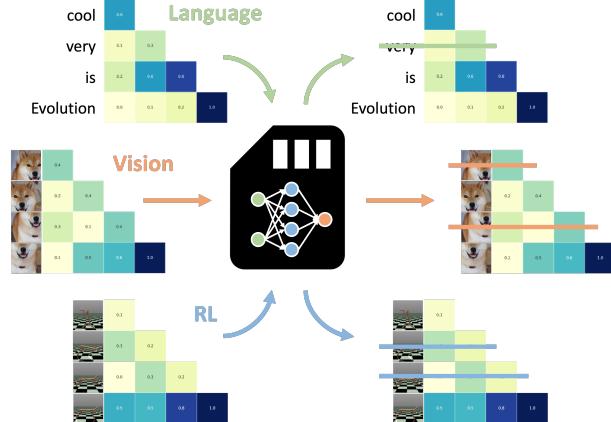


Figure 1: NAMMs use evolution to optimize the performance of LMs by pruning their KV cache memory. Evolved NAMMs can be zero-shot transferred to other transformers, even across input modalities and task domains.

maintaining a long KV cache on GPU devices for efficient execution can be prohibitively expensive. Based on these considerations and motivated by the observed sparsity in self-attention matrices, recent studies in language modeling have proposed several simple hand-designed heuristic strategies to prune the KV cache, attempting to retain most of the original model’s behavior and performance [9].

In contrast, our research aims to go beyond hand-designed strategies as we hypothesize that the conditioning challenge entails new opportunities to improve the capabilities of transformers in downstream applications. A widely evidenced example in support of our hypothesis is the effectiveness of input modifications through prompt engineering [10], even allowing foundation models to learn *in-context* entirely new skills at test time [11]. Furthermore, managing memory at the token level rather than the prompt level could enable providing targeted contexts to individual layers and attention heads, allowing them to focus on the most relevant information for their specific needs.

Motivated by these considerations, we propose using evolutionary optimization to directly learn a new memory system for transformers. We parameterize this framework via what we refer to as Neural Attention Memory Models (NAMMs), learning to preserve only the task-critical tokens from long contexts to maximize downstream performance. Evolution inherently overcomes the non-differentiability of memory management operations with binary outcomes (selecting tokens to preserve/discard) which render popular gradient-based methods incompatible. Our efforts are inspired by the key role that natural evolution played in shaping human memory, which analogously appears to selectively incorporate and actively prune information based on its lifelong usefulness [12–14].

Our NAMMs are conditioned on features entirely constructed from the attention matrix, making them universally applicable to any transformer-based architecture. Learning NAMMs atop a pre-trained Llama 3 8B model [15], we not only obtain efficiency benefits, with substantial reductions in the number of retained tokens in the KV cache, but also *exceed* the performance of the full-context model with notable margins. We validate these findings across 36 different tasks from LongBench [16], InfiniteBench [17], and *ChouBun*¹, a new Japanese benchmark designed to assess long-context capabilities beyond the common English and Chinese. These results are in clear contrast to prior KV cache eviction strategies, often unable to even preserve the original full-context performance.

Furthermore, we show that the generality of our parameterization enables *zero-shot transfer* of NAMMs trained on three natural language tasks to entirely new transformer models. In particular, we observe performance and efficiency gains not only when using the evolved NAMMs with other LMs of increased size, but also transformers concerned with entirely new input modalities designed for vision and reinforcement learning (RL). Our technical contributions can be summarized as follows:

- We introduce NAMMs, a novel memory evolution framework that adds a new dimension to optimizing transformer models without altering their powerful architectures.
- We design and successfully train NAMMs on top of pre-trained transformer models, obtaining both performance and efficiency gains on several long context language tasks.
- We show NAMMs, trained only on language tasks, can be transferred zero-shot to any other transformers, retaining benefits across different input modalities and task domains.

2 Background and preliminaries

Attention and transformers. Transformers are neural network architectures designed specifically for efficiently processing input sequences. These models take as input a stream of input *token embeddings* and produce a set of latents with the same length within their layers. The *self-attention*

¹ChouBun is the pronunciation of “長文”, literally translating to “long text” in Japanese.

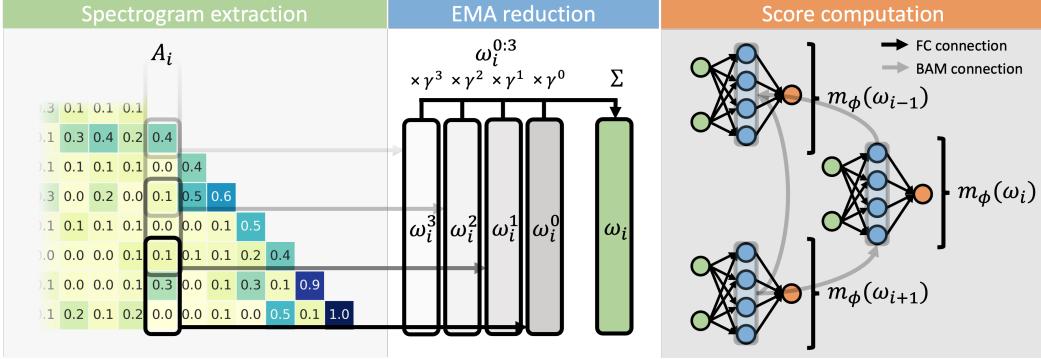


Figure 2: Schematic depiction of our Neural Attention Memory Model design. We extract features from a spectrogram over the KV cache tokens attention values (left), which we reduce via an element-wise EMA (center). These features are fed to m_ϕ with fully connected (FC) and cross-token BAM connections (right). layer [18], characterizes modern transformers, facilitating effective information sharing across token representations. This layer conducts a set of parallel computations, each known as an attention head, mapping tokens to query, key, and value vectors $\in \mathbb{R}^d$. These vectors are organized along the sequence dimension in the matrices Q , K , and V , and the layer’s output is computed as:

$$\text{attention}_M(Q, K, V) = AV, \quad \text{where}, \quad A = \text{softmax} \left(M \times \frac{QK^T}{\sqrt{d}} \right). \quad (1)$$

Here, M represents an optional mask multiplying the *attention matrix* A , usually enforcing an *causal conditioning* such that each token cannot attend to its future. An interpretation of the attention layer comes from the elements of the attention matrix A_i^j , i.e., the dot products between each key i and query j normalized along the column dimension. Intuitively, each of these values can be understood as the relative *importance* of token i in processing the input representation of token j .

Frequency-based feature extraction. An established canonical technique to pre-process one-dimensional non-stationary signals is the Short-Time Fourier Transform (STFT) [19]. This technique has seen plenty of applications for feature extraction concerning audio, biomedical, seismic, and many more kinds of modalities. The STFT performs a time-convolution of a signal, shifting each convolutional window to the frequency domain through a discrete Fourier transform, producing a *spectrogram* representation of the original input. We use $\omega^t \in \mathbb{R}^{M+1}$ to denote the fixed-sized vector produced at each timestep t , where the M frequencies span from zero up to the *Nyquist* frequency. Concretely, the m -th frequency from an STFT for time t is extracted from an input $v \in \mathbb{R}^T$ as:

$$\omega^t[m] = \sum_{t'=0}^T v[t']w[t-t']e^{-\frac{m\pi t}{M}}. \quad (2)$$

Here, the convolutional filter of the STFT is defined by the product of a finite-length *window function* w with each exponential term in the Fourier transform. A popular choice for w is the Hann window [20], designed to minimize frequency overestimation caused by *spectral leakage* [21].

3 Neural Attention Memory Models

An immediate limitation of transformers is the quadratic costs associated with computing the attention matrix A . To partially address this issue, during auto-regressive generation, the latents for the keys and values of the tokens generated at the previous steps are usually stored in what is referred to as the KV cache. This object can be regarded as being analogous to the *memory* of the transformer, which now, at each step, only needs to compute the query, key, and value of the latest token and perform attention over a horizontal vector by exploiting causal ordering. In this section, we describe the feature extraction, architecture, and optimization of NAMMs, which have been designed to act on the KV cache to improve both the performance and practicality of this powerful class of models.

3.1 Attention spectrograms for model-agnostic feature extraction

The feature extraction framework of NAMMs is designed to be agnostic to the parameterization of the base transformer they are applied for. In particular, we build a representation for each token

in the current KV cache memory from its corresponding column vector in the attention matrix A_i . This vector contains precisely the knowledge of each token’s relative importance for all the past-encountered queries, discarding all other information specific to the learned transformer weights. To meaningfully compress this unbounded vector signal, we process it via a short-time Fourier transform with a fixed-sized Hann window (Fig. 2, left). This operation produces a spectrogram representation of the attention columns ω_i^t , representing the frequencies with how the queries attend to each of the stored key tokens (indexed by i) on a compressed time-axis (indexed by t).

As NAMMs rely only on the attention values for their input, they are universally applicable to any layer producing an attention matrix. This property enables us to avoid learning individual NAMMs for different layers, thus, greatly limiting the number of total optimized parameters. Furthermore, it allows efficient training on top of smaller transformers and later zero-shot test-time transfer.

3.2 Memory model design and cross-token communication

NAMMs parameterize a small neural network m_ϕ to output a scalar *selection score* $s_i = m_\phi(\omega_i^{1:T})$ for each i^{th} token in the KV cache. First, to obtain a consistent input dimension, we reduce the attention spectrogram into a smaller feature vector ω_i by compressing the time-axis via an element-wise exponentially moving average (EMA: $\omega_i = \sum_t \gamma^t \omega_i^t$; Fig. 2, center). We then append positional encodings and feed the vector ω_i to the memory model’s network m_ϕ to produce the score s_i . Finally, we evict from the KV cache memory all latent tokens with $s_i < 0$, treating the problem as binary classification. We repeat this process with a fixed interval, every set number of input tokens, n_{up} .

Backward attention memory models (BAM). For the design of m_ϕ , we posit that sharing information from all tokens in memory could be key for assessing their importance. A particularly motivating scenario in LMs arises when considering the case of repeated words or sentences, where learning a diversity measure that compares different tokens would allow preventing redundancies in the KV cache. Corroborating this intuition, even from a biological perspective, memory formation and retention appear to adhere to models of neuronal competition [22]. Based on these considerations, we design the backward attention memory architecture (BAM) for parameter-efficient sharing of information while making use of the powerful inductive biases enabled by masked self-attention. In particular, we implement m_ϕ via an initial self-attention layer with a *counter-causal* mask \hat{M} , which we refer to as *backward* (Fig. 3). This design serves to introduce a purposeful asymmetry that allows our NAMM to distinguish between older and newer tokens. We then output s_i from a final linear operation:

$$o_i = \text{attention}_{\hat{M}}(K_\Omega, V_\Omega, Q_\Omega), \quad s_i = \text{linear}(o_i), \quad (3)$$

where $K_\Omega, V_\Omega, Q_\Omega$ are the key, value, and query matrices from all feature vectors ω_i in memory.

3.3 Training NAMMs with incremental evolution

We evolve our NAMMs to directly optimize performance on a subset of long-context language modeling tasks from LongBench [16]. As we share a single m_ϕ across all layers, even with our largest NAMM we only evolve about 4000 parameters. We use CMA-ES [23] and apply NAMMs atop a Llama 3 8B base model [15] with a context extended from 8192 to 32768 tokens via NTK positional interpolation [24]. Due to the inference costs of LMs with long inputs, we sample a subset of different prompts from each task in each generation and propose training in an *incremental* fashion: starting from a single task, and adding additional tasks at later training stages. Empirically, we found both these choices to provide effective regularization, improving generalization (see App. C). The performance of modern LMs on LongBench varies considerably across tasks, and even across different task prompts. Hence, instead of using the raw scores, we opt to maximize normalized performance relative to the base model’s full-context performance on each same subset of prompts.

We choose three tasks from different LongBench categories where Llama 3 seems to particularly struggle: PassageRetrieval-en, DuReader, and NarrativeQA. We evolve our NAMM for 300 generations in its first incremental phase, 250 in its second, and 120 in its third, to counteract the increasing

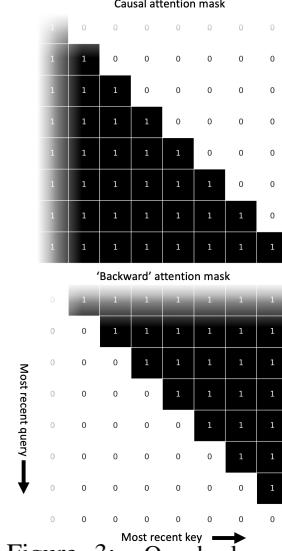


Figure 3: Our backward mask makes each token attend exclusively to its future.

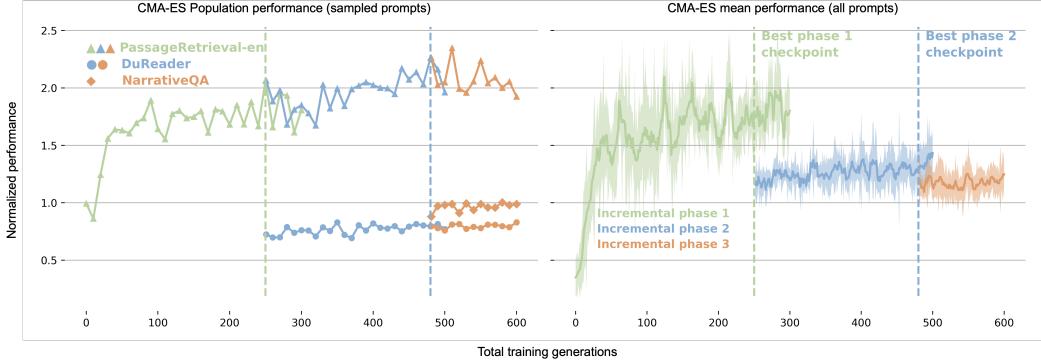


Figure 4: Mean and standard deviation over the CMA-ES population batch performance (left), together with the performance of the learned mean parameter on each task (right).

Table 2: NAMMs evaluation on LongBench. The normalized performance (in brackets) is calculated using the base model with full cache. The tasks used for NAMM’s training are highlighted in gray.

Model/Task id	Single-Doc QA				Multi-Doc QA				Summarization			
	1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Base model	10.38 (1.00)	12.79 (1.00)	22.60 (1.00)	21.31 (1.00)	10.41 (1.00)	12.67 (1.00)	7.54 (1.00)	25.86 (1.00)	29.34 (1.00)	23.93 (1.00)	0.92 (1.00)	2.66 (1.00)
H2O	8.75 (0.84)	13.07 (1.02)	22.11 (0.98)	21.62 (1.01)	10.28 (0.99)	12.40 (0.98)	7.20 (0.95)	26.58 (1.03)	28.56 (0.97)	23.98 (1.00)	0.88 (0.96)	2.25 (0.85)
L2	8.83 (0.85)	13.13 (1.03)	22.22 (0.98)	21.79 (1.02)	9.97 (0.96)	12.15 (0.96)	5.88 (0.78)	24.96 (0.97)	28.05 (0.96)	23.28 (0.97)	1.15 (1.25)	1.52 (0.57)
NAMM (Ours)	9.14 (0.88)	12.63 (0.99)	21.94 (0.97)	21.34 (1.00)	9.71 (0.93)	11.63 (0.92)	6.98 (0.93)	20.58 (0.80)	28.78 (0.98)	24.39 (1.02)	1.04 (1.13)	3.63 (1.36)

Model/Task id	Few-shot Learning				Synthetic			Code		Overall		
	4-1	4-2	4-3	4-4	5-1	5-2	5-3	6-1	6-2	All tasks	Test tasks	Cache size
Base model	73.00 (1.00)	89.45 (1.00)	46.54 (1.00)	40.00 (1.00)	1.48 (1.00)	12.18 (1.00)	28.80 (1.00)	69.09 (1.00)	65.17 (1.00)	28.86 (1.00)	N/A	32768 (1.00)
H2O	73.00 (1.00)	90.03 (1.01)	46.48 (1.00)	34.00 (0.85)	2.18 (1.47)	9.93 (0.82)	27.76 (0.96)	69.37 (1.00)	65.44 (1.00)	28.37 (0.99)	N/A	8192 (0.25)
L2	66.41 (0.91)	84.92 (0.95)	45.78 (0.98)	34.38 (0.86)	3.13 (2.11)	11.00 (0.90)	28.68 (1.00)	73.45 (1.06)	55.20 (0.85)	27.42 (1.00)	N/A	8192 (0.25)
NAMM (Ours)	73.00 (1.00)	89.81 (1.00)	46.35 (1.00)	40.00 (1.00)	3.04 (2.05)	27.55 (2.26)	28.60 (0.99)	69.53 (1.01)	66.35 (1.02)	29.33 (1.11)	1.07	8155 (0.25)

costs of later phases and make more efficient use of computational resources. At the end of each phase, we resume from the best previous checkpoint. We provide training curves of our main NAMM with BAM in Fig. 4, showing the average and standard deviation of the normalized batch performance across the population (left), together with the normalized per-task performance on all samples of the optimized mean from CMA-ES (right). We refer to App. A for additional details and the set of hyper-parameters. We also provide statistics and training curves for other NAMMs in App. C.

4 Experimental Results

In this section, we evaluate and analyze evolved NAMMs as compared to full-context transformers and two recent hand-designed methods for KV cache management: H2O [25] and L2 [26]. We compare each method in terms of absolute and normalized performance and also provide the resulting average cache size recorded at the end of each prompt. We first consider three long-context language modeling benchmarks spanning 36 diverse tasks in three languages, atop the same Llama 3 8B model from training. Then, we evaluate the zero-shot transfer of NAMMs to other *unseen* transformers and task domains. In particular, we not only consider transfer to larger LMs, but also transformers with tokens constructed from modalities other than language. For all these settings, we provide additional results in App. C: evaluating NAMMs with simpler architectures, different training procedures, and comparing performance at every stage of incremental evolution. Lastly, we perform a targeted analysis to understand the behavior of our new memory framework which we later extend in App. D.

4.1 Long-context language understanding

Longbench. In Table 2, we provide results across all LongBench tasks [16]. Our NAMM yields concrete improvements to the Llama 3 transformer both when considering the full set or exclusively the held-out set of *test* tasks that were not used for evolution, with improvements of 11% and 7%. At the same time, our NAMM also yields efficiency benefits, notably reducing the context-extended KV cache size. Instead, while both H2O and L2 produce even smaller cache sizes, they both come with some performance costs - in line with their stated objective of *retaining* rather than *improving* the original full-context performance. These results indicate how two different hand-designed extremes,

Table 3: NAMMs evaluation on InfiniteBench. The normalized overall performance (in brackets) is calculated using the average performance of the base model with full cache.

Model/Task name	Retrieval			Dialogue			Novel			Math			Code			Overall	
	Ret.PassKey	Ret.Number	Ret.KV	En.Dia	En.Sum	En.MC	En.QA	ZH.QA	Math.Find	Code.Run	Code.Debug	All tasks	Cache size				
Base model	0.00	0.00	0.00	1.00	7.73	0.00	1.05	1.79	0.00	0.00	0.00	1.05 (1.00)	32747 (1.00)				
H2O	0.00	0.00	0.00	1.50	5.38	0.00	1.01	1.71	1.71	0.25	0.00	1.05 (1.00)	8193 (0.25)				
L2	0.00	0.00	0.00	1.00	5.41	0.44	0.83	2.59	7.43	0.25	0.00	1.63 (1.55)	8193 (0.25)				
NAMM (Ours)	11.86	11.86	1.80	1.00	14.91	36.24	8.78	17.67	10.57	1.75	4.57	11.00 (10.45)	13192 (0.40)				

Table 5: NAMMs evaluation on LongBench with a Llama 3 70B model. The normalized performance (in brackets) is calculated using the base model with full cache.

Model/Task id	Single-Doc QA				Multi-Doc QA				Summarization				All tasks	Test tasks	Cache size
	1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4			
Base model	9.38 (1.00)	13.84 (1.00)	24.99 (1.00)	17.78 (1.00)	11.73 (1.00)	14.26 (1.00)	8.11 (1.00)	26.43 (1.00)	13.13 (1.00)	24.55 (1.00)	23.20 (1.00)	10.08 (1.00)			
H2O	8.80 (0.94)	13.48 (0.97)	25.02 (1.00)	18.44 (1.04)	12.36 (1.05)	14.32 (1.00)	8.15 (1.01)	26.22 (0.99)	13.37 (1.02)	24.50 (1.00)	23.20 (1.00)	9.22 (0.91)			
L2	8.57 (0.91)	13.40 (0.97)	24.70 (0.99)	17.94 (1.01)	12.77 (1.09)	13.85 (0.97)	7.13 (0.88)	25.74 (0.97)	12.78 (0.97)	23.21 (0.95)	23.35 (1.01)	8.45 (0.84)			
NAMM (Ours)	9.13 (0.97)	13.53 (0.98)	24.25 (0.97)	17.82 (1.00)	11.45 (0.98)	13.76 (0.96)	8.34 (1.03)	21.79 (0.82)	12.66 (0.96)	24.21 (0.99)	23.56 (1.02)	8.62 (0.86)			

Model/Task id	Few-shot Learning				Synthetic			Code			Overall			All tasks	Test tasks	Cache size
	4-1	4-2	4-3	4-4	5-1	5-2	5-3	6-1	6-2							
Base model	78.00 (1.00)	92.43 (1.00)	48.67 (1.00)	45.50 (1.00)	22.50 (1.00)	75.37 (1.00)	33.89 (1.00)	74.60 (1.00)	71.19 (1.00)	35.22 (1.00)	N/A	10107 (1.00)				
H2O	77.50 (0.99)	92.43 (1.00)	48.33 (0.99)	39.75 (0.87)	18.12 (0.81)	64.69 (0.86)	33.89 (1.00)	74.61 (1.00)	71.09 (1.00)	34.17 (0.97)	N/A	6662 (0.66)				
L2	76.50 (0.98)	93.22 (1.00)	46.15 (0.95)	36.25 (0.80)	16.98 (0.75)	64.34 (0.85)	36.28 (1.07)	74.38 (1.00)	67.43 (0.95)	33.50 (0.95)	N/A	6662 (0.66)				
NAMM (Ours)	78.50 (1.01)	92.36 (1.00)	48.49 (1.00)	45.50 (1.00)	19.07 (0.85)	74.19 (0.98)	34.28 (1.01)	74.71 (1.00)	72.42 (1.02)	34.70 (0.99)	0.99	8365 (0.83)				

either retaining all tokens or aggressively dropping them, come with their own set of downsides for either efficiency or performance. On the other hand, NAMMs are the only methods making consistent improvements from the base model across both axes, highlighting how end-to-end optimization can open new orthogonal directions beyond what is feasible with manually-designed heuristics.

InfiniteBench. In Table 3, we provide results across the InfiniteBench tasks [17]. In this benchmark, the average prompt length is close to 200K tokens making it extremely challenging, especially for LMs that were not expensively finetuned for very long context understanding. In fact, as reported by Zhang et al. [17], even GPT4 [1] cannot exceed a performance of 1% on some of its problems. In line with these results, the full-context Llama 3 together with H2O and L2 obtain near-zero performance on most tasks. Instead, our NAMM provides outstanding improvements, bringing overall benchmark performance from 1.05% to 11%. We also observe that our NAMM’s memory size is considerably lower in relation to the base model’s as compared to in LongBench (now only 40%). This result suggests that NAMMs learned a scalable memory strategy, forgetting redundant and detrimental information at an increasing rate with longer contexts without the hand-designed hard cache limits enforced by L2 and H2O.

Table 4: Evaluation on the new ChouBun benchmark.

Model/Task	Extractive QA				Summarization			Overall			All tasks	Cache size
	JA.WikiQA	JA.EdinetQA	JA.CorpSecQA	JA.CorpSecSum								
Base model	22.91 (1.00)	28.34 (1.00)	11.83 (1.00)	21.75 (1.00)	21.21 (1.00)	12099 (1.00)						
H2O	20.76 (0.91)	26.39 (0.93)	10.42 (0.88)	21.87 (1.01)	19.86 (0.94)	8292 (0.69)						
L2	19.60 (0.86)	24.06 (0.85)	8.23 (0.70)	23.83 (1.10)	18.93 (0.89)	8292 (0.69)						
NAMM (Ours)	21.34 (0.93)	28.61 (1.01)	14.64 (1.24)	33.15 (1.52)	24.44 (1.15)	9895 (0.82)						

metrics for a wider range of popular LMs in App. B. In Table 4, we report our results evaluating NAMMs. Once again, we observe a clear contrast with prior hand-designed methods. While integrating either H2O or L2 leads to notable performance drops, our NAMM provides substantial improvements, with overall performance up by 15% from the full-context Llama 3 base model.

4.2 Zero-shot transfer across architectures and modalities

Cross-scale adaptation. In Table 5, we provide results zero-shot transferring our NAMM to the Llama 3 70B model on LongBench. Across all tasks, we find performance to be very close to the full-context baseline with an overall gap of less than 1% even for the test subset. While NAMMs are not able to improve the overall full-context performance in this setting outside specific

Table 6: Evaluation on the LongVideoBench and MLVU benchmarks with a Llava Next Video 7B model.

Model/Task name	LongVideoBench	MLVU	All tasks	Cache size
Base model	43.45 (1.00)	44.23 (1.00)	43.84 (1.00)	7039 (1.00)
H2O	40.91 (0.94)	43.03 (0.97)	41.97 (0.96)	4479 (0.64)
L2	40.84 (0.94)	42.07 (0.95)	41.45 (0.95)	4479 (0.64)
NAMM (Ours)	44.58 (1.03)	44.18 (1.00)	44.38 (1.01)	5100 (0.72)

Table 7: Evaluation on D4RL with a Decision Transformer. The normalized performance (in brackets) is calculated using the base model with full cache.

Model/Task name	Hopper-v3			Walker2d-v3			HalfCheetah-v3			Overall	
	Medium	Med-Replay	Expert	Medium	Med-Replay	Expert	Medium	Med-Replay	Expert	All tasks	Cache size
Base model	33.36 (1.00)	18.37 (1.00)	44.62 (1.00)	68.21 (1.00)	7.18 (1.00)	38.98 (1.00)	34.91 (1.00)	5.06 (1.00)	10.64 (1.00)	29.04 (1.00)	3000 (1.00)
H2O	33.19 (1.00)	17.86 (0.97)	49.10 (1.10)	67.63 (0.99)	7.59 (1.06)	40.03 (1.03)	26.73 (0.77)	4.46 (0.88)	11.74 (1.10)	28.70 (0.99)	2048 (0.68)
L2	32.85 (0.98)	17.96 (0.98)	43.75 (0.98)	65.47 (0.96)	7.18 (1.00)	40.64 (1.04)	30.10 (0.86)	4.76 (0.94)	8.52 (0.80)	27.91 (0.96)	2048 (0.68)
NAMM (Ours)	36.10 (1.08)	18.86 (1.03)	49.39 (1.11)	70.87 (1.04)	7.53 (1.05)	50.02 (1.28)	34.56 (0.99)	5.90 (1.17)	12.34 (1.16)	31.73 (1.09)	2434 (0.81)

task categories (e.g., coding and few-shot learning), they still outperform both H2O and L2 baselines and retain a similar efficiency as with their original training transformer.

Vision Language Understanding. In Table 6, we provide results zero-shot transferring NAMMs to the vision domain atop a Llava Next Video 7B transformer [27] on LongVideoBench [28] and Multi-Task Long Video Understanding (MLVU) [29]. As when evaluated with Llama 8B, our NAMM is the only method recording gains over the full-context model in both tasks. We also find that our NAMM learns to act almost exclusively on images even though it never trained with such modality, forgetting redundant video frames rather than the language prompt descriptions, validating its flexibility.

Reinforcement learning. In Table 7, we provide our zero-shot transfer results for the offline RL, where we apply NAMMs atop a decision transformer [30] and consider the canonical continuous-control tasks from the D4RL benchmark [31]. We find our NAMM improves the base transformer quite considerably in this domain across eight out of nine offline tasks with over 9% overall gains, opposing the performance loss of the other efficient baselines. We posit that since the nature of the decision transformer optimization is closely tied to behavior cloning, the ability to discard part of the context allows NAMMs to *forget* and avoid imitating past mistakes autoregressively. In support of this hypothesis, we observed slightly higher average rewards in the transitions for the retained tokens (by 1.4%, 0.8%, and 12.3% for the Hopper, Walker2d, and HalfCheetah environments, respectively).

NAMMs comparison. In Table 8, we provide summarized results comparing NAMMs with either BAM or the simpler MLP architecture at the end of each stage of incremental evolution. First, we note that even the MLP NAMM after stage 1 impressively improves performance across all language benchmarks. Additionally, performance sees near-monotonic improvements with each additional stage of incremental evolution in both language and zero-shot transfer settings. Comparing our implementations, the performance benefits from the memory models with BAM appear consistently superior to the MLP. Moreover, on ChouBun, we observe that the performance with BAM sees a notable upswing after the second stage of incremental training, which might be associated with the introduction of another ideogram-based language in the training set.² The same improvement not occurring with the MLP-based NAMMs might be further evidence of architectural performance saturation, highlighting the effectiveness of our main implementation.

Table 8: Summarized comparison of different NAMMs in language modeling (top) and zero-shot transfer (bottom)

Model/Eval	LongBench		InfiniteBench		ChouBun	
	Performance	Cache size	Performance	Cache size	Performance	Cache size
Base model	28.86 (1.00)	32768 (1.00)	1.05 (1.00)	32747 (1.00)	21.21 (1.00)	12099 (1.00)
NAMM (MLP, s1)	28.83 (1.05)	7639 (0.23)	3.08 (2.93)	11329 (0.35)	22.09 (1.04)	9525 (0.79)
NAMM (MLP, s2)	29.22 (1.07)	8475 (0.26)	4.00 (3.80)	13031 (0.40)	22.06 (1.04)	9815 (0.81)
NAMM (BAM, s1)	28.91 (1.05)	7951 (0.24)	10.14 (0.63)	11173 (0.34)	22.73 (1.07)	9569 (0.79)
NAMM (BAM, s2)	29.25 (1.07)	8267 (0.25)	9.78 (0.29)	12789 (0.39)	24.05 (1.13)	9867 (0.82)
NAMM (BAM, s3)	29.33 (1.11)	8155 (0.25)	11.00 (0.45)	13192 (0.40)	24.44 (1.15)	9895 (0.82)

Model/Eval	Llama 3 70B		Computer Vision		Reinforcement Learning	
	Performance	Cache size	Performance	Cache size	Performance	Cache size
Base model	35.22 (1.00)	10107 (1.00)	43.84 (1.00)	7039 (1.00)	29.04 (1.00)	3000 (1.00)
NAMM (MLP, s1)	34.11 (0.97)	7930 (0.78)	40.44 (0.92)	584 (0.08)	29.30 (1.01)	1993 (0.66)
NAMM (MLP, s2)	34.29 (0.97)	8445 (0.84)	40.39 (0.92)	713 (0.10)	29.58 (1.02)	2834 (0.94)
NAMM (BAM, s1)	34.11 (0.97)	7947 (0.79)	41.52 (0.95)	723 (0.10)	30.44 (1.05)	2009 (0.67)
NAMM (BAM, s2)	25.20 (0.72)	8276 (0.82)	44.63 (1.02)	4948 (0.70)	31.53 (1.09)	2534 (0.84)
NAMM (BAM, s3)	34.70 (0.99)	8365 (0.83)	44.38 (0.01)	5100 (0.72)	31.73 (1.09)	2434 (0.81)

4.3 Understanding Neural Attention Memory Models

Influence of layer depth. We begin analyzing the final amount of retained tokens by NAMMs and their oldness³. At the top of Fig. 5, we provide these normalized metrics as a function of layer depth. Our learned NAMM does not appear to affect the KV cache uniformly, retaining visibly more and older tokens for some of the early-middle layers of the base transformer. One possible interpretation

²The DuReader task, used in the second stage of incremental training, uses the Chinese language.

³We define *oldness* of a retained token as the number of new queries since its introduction in the KV cache.

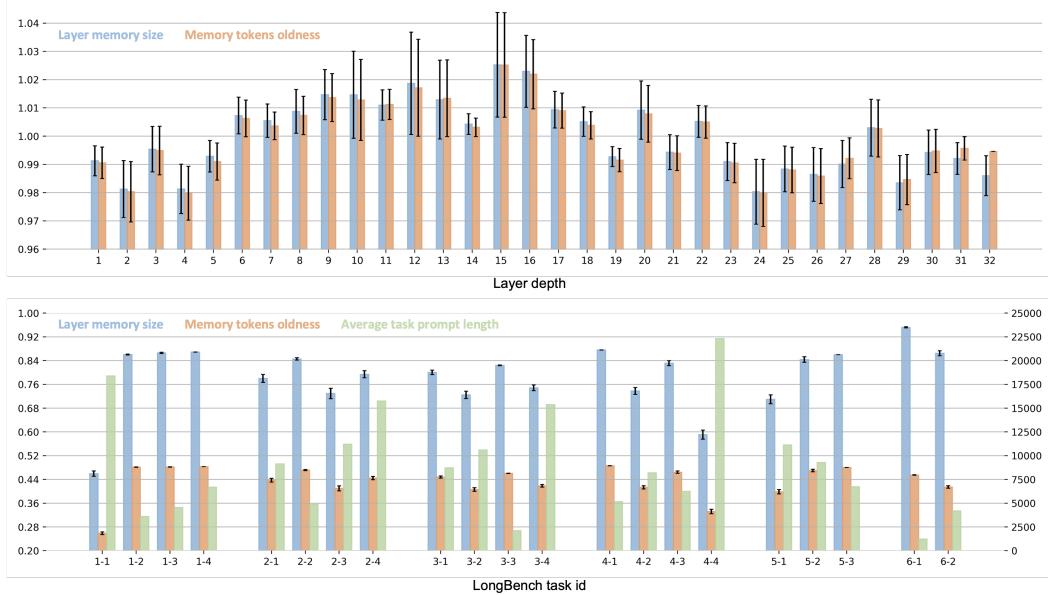


Figure 5: Memory size and token oldness recorded for each layer in Llama (top) or task in LongBench (bottom). These values are normalized with their average across tasks or the mean sample length respectively.

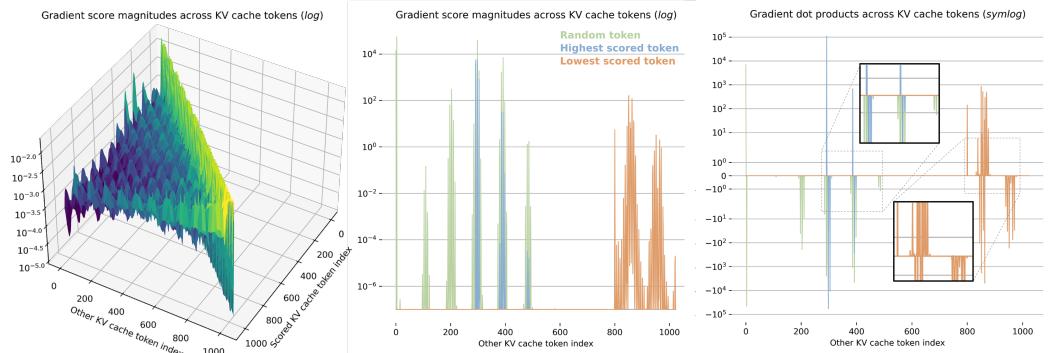


Figure 6: Density plot of gradient magnitudes for each score with respect to all memory tokens (left), together with a qualitative analysis extracting slices from three tokens (center) and computing the dot products of the gradients with the scored-token’s feature vector (right).

of our results, in line with recent analysis [32], is that these layers might be particularly important for aggregating information over longer contexts, thus requiring larger memories than the rest.

Influence of task structure. At the bottom of Fig. 5, we instead provide these metrics varying the source task, normalizing by the average prompt lengths shown in green. Our results show an inverse correlation between normalized memory size and prompt length, with a Pearson coefficient of -0.84, confirming our earlier observations of sub-linear memory growth and favorable scaling to longer prompts. In the code completion tasks with ids 6-*, we also observe that NAMMs learned to preserve visibly more tokens relative to the prompt lengths. This result appears intuitively consistent with the higher information density in code, leaving room for less redundancy than in natural language.

Backward attention cross-token interactions. We record the gradients of each token score s_i with respect to all input features v_j for *all tokens in memory* after storing 1024 tokens, i.e., for $j = 1, 2, \dots, 1024$. We denote these quantities as $\nabla g_i^j = \partial s_i / \partial v_j$ and illustrate our results on the PassageRetrieval-en task for a randomly selected layer and prompt in Fig. 6. On the left subplot, we visualize the squared magnitudes $(\nabla g_i^j)^T \nabla g_i^j$ for each combination of tokens (either scored or attended upon in BAM, i.e., indexed by i or j). Here, the effects of the backward mask are clearly visible, allowing tokens to exclusively attend to later ones. Predictably, these magnitudes mostly peak on the subplot’s diagonal, indicating the self-influence that each token’s features have on its corresponding score. However, we also find exceptions, as shown in the center subplot, where we overlap three slices from our surface plot for the gradients of the first, together with the highest and

lowest-scored tokens in memory (respectively indexed by $i = 0$, 292, and 800). We provide directional information of these gradient vectors in the right subplot, where we take their dot products with the scored token’s own feature vector $(\nabla g_i^j)^T v_i$. After the first spike at $i = j$, most other dot-products with the largest magnitudes consistently have negative values. Hence, we can deduce that the scores of these tokens would benefit from pushing the representations of future tokens away from their own. This result appears to validate the hypothesis that BAM learns a mechanism for cross-token competition, promoting tokens covering diverse frequencies in the attention spectrogram.

5 Related works

Devoto et al. [26] and Yao et al. [33] try to identify the least important tokens to evict using heuristics such as L2 magnitude and entropy. Alternative strategies include considering simple statistics from the attention matrix [25, 34, 35]. Ge et al. [36] and Li et al. [37] build on these ideas by applying multiple strategies based on matching specific attention patterns. Unlike this prior work, our approach uniquely employs a black-box model to *learn* token-level memory management and shows potential for improving both the performance and efficiency of transformers. We refer to App. E for references and connections to the wider literature, including efficient architectures, memory, and evolution.

6 Discussion and future work

This work introduced Neural Attention Memory Models, providing a new framework to enhance the performance of transformers while significantly reducing memory footprint. By evolving NAMMs on top of pre-trained LMs, we demonstrated their effectiveness across diverse long-context tasks in three languages, significantly surpassing previous hand-designed KV cache eviction frequently hindering performance, and the original model relying on costly full-context conditioning. Our carefully designed approach also enabled NAMMs, trained solely on language tasks, to achieve zero-shot transferability across architectures, input modalities, and task domains. This work has only begun to explore the design space of our memory models, which we anticipate might offer many new opportunities to advance future generations of transformers. In this regard, we believe NAMMs should not be viewed as a replacement for gradient-based optimization, but rather an orthogonal framework that could be combined and alternated with parameter fine-tuning. Such an extension has the potential to unlock efficient long-context training, drawing parallels to the iterative process of learning and evolution that shaped human memory.

Author contributions

Edoardo Cetin initiated the project, led the design and implementation of NAMMs, and provided major contributions to writing. Qi Sun designed and implemented the zero-shot transfer experiments with Llama 3 70B and Llava Next Video 7B, and provided contributions and feedback to writing. Tianyu Zhao devised and implemented ChouBun, and provided contributions and feedback to writing. Yujin Tang coordinated the project, gave key advice for the design of NAMMs, and provided major contributions to writing.

Acknowledgements

The authors would like to thank David Ha and Llion Jones for providing valuable discussions during the early stages and feedback while drafting the text. This paper is based on results obtained from a project, JPNP20017, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

- [2] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688*, 2023.
- [3] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- [4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [5] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.
- [6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.
- [7] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*, 2023.
- [8] Yunpeng Huang, Jingwei Xu, Zixu Jiang, Junyu Lai, Zenan Li, Yuan Yao, Taolue Chen, Lijuan Yang, Zhou Xin, and Xiaoxing Ma. Advancing transformer architecture in long-context large language models: A comprehensive survey. *arXiv preprint arXiv:2311.12351*, 2023.
- [9] Shi Luohe, Zhang Hongyi, Yao Yao, Li Zuchao, and Zhao Hai. Keep the cost down: A review on methods to optimize llm’s kv-cache consumption. *arXiv preprint arXiv:2407.18003*, 2024.
- [10] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.
- [11] 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.
- [12] David F Sherry and Daniel L Schacter. The evolution of multiple memory systems. *Psychological review*, 94(4):439, 1987.
- [13] James S Nairne and Josefa NS Pandeirada. Adaptive memory: Ancestral priorities and the mnemonic value of survival processing. *Cognitive psychology*, 61(1):1–22, 2010.
- [14] Paul W Frankland and Bruno Bontempi. The organization of recent and remote memories. *Nature reviews neuroscience*, 6(2):119–130, 2005.
- [15] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [16] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.
- [17] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, et al. Infinitebench: Extending long context evaluation beyond 100k tokens. *arXiv preprint arXiv:2402.13718*, 2024.
- [18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf.
- [19] Jont B Allen and Lawrence R Rabiner. A unified approach to short-time fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 1977.

- [20] Alan V Oppenheim. *Discrete-time signal processing*. Pearson Education India, 1999.
- [21] Fredric J Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.
- [22] Jin-Hee Han, Steven A Kushner, Adelaide P Yiu, Christy J Cole, Anna Matynia, Robert A Brown, Rachael L Neve, John F Guzowski, Alcino J Silva, and Sheena A Josselyn. Neuronal competition and selection during memory formation. *science*, 316(5823):457–460, 2007.
- [23] Nikolaus Hansen. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pages 75–102, 2006.
- [24] bloc97. NTK-Aware Scaled RoPE allows LLaMA models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation., 2023. URL https://www.reddit.com/r/LocalLLaMA/comments/141z7j5/ntkaware_scaled_rope_allows_llama_models_to_have/.
- [25] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.
- [26] Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective l_2 norm-based strategy for kv cache compression. *arXiv preprint arXiv:2406.11430*, 2024.
- [27] Yuanhan Zhang, Bo Li, haotian Liu, Yong jae Lee, Liangke Gui, Di Fu, Jiashi Feng, Ziwei Liu, and Chunyuan Li. Llava-next: A strong zero-shot video understanding model, April 2024. URL <https://llava-v1.github.io/blog/2024-04-30-llava-next-video/>.
- [28] Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. Longvideobench: A benchmark for long-context interleaved video-language understanding, 2024. URL <https://arxiv.org/abs/2407.15754>.
- [29] Junjie Zhou, Yan Shu, Bo Zhao, Boya Wu, Shitao Xiao, Xi Yang, Yongping Xiong, Bo Zhang, Tiejun Huang, and Zheng Liu. Mlvu: A comprehensive benchmark for multi-task long video understanding. *arXiv preprint arXiv:2406.04264*, 2024.
- [30] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.
- [31] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.
- [32] Chris Wendler, Veniamin Veselovsky, Giovanni Monea, and Robert West. Do llamas work in english? on the latent language of multilingual transformers. *arXiv preprint arXiv:2402.10588*, 2024.
- [33] Yao Yao, Zuchao Li, and Hai Zhao. Sirllm: Streaming infinite retentive llm. *arXiv preprint arXiv:2405.12528*, 2024.
- [34] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyriolidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024.
- [35] Matanel Oren, Michael Hassid, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.
- [36] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=uNrFpDPMyo>.
- [37] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.
- [38] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

- [39] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.
- [40] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. Llm maybe longlm: Self-extend llm context window without tuning. *arXiv preprint arXiv:2401.01325*, 2024.
- [41] 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.
- [42] Bo Li, Peiyuan Zhang, Kaichen Zhang, Fanyi Pu, Xinrun Du, Yuhao Dong, Haotian Liu, Yuanhan Zhang, Ge Zhang, Chunyuan Li, and Ziwei Liu. Lmms-eval: Accelerating the development of large multimodal models, March 2024. URL <https://github.com/EvolvingLMMs-Lab/lmms-eval>.
- [43] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [44] Edward Beeching and Thomas Simonini. Introducing decision transformers on hugging face, Mar 2022. URL <https://huggingface.co/blog/decision-transformers>.
- [45] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.
- [46] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.
- [47] William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. Reducing transformer key-value cache size with cross-layer attention. *arXiv preprint arXiv:2405.12981*, 2024.
- [48] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.
- [49] Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*, 2024.
- [50] Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache. *arXiv preprint arXiv:2403.04643*, 2024.
- [51] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaire Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghai Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

- [52] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.
- [53] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.
- [54] Jack Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Timothy Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. *Advances in Neural Information Processing Systems*, 29, 2016.
- [55] Guillaume Lample, Alexandre Sablayrolles, Marc’Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.
- [56] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [57] 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.
- [58] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.
- [59] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.
- [60] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.
- [61] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.
- [62] Dongseong Hwang, Weiran Wang, Zhuoyuan Huo, Khe Chai Sim, and Pedro Moreno Mengibar. Transformerfam: Feedback attention is working memory. *arXiv preprint arXiv:2404.09173*, 2024.
- [63] Yujin Tang and David Ha. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 22574–22587, 2021.
- [64] David So, Quoc Le, and Chen Liang. The evolved transformer. In *International conference on machine learning*, pages 5877–5886. PMLR, 2019.
- [65] Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. Evolutionary optimization of model merging recipes. *arXiv preprint arXiv:2403.13187*, 2024.
- [66] Greg Kamradt. Needleinahaystack: Doing simple retrieval from llm models at various context lengths to measure accuracy, 2024. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack/tree/main.
- [67] Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. Longalign: A recipe for long context alignment of large language models. *arXiv preprint arXiv:2401.18058*, 2024.

Table 9: NAMMs hyper-parameters used for training and evaluation. The omitted CMA-ES hyper-parameters can be obtained by following the recommended default calculation by Hansen [23].

NAMMs hyperparameters	
Spectrogram window size n_w	32
Spectrogram window stride s_w	16
Spectrogram window type	Hann
Spectrogram EMA reduction coefficient γ	0.99^{16}
Positional features	8
NAMMs execution delay	512
NAMMs non-linearity	ReLU
Optimization hyperparameters, notation from Hansen [23]	
Evolution algorithm	CMA-ES
Elite ratio	0.5
Mean coefficient c_m	1
Initial step size σ	0.65
Samples batch size per-task	64
Population size	32
Task for incremental stage 1	PassageRetrieval-en
Task for incremental stage 2	DuReader
Task for incremental stage 3	NarrativeQA
BAM-specific	
Hidden dimensions	16
Use bias	True
Masking strategy	counter-causal
Number of attention layers	1
Number of final linear layers	1
Use residual connections	True
Use multiplicative interactions	True
MLP-specific	
Hidden dimension	25
Number of hidden layers	2
Use residual connections	True

A Implementation details

A.1 Model specifics and NAMMs execution

We evolve our Neural Attention Memory Models on top of a context-extended Llama 3 8B [15] base model. In particular, we employ the NTK-aware positional interpolation strategy [24] to extend the context by four times from 8192 to 32768. Unlike prior strategies that require further gradient fine-tuning to avoid performance collapse [38], NTK-aware positional interpolation has been shown to produce sensible results even when applied zero-shot. In case the length of a task prompt still exceeds 32768 we perform mid-sentence cropping [39, 40], as standard in long-context LM evaluation [16, 17].

When applying NAMMs, we only affect the execution of the base model with a fixed frequency, once every $n_{up} = 512$ steps. When feeding longer prompts to our model, we simply split the tokens into n_{up} -sized chunks. We note that due to modern frameworks being bound primarily by memory constraints, input-splitting in itself has minimal effects on running time, with similar approaches being already performed under the hood by established kernel procedures [41].

A.2 Feature extraction and architecture details

Our new feature extraction framework is a key component for enabling the transfer properties of NAMMs. In practice, we extract the attention spectrogram from the real-valued attention matrix using a Hann window of size $n_w = 32$, resulting in just seventeen complex-values frequencies that we convert to real numbers by simply taking their magnitude, yielding each $\omega_i^t \in \mathbb{R}^{17}$. We use a stride of

half the window size $s_w = 16$, producing $n_T = n_{up}/s_w = 32$ frequency representations over the time axis of the attention matrix from the latest chunk of n_{up} queries, $\omega_i^1, \dots, \omega_i^{n_T}$. Thus, we reduce these frequency representations over the time axis via an element-wise exponentially moving average operation. We note that our EMA does not only consider the n_T representations computed for the frequency of each token in the n_{up} -sized chunk of the latest queries, but also the discounted EMA at the *previous execution step* or our memory for each retained token, denoted ω'_i . Thus, each of our reduced spectrogram representations reflects the full history of previous attention values:

$$\omega_i = \left(\sum_{t=1}^{n_T} \gamma^{t-1} \omega_i^t \right) + \gamma^{n_T} \omega'_i, \quad (4)$$

where we use γ to denote the EMA’s discount factor. To expedite learning the weights of our architecture, we ensure all spectrogram features have unit variance at initialization across our training data, using the statistics of the base Llama 3 model computed on the first task employed in incremental learning (PassageRetrieval). Finally, we also concatenate a small eight-dimensional sinusoidal positional embedding using the *oldness* of each token, i.e., the amounts of new queries observed since its introduction in the KV cache.

Our backward-attention memory network processes these representations by directly first applying the self-attention layer employing the counter-autoregressive backward masking introduced in Section 3, designed to facilitate asymmetric interactions between tokens in memory. The output of self-attention is then fed to a single final linear layer to obtain the final score. We employed a few important additional design choices following some preliminary testing. First, motivated by efficiency considerations, we use a single head within our attention mechanism and no layer normalization. Second, our attention layer produces outputs that are twice the dimensionality of the spectrogram features. These outputs are integrated back into the main network before the final linear layer via both residual and multiplicative interactions. We provide a schematic depiction of our minimal architecture in Figure 7. Through our minimalist design choices, our full network comprises only just over four thousand learnable parameters, a negligible amount, orders of magnitudes lower than even a single layer in modern transformers.

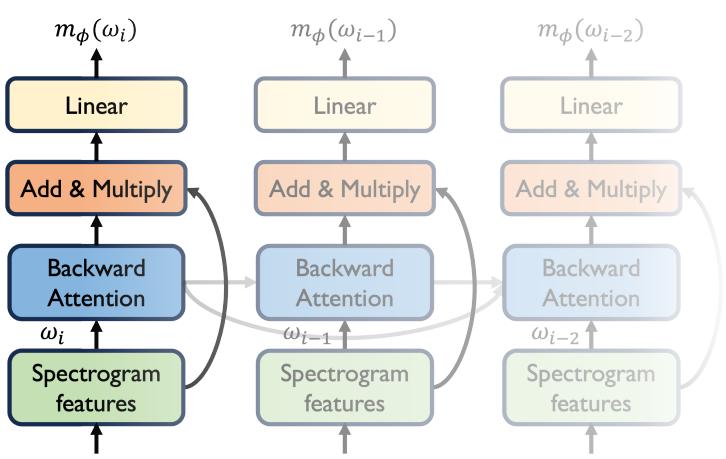


Figure 7: Schematic depiction of the components of our Neural Attention Memory Models, denoted m_ϕ , parameterized with our BAM architecture. The spectrogram representation of each token, denoted ω_i , is processed by an attention layer followed by a simple linear operation to output its relative score. Backward masking introduces asymmetry, ensuring that each token can only attend to its future relatives.

A.3 Zero-shot transfer

For our zero-shot transfer experiments, we consider a Llama 3 transformer with 70B parameters [15], a Llava Next Video transformer with 7B parameters [27], and a decision transformer [30] with about 1M parameters. For our 70B experiments, we follow the exact same setup as when evaluating our 7B Llama model used in training. For our video-language model, we extract 12×12 image tokens from 48 uniformly sampled frames, 6912 in total. We also slightly shift the selection score threshold by 5, to counteract the lower number of total tokens and get a comparable average cache size to the L2 and H2O baselines. We adapt the code and follow the standardized experimental setup from Li et al. [42]. For the reinforcement learning experiments, we encode each state, action, and return-to-go into separate tokens and do not apply any restrictions or modifications to our standard NAMM LM setup.

We average the performance collected over 20 random seeds to account for the stochasticity of the initial state in the Gym Mujoco environments [43]. Our RL experiments adapt the checkpoints and setup provided by Beeching and Simonini [44].

Table 10: Statistics of ChouBun. Lengths are counted by tokens produced by Llama 3 tokenizer.

Statistics	Extractive QA			Summarization		Overall
	JA.WikiQA	JA.EdinetQA	JA.CorpSecQA	JA.CorpSecSum	All tasks	
Number of documents	20	20	30	30	70	
Number of QA pairs	200	390	150	30	770	
Number of reference answers	1	1	1	5	1 or 5	
Document length max.	13027	10152	85981	85981	85981	
Document length mean	10131	8994	26220	26220	13317	
Document length min.	8196	6825	5640	5640	5640	
Answer length max.	40	208	30	140	208	
Answer length mean	7	11	8	80	21	
Answer length min.	1	1	1	55	1	

Table 11: Performance of a wider range of LLMs on the ChouBun benchmark.

Model/Task name	Extractive QA			Summarization		Overall	
	JA.WikiQA	JA.EdinetQA	JA.CorpSecQA	JA.CorpSecSum	All tasks	Max. length	
mistralai/Mistral-7B-v0.1	8.68	8.34	16.25	10.50	10.94	32768	
rinna/llama-3-youko-8b	16.68	12.23	17.03	22.27	17.05	8192	
meta-llama/Meta-Llama-3-8B	14.58	14.77	16.86	22.84	17.27	8192	
meta-llama/Llama-2-7b-hf	16.77	9.92	20.86	21.97	17.38	2048	
01-ai/yi-6b-200k	30.36	23.64	38.09	21.11	28.30	200000	
elyza/Llama-3-ELYZA-JP-8B	20.77	21.45	35.59	40.21	29.50	8192	

B ChouBun details

The ChouBun benchmark is created to assess the generalization ability of NAMMs to a new language (Japanese), but we hope it will also serve as a standard benchmark for Japanese LLMs. The benchmark is composed of two task categories — extractive QA and abstractive summarization — and four tasks as follows.

- *JA.WikiQA* is an extractive QA task about 20 randomly sampled articles from the 20240429 dump of Japanese Wikipedia⁴. Each article corresponds to 10 QA pairs, and there are 200 QA pairs in total.
- *JA.EdinetQA* is an extractive QA task based on 20 security reports from EDINET⁵. The EDINET security reports are in CSV format, which makes them less human-readable. Nevertheless, we choose not to convert the format because the conversion process per se is non-trivial, and using a CSV-style text input helps us evaluate a model’s capability of understanding structured data. The total number of QA pairs in *JA.EdinetQA* is 390.
- *JA.CorpSecQA* is another extractive QA task based on 30 security reports downloaded from three corporation websites (MUFG⁶, NTT⁷, and Toyota⁸). We extract texts from original file in PDF format. There are 150 QA pairs in total.
- *JA.CorpSecSum* is an abstractive summarization task based on the same data of *JA.CorpSecQA*. Each document corresponds to one data point, and we collect 5 reference summaries for each data point.

Collecting human annotations for long-text tasks is challenging, therefore we use synthetic QA pairs and summaries. In particular, we prompt various LLMs⁹ to generate multiple question-answer pairs or summaries for each document. Different instructions are designed for the two tasks and they are

⁴<https://dumps.wikimedia.org/other/cirrussearch/>

⁵<https://disclosure2.edinet-fsa.go.jp/>

⁶https://www.mufg.jp/ir/report/security_report/

⁷<https://group.ntt.jp/ir/library/results/>

⁸<https://global.toyota/jp/ir/library/securities-report/>

⁹gpt-4o-2024-05-13, gpt-4o-mini-2024-07-18, gpt-4-turbo-2024-04-09, and claude-3-5-sonnet-20240620

Prompt for extractive QA

```
抽出型の長文QAモデルのトレーニングデータを作成しています。  
コンテキストとして長い文書を提供します。  
文書を注意深く読み、分析し、20個の質問と回答のペアを生成してください。  
生成されたQAペアの要件は以下の通りです。\\n  
1. 回答は文書からのテキストの一部でなければなりません。\\n  
2. 回答は短く簡潔なテキストの一部であるべきです。\\n  
3. 質問は多様で、文書の異なる側面をカバーすべきです。\\n  
4. 回答は、直接文章の内容を引用してください。余計な情報は含めないでください。\\n  
以下の形式で20個の質問と回答のペアを直接回答してください:\\n  
### 質問 1 ###\\n  
{question_1}\\n  
### 回答 1 ###\\n  
{answer_1}\\n  
...\\n  
### 質問 20 ###\\n  
{question_20}\\n  
### 回答 20 ###\\n  
{answer_20}\\n  
文書は以下の通りです:\\n  
### 文書 ###\\n  
{doc_text}
```

Prompt for abstractive summarization

```
抽象的な長文要約モデルのトレーニングデータを作成しています。  
コンテキストとして長い文書を提供します。  
文書を注意深く読み、分析し、5個の要約例を生成してください。  
以下は生成される要約の要件です。\\n  
1. 各要約は短く簡潔であるべきです。\\n  
2. 各要約は文書の一般的なアイデア、トレンド、洞察を網羅すべきです。\\n  
3. すべての要約は内容が同一でありながら、表現が多様であるべきです。\\n  
以下の形式で5個の要約を直接返信してください:\\n  
### 要約 1 ###\\n  
{summary_1}\\n  
...\\n  
### 要約 5 ###\\n  
{summary_5}\\n  
文書は以下の通りです:\\n  
### 文書 ###\\n  
{doc_text}
```

Figure 8: LLM prompts for generating synthetic QA pairs and summaries in ChouBun.

shown in Figure 8. To improve the reliability of the synthetic data, we ensure that every answer in extractive QA tasks is a text span presented in its corresponding source document. In Table 10, we provide the statistics of the benchmark.

We use F1 score and ROUGE score for evaluation in the extractive QA tasks and summarization task, respectively. Reference text and hypothesis text are pre-tokenized by the MeCab tokenizer¹⁰. A wider range of LLMs' performance on the ChouBun benchmark is presented in Table 11.

¹⁰<https://github.com/polm/fugashi>

Table 12: NAMMs evaluation on LongBench [16]. The normalized performance (in brackets) is calculated using the base model with full cache. The aggregate test task performance of NAMMs models is taken by averaging the normalized scores on the tasks not used for incremental evolution.

Model/Task id	Single-Doc QA				Multi-Doc QA				Summarization			
	1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Base model	10.38 (1.00)	12.79 (1.00)	22.60 (1.00)	21.31 (1.00)	10.41 (1.00)	12.67 (1.00)	7.54 (1.00)	25.86 (1.00)	29.34 (1.00)	23.93 (1.00)	0.92 (1.00)	2.66 (1.00)
NAMM (MLP, s1)	7.60 (0.73)	12.74 (1.00)	22.74 (1.01)	21.08 (0.99)	9.58 (0.92)	12.24 (0.97)	6.48 (0.86)	19.41 (0.75)	27.76 (0.95)	23.61 (0.99)	0.95 (1.03)	3.44 (1.29)
NAMM (MLP, s2)	6.76 (0.65)	12.77 (1.00)	23.74 (1.05)	20.56 (0.96)	9.69 (0.93)	12.21 (0.96)	6.93 (0.92)	22.40 (0.87)	27.30 (0.93)	24.20 (1.01)	1.72 (1.87)	2.78 (1.05)
NAMM (BAM, s1)	5.77 (0.56)	12.76 (1.00)	22.94 (1.02)	21.55 (1.01)	9.47 (0.91)	12.21 (0.96)	6.51 (0.86)	18.73 (0.72)	28.06 (0.96)	23.97 (1.00)	1.01 (1.10)	4.00 (1.50)
NAMM (BAM, s2)	7.08 (0.68)	12.70 (0.99)	22.21 (0.98)	21.50 (1.01)	9.94 (0.95)	12.21 (0.96)	7.13 (0.95)	20.34 (0.79)	28.87 (0.98)	23.84 (1.00)	0.92 (1.00)	3.94 (1.48)
NAMM (BAM, s3)	9.14 (0.88)	12.63 (0.99)	21.94 (0.97)	21.34 (1.00)	9.71 (0.93)	11.63 (0.92)	6.98 (0.93)	20.58 (0.80)	28.78 (0.98)	24.39 (1.02)	1.04 (1.13)	3.63 (1.36)

Model/Task id	Few-shot Learning				Synthetic			Code		Overall		
	4-1	4-2	4-3	4-4	5-1	5-2	5-3	6-1	6-2	All tasks	Test tasks	Cache size
Base model	73.00 (1.00)	89.45 (1.00)	46.54 (1.00)	40.00 (1.00)	1.48 (1.00)	12.18 (1.00)	28.80 (1.00)	69.09 (1.00)	65.17 (1.00)	28.86 (1.00)	N/A	32768 (1.00)
NAMM (MLP, s1)	73.00 (1.00)	89.48 (1.00)	46.80 (1.01)	37.50 (0.94)	2.46 (1.66)	23.98 (1.97)	28.46 (0.99)	69.75 (1.01)	66.40 (1.02)	28.83 (1.05)	1.01	7639 (0.23)
NAMM (MLP, s2)	74.00 (1.01)	88.64 (0.99)	46.04 (0.99)	41.50 (1.04)	1.53 (1.03)	25.94 (2.13)	29.78 (1.03)	69.80 (1.01)	65.23 (1.00)	29.22 (1.07)	1.02	8475 (0.26)
NAMM (BAM, s1)	73.00 (1.00)	89.81 (1.00)	46.70 (1.00)	38.75 (0.97)	2.19 (1.48)	25.14 (2.06)	28.51 (0.99)	69.50 (1.01)	66.51 (1.02)	28.91 (1.05)	1.00	7951 (0.24)
NAMM (BAM, s2)	73.00 (1.00)	90.03 (1.01)	46.85 (1.01)	42.00 (1.05)	2.35 (1.59)	24.69 (2.03)	28.46 (0.99)	69.65 (1.01)	66.57 (1.02)	29.25 (1.07)	1.04	8267 (0.25)
NAMM (BAM, s3)	73.00 (1.00)	89.81 (1.00)	46.35 (1.00)	40.00 (1.00)	3.04 (2.05)	27.55 (2.26)	28.60 (0.99)	69.53 (1.01)	66.35 (1.02)	29.33 (1.11)	1.07	8155 (0.25)

Table 13: NAMMs evaluation on InfiniteBench [17]. The normalized overall performance (in brackets) is calculated using the average performance of the base model with full cache.

Model/Task name	Retrieval			Dialogue			Novel			Math		Code		Overall	
	Ret.PassKey	Ret.Number	Ret.KV	En.Dia	En.Sum	En.MC	En.QA	ZH.QA	Math.Find	Code.Run	Code.Debug	All tasks	Cache size		
Base model	0.00	0.00	0.00	1.00	7.73	0.00	1.05	1.79	0.00	0.00	0.00	1.05 (1.00)	32747 (1.00)		
NAMM (MLP, s1)	0.00	10.00	0.00	3.00	7.27	3.93	1.57	4.26	0.57	0.00	3.30	3.08 (2.93)	11329 (0.35)		
NAMM (MLP, s2)	10.17	11.86	0.00	2.50	7.48	3.06	1.58	4.10	1.71	0.00	1.52	4.00 (3.80)	13031 (0.40)		
NAMM (BAM, s1)	9.49	9.83	1.80	0.50	14.36	37.12	8.95	16.20	5.71	1.50	6.09	10.14 (9.63)	11173 (0.34)		
NAMM (BAM, s2)	11.86	11.86	1.80	1.00	14.62	35.37	8.96	15.45	0.57	1.75	4.31	9.78 (9.29)	12789 (0.39)		
NAMM (BAM, s3)	11.86	11.86	1.80	1.00	14.91	36.24	8.78	17.67	10.57	1.75	4.57	11.00 (10.45)	13192 (0.40)		

C Additional results

C.1 Performance across incremental stages and architectures

We provide additional results to the ones provided in Section 4, evaluating a fully-trained NAMM with our BAM architecture design. Here, we compare performance across different NAMMs, evaluating the best checkpoints after each stage of incremental training stage, and ablating the BAM architecture with an MLP.

Extended language modeling results. We report our results for LongBench, InfiniteBench, and ChouBun in Tables 12, 13, 14. First, we note that even training on a single task with our simple MLP architecture impressively improves performance across all benchmarks. Additionally, performance across benchmarks sees near-monotonic further improvements with each stage of our incremental evolution recipe. Comparing our implementations, we note that the performance benefits from the memory models with backward attention are consistently superior to the fully connected variant in both initial stages of incremental training, empirically validating our hypothesis about the importance of global KV cache information for determining the importance of each token. Lastly, on ChouBun, we observe that the performance with BAM sees a notable upswing after the second stage of incremental training, which might be associated with the introduction of another ideogram-based language in the training set.¹¹ The same improvement not occurring with the MLP-based NAMMs might be further evidence of architectural performance saturation, highlighting once again the effectiveness of our main implementation design.

Extended zero-shot transfer results. We report our extended zero-shot transfer results for the 70B model and the offline RL setting in Tables 16 and 15. We see the benefits from NAMMs again increase as we incorporate backward attention, and with each stage of incremental training to a similar extent as with the language modeling tasks. These results further highlight the potential benefits of scaling up the architecture of our memory model and increasing the number of incremental stages. To this end, given the generality of our parameterization, an interesting unexplored approach could

¹¹The DuReader task, used in the second stage of incremental training, uses the Chinese language.

Table 14: NAMMs evaluation on the new ChouBun benchmark. The normalized performance (in brackets) is calculated using the base model with full cache.

Model/Task name	Extractive QA			Summarization		Overall	
	JA.WikiQA	JA.EdinetQA	JA.CorpSecQA	JA.CorpSecSum	All tasks	Cache size	
Base model	22.91 (1.00)	28.34 (1.00)	11.83 (1.00)	21.75 (1.00)	21.21 (1.00)	12099 (1.00)	
NAMM (MLP, s1)	21.60 (0.94)	26.81 (0.95)	10.34 (0.87)	29.60 (1.36)	22.09 (1.04)	9525 (0.79)	
NAMM (MLP, s2)	20.76 (0.91)	26.30 (0.93)	11.86 (1.00)	29.32 (1.35)	22.06 (1.04)	9815 (0.81)	
NAMM (BAM, s1)	19.19 (0.84)	28.85 (1.02)	14.36 (1.21)	28.51 (1.31)	22.73 (1.07)	9569 (0.79)	
NAMM (BAM, s2)	20.75 (0.91)	28.46 (1.00)	14.55 (1.23)	32.45 (1.49)	24.05 (1.13)	9867 (0.82)	
NAMM (BAM, s3)	21.34 (0.93)	28.61 (1.01)	14.64 (1.24)	33.15 (1.52)	24.44 (1.15)	9895 (0.82)	

Table 15: NAMMs evaluation on D4RL [31] using a Decision Transformer model [30, 44]. The normalized overall performance (in brackets) is calculated using the average performance of the base model with full cache.

Model/Task name	Hopper-v3			Walker2d-v3			HalfCheetah-v3			Overall	
	Medium	Med-Replay	Expert	Medium	Med-Replay	Expert	Medium	Med-Replay	Expert	All tasks	Cache size
Base model	33.36 (1.00)	18.37 (1.00)	44.62 (1.00)	68.21 (1.00)	7.18 (1.00)	38.98 (1.00)	34.91 (1.00)	5.06 (1.00)	10.64 (1.00)	29.04 (1.00)	3000 (1.00)
NAMM (MLP, s1)	33.01 (0.99)	18.39 (1.00)	38.09 (0.85)	70.82 (1.04)	7.25 (1.01)	44.61 (1.14)	35.64 (1.02)	5.05 (1.00)	10.87 (1.02)	29.30 (1.01)	1993 (0.66)
NAMM (MLP, s2)	33.48 (1.00)	19.24 (1.05)	30.07 (0.67)	73.22 (1.07)	7.95 (1.11)	48.21 (1.24)	33.59 (0.96)	5.81 (1.15)	14.67 (1.38)	29.58 (1.02)	2834 (0.94)
NAMM (BAM, s1)	35.02 (1.05)	18.24 (0.99)	45.95 (1.03)	69.33 (1.02)	7.91 (1.10)	44.45 (1.14)	34.25 (0.98)	5.12 (1.01)	13.68 (1.29)	30.44 (1.05)	2009 (0.67)
NAMM (BAM, s2)	35.18 (1.05)	18.79 (1.02)	48.08 (1.08)	71.97 (1.06)	7.70 (1.07)	49.74 (1.28)	35.67 (1.02)	5.78 (1.14)	10.82 (1.02)	31.53 (1.09)	2534 (0.84)
NAMM (BAM, s3)	36.10 (1.08)	18.86 (1.03)	49.39 (1.11)	70.87 (1.04)	7.53 (1.05)	50.02 (1.28)	34.56 (0.99)	5.90 (1.17)	12.34 (1.16)	31.73 (1.09)	2434 (0.81)

be to incorporate different base models and input modalities during evolutionary training, something that would substantially increase problem diversity to obtain an even more robust transfer behavior.

C.2 Training curves with fully-connected NAMMs

In Figure 9, we provide training curves of our Neural Attention Memory Model using a simple MLP architecture rather than backward attention, evaluated in Section 4. In the left sub-plot, we show the average and standard deviation of the normalized batch performance across the population, while in the right sub-plot, we show the normalized per-task and average performance on all samples of the optimized mean from CMA-ES. When compared with the BAM training curve from Figure 4, we note a few interesting differences, although its evaluation performance on the full LongBench benchmark is lower across both incremental phases (see Table 2), both its population batch performance and the CMA-ES full-task performance on the training sets are either comparable or slightly higher than BAM’s. This dichotomy appears to indicate that cross-token interactions might provide a better inductive bias, mitigating the overfitting potential of NAMMs.

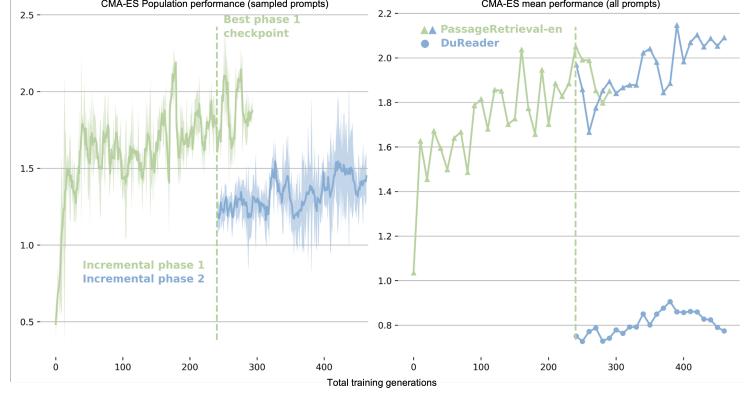


Figure 9: Mean and standard deviation over the CMA-ES population batch performance (left), together with the performance of the learned mean parameter on each task (right) for the training of the MLP NAMM.

Table 16: NAMMs evaluation on LongBench [16] with a Llama 3 70B model. The normalized performance (in brackets) is calculated using the base model with full cache. The aggregate test task performance of NAMMs models is taken by averaging the normalized scores on the tasks not used for incremental evolution. The tasks on which NAMMs are trained are highlighted with a gray background.

Model/Task id	Single-Doc QA				Multi-Doc QA				Summarization			
	1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Base model	9.38 (1.00)	13.84 (1.00)	24.99 (1.00)	17.78 (1.00)	11.73 (1.00)	14.26 (1.00)	8.11 (1.00)	26.43 (1.00)	13.13 (1.00)	24.55 (1.00)	23.20 (1.00)	10.08 (1.00)
NAMM (MLP, s1)	6.94 (0.74)	13.82 (1.00)	24.27 (0.97)	17.60 (0.99)	10.83 (0.92)	14.17 (0.99)	7.89 (0.97)	20.81 (0.79)	13.09 (1.00)	23.30 (0.95)	23.28 (1.00)	8.66 (0.86)
NAMM (MLP, s2)	7.88 (0.84)	13.71 (0.99)	23.27 (0.93)	18.18 (1.02)	11.41 (0.97)	14.11 (0.99)	8.07 (0.99)	21.75 (0.82)	14.28 (1.09)	24.48 (1.00)	22.00 (0.95)	8.99 (0.89)
NAMM (BAM, s1)	7.31 (0.78)	13.75 (0.99)	24.51 (0.98)	17.78 (1.00)	10.82 (0.92)	14.08 (0.99)	7.59 (0.94)	19.27 (0.73)	13.89 (1.06)	23.71 (0.97)	23.41 (1.01)	8.87 (0.88)
NAMM (BAM, s2)	3.57 (0.38)	13.86 (1.00)	23.02 (0.92)	18.71 (1.05)	4.94 (0.42)	13.32 (0.93)	1.90 (0.23)	17.74 (0.67)	10.39 (0.79)	20.45 (0.83)	23.18 (1.00)	8.13 (0.81)
NAMM (BAM, s3)	9.13 (0.97)	13.53 (0.98)	24.25 (0.97)	17.82 (1.00)	11.45 (0.98)	13.76 (0.96)	8.34 (1.03)	21.79 (0.82)	12.66 (0.96)	24.21 (0.99)	23.56 (1.02)	8.62 (0.86)

Model/Task id	Few-shot Learning				Synthetic			Code		Overall		
	4-1	4-2	4-3	4-4	5-1	5-2	5-3	6-1	6-2	All tasks	Test tasks	Cache size
Base model	78.00 (1.00)	92.43 (1.00)	48.67 (1.00)	45.50 (1.00)	22.50 (1.00)	75.37 (1.00)	33.89 (1.00)	74.60 (1.00)	71.19 (1.00)	35.22 (1.00)	N/A	10107 (1.00)
NAMM (MLP, s1)	78.00 (1.00)	92.28 (1.00)	48.37 (0.99)	43.50 (0.96)	20.76 (0.92)	68.66 (0.91)	33.89 (1.00)	74.58 (1.00)	71.68 (1.01)	34.11 (0.97)	0.99	7930 (0.78)
NAMM (MLP, s2)	77.00 (0.99)	91.93 (0.99)	48.60 (1.00)	44.75 (0.98)	17.17 (0.76)	70.21 (0.93)	36.18 (1.07)	74.72 (1.07)	71.30 (1.00)	34.29 (0.97)	0.99	8445 (0.84)
NAMM (BAM, s1)	77.50 (0.99)	92.46 (1.00)	48.24 (0.99)	45.00 (0.99)	17.32 (0.77)	69.87 (0.93)	33.89 (1.00)	74.58 (1.00)	72.40 (1.02)	34.11 (0.97)	0.99	7947 (0.79)
NAMM (BAM, s2)	74.50 (0.96)	51.45 (0.56)	39.73 (0.82)	15.00 (0.33)	5.86 (0.26)	13.35 (0.18)	34.29 (1.00)	73.81 (0.99)	61.91 (0.87)	25.20 (0.72)	0.79	8276 (0.82)
NAMM (BAM, s3)	78.50 (1.01)	92.36 (1.00)	48.49 (1.00)	45.50 (1.00)	19.07 (0.85)	74.19 (0.98)	34.28 (1.01)	74.71 (1.00)	72.42 (1.02)	34.70 (0.99)	0.99	8365 (0.83)

C.3 Evolution of memory size during training

In Figure 10, we provide training curves for the evolution of the memory size collected at the end of each task prompt of our NAMMs. On the left and right subplots, we provide results for the BAM and MLP implementations, respectively. For both architectures, we find that the memory size generally increases with training. This result suggests that NAMMs might learn to recognize additional valuable tokens as training progresses, enabling the corresponding performance improvements on the training tasks. Hence, they might indicate that there is some degree of a trade-off between the efficiency and performance of NAMMs. However, we note that both models are trained only for performance maximization, without any incentive to be more conservative. To this end, exploring regularization strategies to make NAMMs aware of deployment costs is an interesting direction for future work to obtain tailored sweet spots to cater to instance-specific resource constraints.

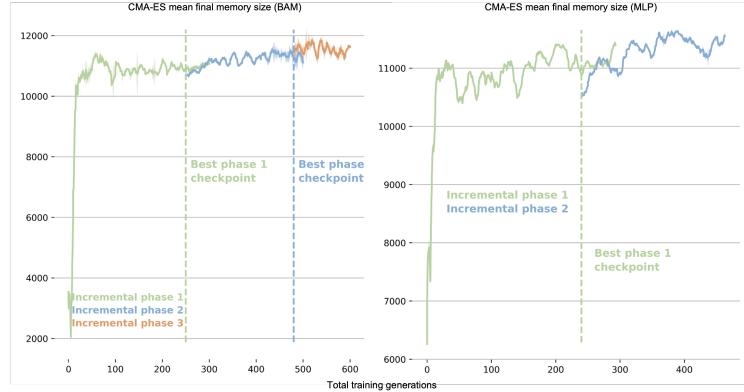


Figure 10: Final memory size of NAMM parameterized by the learned mean of CMA-ES for both the BAM (left) and the MLP implementations (right).

C.4 Incremental training ablation

We provide a full set of ablations results for our incremental training strategy, training a Neural Attention Memory Model with the BAM architecture from scratch on both the PassageRetrieval-en and DuReader tasks, as employed during the second stage of incremental learning. We evolve this Neural Attention Memory Model for 360 consecutive generations and provide training curves in Figure 11. In the left sub-plot, we show the average and standard deviation of the normalized batch performance across the population, in the center sub-plot, we show the normalized per-task and average performance on all samples of the optimized mean from CMA-ES, and on the right subplot we show the corresponding memory size. Furthermore, in Table 17, we provide the full LongBench evaluation results for this baseline, also showing our original incremental model’s performance for ease of comparison. Interestingly, the non-incremental NAMM obtained a notably higher score on the training tasks with a normalized performance of 1.57, in contrast to the normalized performance

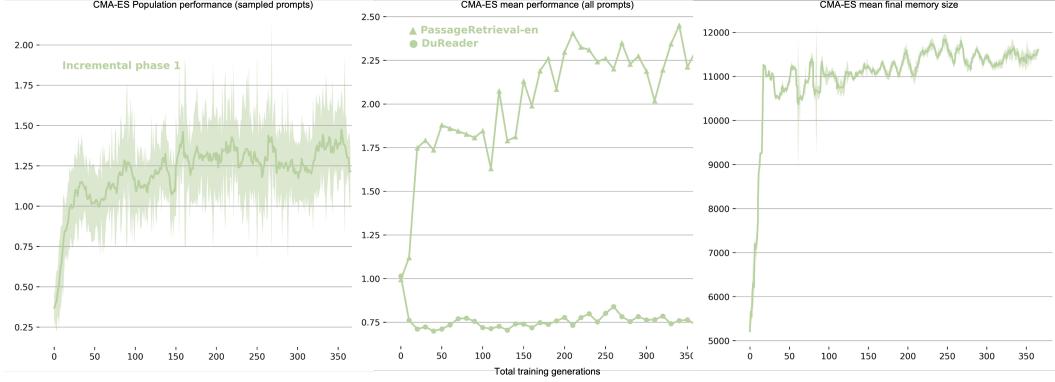


Figure 11: Mean and standard deviation over the CMA-ES population batch performance (left), together with the performance of the learned mean parameter on each task (center) and its final memory size for the NAMM trained without incremental evolution.

Table 17: NAMMs incremental learning (IL) ablation evaluation on LongBench [16]. The *No IL* baseline is trained from scratch on both the PassageRetrieval-en and DuReader tasks, the same employed during the second stage of incremental learning.

Model/Task id	Single-Doc QA				Multi-Doc QA				Summarization			
	1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Base model	10.38 (1.00)	12.79 (1.00)	22.60 (1.00)	21.31 (1.00)	10.41 (1.00)	12.67 (1.00)	7.54 (1.00)	25.86 (1.00)	29.34 (1.00)	23.93 (1.00)	0.92 (1.00)	2.66 (1.00)
NAMM (BAM, s1)	5.77 (0.56)	12.76 (1.00)	22.94 (1.02)	21.55 (1.01)	9.47 (0.91)	12.21 (0.96)	6.51 (0.86)	18.73 (0.72)	28.06 (0.96)	23.97 (1.00)	1.01 (1.10)	4.00 (1.50)
NAMM (BAM, s2)	7.08 (0.68)	12.70 (0.99)	22.21 (0.98)	21.50 (1.01)	9.94 (0.95)	12.21 (0.96)	7.13 (0.95)	20.34 (0.79)	28.87 (0.98)	23.84 (1.00)	0.92 (1.00)	3.94 (1.48)
NAMM (BAM, s3)	9.14 (0.88)	12.63 (0.99)	21.94 (0.97)	21.34 (1.00)	9.71 (0.93)	11.63 (0.92)	6.98 (0.93)	20.58 (0.80)	28.78 (0.98)	24.39 (1.02)	1.04 (1.13)	3.63 (1.36)
NAMM (BAM, no IL)	6.46 (0.62)	12.72 (0.99)	22.87 (1.01)	21.22 (1.00)	9.91 (0.95)	11.77 (0.93)	5.61 (0.74)	18.94 (0.73)	27.63 (0.94)	22.60 (0.94)	0.91 (0.99)	1.75 (0.66)

Model/Task id	Few-shot Learning				Synthetic				Code		Overall	
	4-1	4-2	4-3	4-4	5-1	5-2	5-3	6-1	6-2	All tasks	Test tasks	Cache size
Base model	73.00 (1.00)	89.45 (1.00)	46.54 (1.00)	40.00 (1.00)	1.48 (1.00)	12.18 (1.00)	28.80 (1.00)	69.09 (1.00)	65.17 (1.00)	28.86 (1.00)	N/A	10107
NAMM (BAM, s1)	73.00 (1.00)	89.81 (1.00)	46.70 (1.00)	38.75 (0.97)	2.19 (1.48)	25.14 (0.06)	28.51 (0.99)	69.50 (1.01)	66.51 (1.02)	28.91 (1.05)	1.00	8205
NAMM (BAM, s2)	73.00 (1.00)	90.03 (1.01)	46.85 (1.00)	42.00 (1.05)	2.35 (1.59)	24.69 (2.03)	28.46 (0.99)	69.65 (1.01)	66.57 (1.02)	29.25 (1.07)	1.04	8521
NAMM (BAM, s3)	73.00 (1.00)	89.81 (1.00)	46.35 (1.00)	40.00 (1.00)	3.04 (2.05)	27.55 (2.26)	28.60 (0.99)	69.53 (1.01)	66.35 (1.02)	29.33 (1.11)	1.07	8409
NAMM (BAM, no IL)	73.00 (1.00)	89.28 (1.00)	46.43 (1.00)	38.75 (0.97)	2.49 (1.68)	29.28 (2.40)	28.46 (0.99)	69.80 (1.01)	64.77 (0.99)	28.79 (1.03)	0.98	8457

of 1.41 achieved by the best checkpoint from the second incremental training stage. Yet, outside the PassageRetrieval-en and DuReader tasks, its performance is notably inferior and very close to the original performance of the base model. These results appear to indicate that the usefulness of incremental training goes beyond the faster evolution provided by reducing the number of evaluation prompts to assess performance and that this strategy plays an important role in regularizing evolution and making Neural Attention Memory Models effectively generalize to new tasks.

	Layer #15, Head #1	Layer #24, Head #6
PassageRetrieval-en (Natural language)	<pre>< begin_of_text > Here are 30 paragraphs from Wikipedia, along with an abstract. Please determine which paragraph the abstract is from. [...]</pre> <p>Paragraph 4: At the age of 21, she returned to France, and was soon the declared mistress of Louis Henri, Duke of Bourbon ("Monsieur le Due")-who was prime minister at the beginning of the reign of Louis XV (1723-1726). During his ministry she dominated the royal court and engineered the marriage of Louis XV of France to Marie Leszczynska instead of Mademoiselle de Vermandois, the younger sister of the Duke of Bourbon. In 1725, her scheme to have Bourbon's rival Fleury exiled failed. Instead, Fleury was recalled and in turn became prime minister causing Bourbon to be banished to his castle at Chantilly. [...]</p>	<pre>< begin_of_text > Here are 30 paragraphs from Wikipedia, along with an abstract. Please determine which paragraph the abstract is from. [...]</pre> <p>Paragraph 4: At the age of 21, she returned to France, and was soon the declared mistress of Louis Henri, Duke of Bourbon ("Monsieur le Due")-who was prime minister at the beginning of the reign of Louis XV (1723-1726). During his ministry she dominated the royal court and engineered the marriage of Louis XV of France to Marie Leszczynska instead of Mademoiselle de Vermandois, the younger sister of the Duke of Bourbon. In 1725, her scheme to have Bourbon's rival Fleury exiled failed. Instead, Fleury was recalled and in turn became prime minister causing Bourbon to be banished to his castle at Chantilly. [...]</p>
	<pre>< begin_of_text >Please complete the code given below. src/kademlia/operation/KadRefreshOperation.java public class KadRefreshOperation implements Operation { [...] import kademlia.util.serializer.JsonDHTSerializer; import kademlia.util.serializer.JsonRoutingTableSerializer; import kademlia.util.serializer.JsonSerializer; /** * The main Kademlia Node on the network, this node manages * everything for this local system. * * @author Joshua Kissoon [...]</pre>	<pre>< begin_of_text >Please complete the code given below. src/kademlia/operation/KadRefreshOperation.java public class KadRefreshOperation implements Operation { [...] import kademlia.util.serializer.JsonDHTSerializer; import kademlia.util.serializer.JsonRoutingTableSerializer; import kademlia.util.serializer.JsonSerializer; /** * The main Kademlia Node on the network, this node manages * everything for this local system. * * @author Joshua Kissoon [...]</pre>

Figure 12: Qualitative inspection of the text from decoding the retained tokens in the KV cache after applying NAMMs. We compare the behavior of NAMMs across the layers with the highest (left) and lowest average retained tokens (15 and 24), for tokens from either a natural language (top) or coding task (bottom).

D Additional analysis

D.1 Selected qualitative examples

We analyze the effects of layer depth and task structure on the kinds of tokens being forgotten by NAMMs by inspecting a few selected prompts. In particular, we compare the layers with the highest and lowest average retained tokens (15 and 24), for tokens from either a natural language or coding task (PassageRetrieval-en and RepoBench-P). As exemplified in Figure 12, we see that for early-middle layers, NAMMs tend to focus on retaining global information such as the task preamble and key words throughout the text. Instead, for later layers, NAMMs seem to learn to forget many of these tokens, whose information has likely been already incorporated into the other higher-level latents by the previous layers, allowing the transformer to focus more on tokens associated with more detailed local information. Furthermore, we find that, in coding tasks, the pruned tokens are mostly contiguous, corresponding to whitespace, comments, and whole segments of boilerplate code. This is in contrast with the behavior we observed for natural language, where NAMMs often appear trying to exploit some of the grammatical redundancies of the English syntax often dropping specific tokens mid-sentences.

D.2 Sensitivity to attention frequencies and positional encodings

We analyze the magnitudes of the gradients of the token scores s_i with respect to each dimension in the token feature vectors. This procedure quantifies how varying each dimension in our attention spectrogram representation locally affects the output score of NAMMs, thus, providing a heuristic measure of its relevance (since scores determine which tokens get discarded). In Figure 13, we plot the distribution of magnitudes for all the seventeen features up to the Nyquist frequency (0 to 16) in the attention spectrogram. All frequency distributions seem to cover a wide range of values, with each mean being close to the global mean, seemingly indicating NAMMs learn to make use of all available spectrogram information for at least some of the tokens. Additionally, we note that many of the higher frequencies have distributions with higher means and larger tails than the ‘ground frequency’ at dimension 0. Furthermore, as shown in the rightmost-lower subplot, NAMMs appear visibly less sensitive to recency information provided by the concatenated positional embeddings, with a lower total influence than frequency information on token scores. Overall, these observations seem to further validate the importance of going beyond simple hand-designed methods solely based on token recency and the sum of the attention values, which has so far been considered a strong established recipe for KV cache management [25, 26, 35, 36].

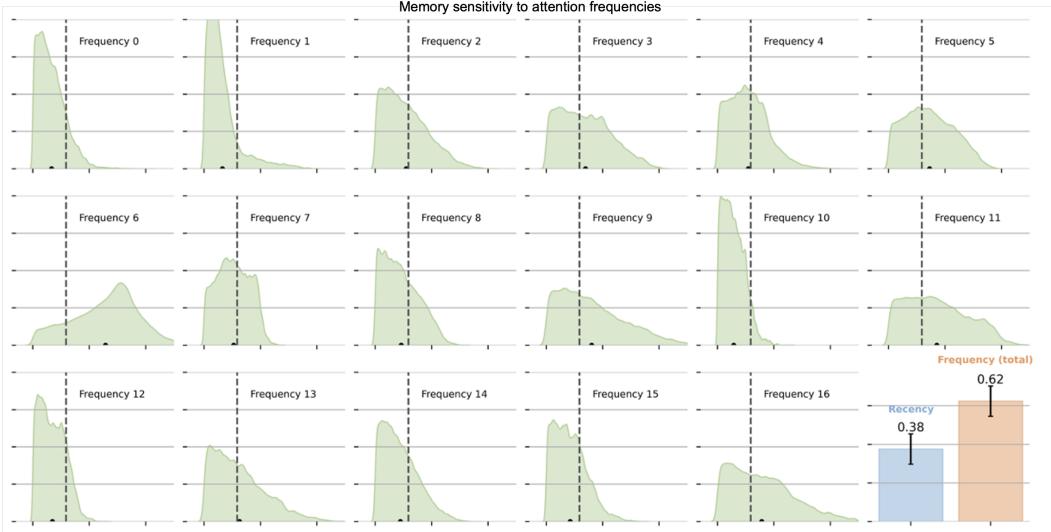


Figure 13: Distribution of gradient magnitudes for the token scores with respect to all the seventeen features in our attention spectrogram representations. In the rightmost-lower subplot, we also compare the total magnitudes of the frequency information with the recency information in the positional embeddings.

D.3 InfiniteBench results comparison

On the InfiniteBench tasks, our NAMM achieve particularly outstanding improvements over the base model and other baselines, with an over ten-fold score increase (from 1.05% to 11%). However, we note that even with NAMMs, the performance of Llama 3 8B still lags considerably behind the performance of powerful LMs designed specifically for long-context problems, as reported in Zhang et al. [17]. Nonetheless, on the En.Sum task, concerned with the summarization of fictitious novels, we find our main NAMM brings the performance of the context-extended Llama 3 from 7.73 to 14.91 even slightly beyond GPT4’s (14.73). While this performance is still low in absolute terms¹², such a result appears quite notable and suggests that improvements from NAMMs are orthogonal in nature to the ones brought by architectural improvements and scaling, which, by themselves, might be insufficient to address the challenges brought by long and noisy contexts.

We qualitatively inspect the effects of NAMMs on En.Sum by comparing example answers generated by Llama 3 with and without our memory models, together with examples generated by GPT4. As illustrated in Figure 14, we find both the Llama and GPT models to incur several failure modes, producing answers that entirely miss the objective of the original task. For instance, the context-extended Llama 3 often gets stuck in generation loops continuously repeating part of sentences without coherent structure. Instead, the GPT answers appear to forego summarizing the text and rather attempt to continue the provided passage, by generating end-of-text tokens or even roleplaying some of the characters. However, while introducing NAMMs appears to avoid many instances of these failure modes, we find the summarization of the memory-augmented Llama 3 still displays many imperfections such as misspelling character names (left) or lacking much depth by being extremely concise (right).

¹²InfiniteBench tasks are scored in a range between 0 and 100.

Figure 14: Qualitative examples comparing the ground produced responses by Llama3 with and without our NAMM memory, together with GPT4, on two prompts from the En.Sum task part of InfiniteBench.

E Extended related works

Similar to our NAMMs implementation, memory management through token eviction has been explored mostly to reduce memory constraints and enable querying LMs with longer contexts [9]. Commonly, strategies entail simply cropping input prompts to a shorter length, often more effective when done from the middle rather than the ends [39, 40]. More advanced, several heuristic strategies have been proposed to identify and evict the least important tokens in the KV cache, selectively pruning it to a fixed size for each layer. These strategies assess token relevance using metrics like L2 magnitude [26] or entropy [33], or analyze statistics from the attention matrix, such as value magnitude or cumulative sums [25, 34, 35]. Building on these ideas, Ge et al. [36] and Li et al. [37] apply multiple strategies simultaneously, choosing the best fit for each layer by matching them with specific attention patterns. However, unlike previous work, our approach uniquely employs a black-box model to *learn* KV cache management, aiming to enhance efficiency and boost performance.

Many other methods to reduce memory consumption, affecting the KV cache, are mostly orthogonal and likely complementary to our approach. For instance, MQA [45] and GQA [46] propose merging different attention heads during the training of LLMs, either fully or partially, to improve deployment-time throughput. Brandon et al. [47], pushed these strategies further, attempting to merge heads even across different layers. GQA is commonly employed in many modern LMs, including the LLama 3 family of models which we use to train and evaluate NAMMs on language tasks [15]. Furthermore, several methods have looked at KV cache compression through either quantization of the keys and values [48–50] or even the whole hidden states [51]. Similarly to the aforementioned prior work concerning KV cache pruning, these methods considered mainly hand-designed strategies, such as employing different quantization rates based on heuristically recognizing important tokens. We note that using evolution to optimize for which channels to merge or compress could also yield new interesting unexplored approaches, combining these orthogonal directions with some of the principles introduced by NAMMs.

There has also been much research interest in exploring new architectures to explicitly model components of a memory system or to address key challenges of reasoning over longer contexts. For instance, past work has looked at incorporating neural models of memory within neural networks by implementing different reading and writing operations - either directly replacing their layers [52, 53], or introducing new auxiliary components [54, 55]. In relation to transformers, more recent

works have been proposed rethinking the ingredients of the self-attention operation, mostly in the context of LMs. These works looked at either efficient linear approximation to self-attention to overcome quadratic costs [56–59], or introducing new kinds of persistent tokens and storage to extend information propagation [60–62]. However, as also noted by Dao et al. [41], none of these methods and approximations have managed to replace standard approaches so far. We take a different approach that can be integrated in a zero-shot manner even without any fine-tuning.

Lastly, methodologically related to NAMMs, there have been other prior methods making use of evolution for or with transformer models. For example, Tang and Ha [63] also trained a small attention-based model through evolution, exploiting the inherent parameter efficiency behind these operations. Furthermore, So et al. [64] proposed using evolution to meta-optimize the basic building of transformers via neural architecture search, while Akiba et al. [65] focused on evolving different merging strategies across layers belonging to LMs with different capabilities. As for these works, we note that evolution plays a critical role for NAMMs, allowing us to directly optimize for target performance and overcome the inherent non-differentiability underlying our new framework.

Table 18: NAMMs evaluation on the canonical *Needle In A Haystack* task [66]. The normalized performance (in brackets) is calculated using the base model with full cache.

Model/Task name	Needle prompt length			Overall	
	0-10000	10001-20000	20001+	All prompt lengths	Cache size
Base model (full cache)	8.87 (1.00)	7.53 (1.00)	3.50 (1.00)	6.32 (1.00)	32768
NAMM (BAM)	9.00 (1.02)	4.80 (0.64)	3.05 (0.87)	5.36 (0.85)	10208
NAMM (BAM), $\gamma = 0.9999^{s_w}$	9.00 (1.02)	5.33 (0.71)	3.45 (0.99)	5.68 (0.90)	10347

F Limitations and future extensions

F.1 Exploring the design space of Neural Attention Memory Models

In this work, we introduced Neural Attention Memory Models and showed their efficacy and potential to improve the performance and efficiency of transformers, even when evaluated zero-shot for unseen architectures and domains. However, given the novelty of our framework, we note that our design choices were mostly motivated by simplicity and practicality rather than quantitative empirical evidence. Thus, there is an extremely large design space in terms of the implementation, training, and deployment of these models that should be explored beyond this work, which is likely to yield further improvements.

For instance, while our current feature extraction, based on computing the spectrogram of the attention matrix, enables capturing global frequency information about the attention values of each token, it might fall short of modeling local information with enough granularity. This hypothesized limitation inherently comes from a few design choices we made with the purpose of limiting the input size and corresponding parameter count of our memory models. In particular, our spectrogram features only consider the real components of a short-time Fourier transform with a small Hann window of size thirty-two. Thus, we only provide NAMMs information about a relatively limited number of thirty-two frequencies, losing any notion of the phase of the attention matrix that would be captured by the full complex-valued Fourier coefficients. Consequently, the representations of tokens with high attention values for entirely non-overlapping queries occurring with the same frequency would be indistinguishable to our models. Moreover, our exponentially moving average reduction over the time dimension of the spectrograms provides an additional layer of heavy compression inevitably trading off expressivity for simplicity.

To partially address these concerns, an alternative design we explored entailed delaying the initial element-wise exponentially moving average reduction. Concretely, this involved computing T different scores, feeding m_ϕ all feature vectors ω_i^t for $t = 1, 2, \dots, T$, across the attention spectrogram’s compressed time axis, only then reducing the resulting scores $s_i^{1:T}$ via EMA. While, in principle, this alternative ordering would allow for additional expressivity without adding to the parameter count, in practice, when evaluated with an initial version of the simple 2-layer MLP model, we found no significant performance difference and opted for the former lighter option. However, introducing cross-token interactions with the improved BAM design and further scaling is likely to introduce a need of re-evaluating this choice.

One further limitation comes from the current reliance on the exact values of the attention matrix. This reliance precludes NAMMs training from making use of fast kernel algorithms developed to accelerate inference by foregoing materializing attention values [41]. While the main focus of this work has been to introduce NAMMs and display its potential to improve transformers across different domains, more scalable parameterizations and efficient backend integrations remain exciting open challenges for future research.

F.2 Improving long-context sparse retrievals

One notable example exemplifying some of the aforementioned limitations, comes from the canonical *Needle In A Haystack* task [66], which has been used to qualitatively evaluate LLMs for their ability to remember sparse information over long *noisy* horizons. We provide results on this task using the best-performing NAMM after three stages of incremental training with the BAM architecture, averaging evaluation scores provided by a GPT-4 model [1] across different prompt ranges, consistently with Bai

et al. [67]. As shown in Table 18, while NAMMs do not manage to exceed the overall performance of the base model, they still provide some notable efficiency gains. However, looking more closely at the score distribution across different prompt length ranges we observe an unexpected trend that is in contrast with the rest of our results on other benchmarks. In particular, while our NAMM obtains slightly higher than the base model for prompts with a size less than 10000, it seems to increasingly struggle with longer prompts.

After comparing the spectrogram features extracted for the different prompts, our explanation for these results highlights one current failure mode of the current implementation. In particular, the Needle In a Haystack task is constructed such that the model is tasked to remember some important information introduced at the beginning of the prompt, and later followed by completely unrelated ‘filler’ text. Hence, the attention scores and the corresponding spectrogram features for the tokens containing the relevant information are forcibly sparse, being high only at the very beginning of the prompt. Yet, since the evaluated NAMM reduces these features over the time axis of the spectrogram with an EMA coefficient of $\gamma = 0.99^{sw}$, all the frequency information regarding these tokens will be inevitably overwritten. To empirically validate our theory we provide results simply raising the EMA coefficient from $\gamma = 0.99^{sw}$ to $\gamma = 0.9999^{sw}$. Since our NAMMs was never actually trained with this higher coefficient, we note that this change effectively brings the input features out-of-distribution. Nonetheless, as shown in the final row of Table 18, the larger coefficient still manages to improve performance on the longer prompts by enabling the preservation of the frequency components from the target ‘needle’ over a longer horizon. These findings suggest that future NAMM designs should consider higher EMA reduction coefficients or, potentially, even directly *learning* this parameter with evolution in addition to the NAMM’s network weights.