Neurocache: Efficient Vector Retrieval for Long-range Language Modeling

Ali Safaya
asafaya19@ku.edu.tr

Deniz Yuret
dyuret@ku.edu.tr

KUIS AI Center
Computer Engineering Department
Koç University

Abstract

This paper introduces Neurocache, an approach to extend the effective context size of large language models (LLMs) using an external vector cache to store its past states. Like recent vector retrieval approaches, Neurocache uses an efficient k-nearest-neighbor (kNN) algorithm to retrieve relevant past states and incorporate them into the attention process. Neurocache improves upon previous methods by (1) storing compressed states, which reduces cache size; (2) performing a single retrieval operation per token which increases inference speed; and (3) extending the retrieval window to neighboring states, which improves both language modeling and downstream task accuracy. Our experiments show the effectiveness of Neurocache both for models trained from scratch and for pre-trained models such as Llama2-7B and Mistral-7B when enhanced with the cache mechanism. We also compare Neurocache with text retrieval methods and show improvements in single-document question-answering and few-shot learning tasks. We made the source code available under: https://github.com/alisafaya/neurocache

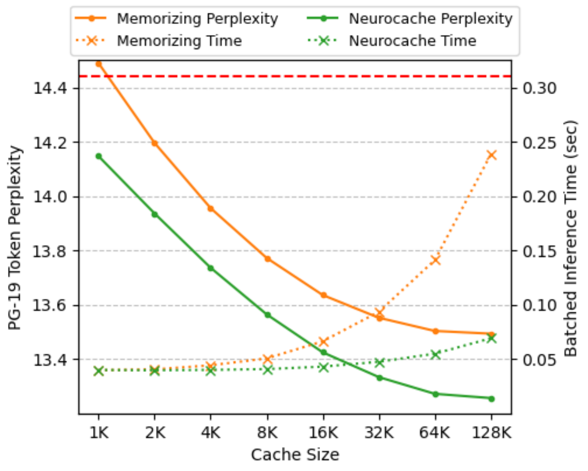

Figure 1: Performance and Scalability of Neurocache vs. Memorizing Transformers (Wu et al., 2022) on PG-19: The graph illustrates Neurocache's consistently lower token perplexity and faster inference times across various cache sizes on the Project Gutenberg-19 dataset, demonstrating its efficiency and scalability.

1 Introduction

Recent advancements in natural language processing have been significantly driven by the development of large language models (LLMs) such as GPT-3, GPT-4, Llama, and Llama2 (Brown et al., 2020; OpenAI, 2023; Touvron et al., 2023a,b). While demonstrating impressive capabilities, these models are constrained by limited context window sizes. This limitation becomes apparent in tasks that require understanding long documents, such as document summarization and academic literature review, where processing hundreds of thousands of tokens is necessary.

Various methods, including sparse attention (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020), have been explored to address this limitation. However, these approaches often struggle to utilize their extended contexts (Liu et al., 2023) fully. Recent research by Xu et al. (2023) shows that retrieval-augmented models with shorter contexts (4K tokens) can match the performance of models with longer contexts (16K/32K tokens), maintaining efficiency during inference. This emphasizes the potential of retrieval-augmented strategies in LLMs.

In response to these challenges, we introduce Neurocache. Neurocache employs an efficient k-nearest-neighbor (kNN) strategy for retrieving relevant past states from a compressed external vector cache. This approach is designed to optimize hidden state caching and retrieval, thereby enhancing language modeling quality and increasing inference speed.

Neurocache advances over exiting methods by reducing the cache size through the storage of compressed states, performing a single retrieval operation per token to boost inference speed, and extending the retrieval window to include neighboring states for improved language modeling and downstream accuracy. Figure 1 illustrates the advantages of Neurocache in terms of inference speed and language modeling accuracy over methods like Memorizing Transformers (Wu et al., 2022).

Our evaluation of Neurocache encompasses both models trained from scratch and established pretrained models such as Llama2-7B and Mistral-7B (Touvron et al., 2023b; Jiang et al., 2023), demonstrating its effectiveness in enhancing language models for downstream tasks. Specifically, we highlight Neurocache's improvements in single-document question-answering and few-shot learning tasks when compared to traditional text retrieval methods. Moreover, Neurocache's integration extends the maximum context length of these models to 128K tokens, indicating its significant impact on long-document processing.

In summary, Neurocache represents a substantial step forward in addressing the challenges of processing long documents in LLMs, offering a blend of efficiency, adaptability, and enhanced performance. Our comprehensive experiments and analysis showcase Neurocache's potential in revolutionizing the understanding of long documents in natural language processing.

2 Related Work

Transformers have made significant advancements in natural language processing but face challenges in processing long contexts. Various methods have been developed to extend the context window while maintaining computational efficiency (Huang et al., 2023).

Recent methods include the continued training or fine-tuning of short-context language models (Nijkamp et al., 2023; Chen et al., 2023b), positional interpolation (Chen et al., 2023a), ALiBi (Press et al., 2022), and sparse and efficient attention designs (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). These approaches reflect the evolving landscape of solutions for managing extended attention windows in large language models (LLMs).

However, language models still encounter difficulties in processing longer contexts (Liu et al., 2023). Studies have indicated that retrieval-augmented models with shorter contexts (4K) can surpass models with longer contexts (16K/32K) in performance (Xu et al., 2023).

Prominent strategies in this area are Text Retrieval and Vector Retrieval. Text Retrieval involves identifying and processing the most relevant segments of long documents. Vector Retrieval, on the other hand, integrates relevant hidden representations of the input, like hidden states or key-value pairs, into the model.

2.1 Text Retrieval

Text retrieval methods focus on processing relevant segments of long documents. Integrating retrieval mechanisms into language models, such as REALM (Guu et al., 2020), DPR (Karpukhin et al., 2020), RETRO (Borgeaud et al., 2021), and RALM (Ram et al., 2023), has enhanced model performance in various tasks.

A limitation of Text Retrieval is its dependency on external retrievers for identifying relevant segments of the context, often employing algorithms like BM25 (Robertson and Zaragoza, 2009), Contriever (Izacard et al., 2022), and others (Borgeaud et al., 2021; Ram et al., 2023; Karpukhin et al., 2020).

2.2 Vector Retrieval

Vector retrieval methods extend the context window by incorporating relevant hidden states from an external cache of past inputs' representations.

Memorizing Transformers present a novel adaptation to the traditional transformer decoder structure for handling lengthy documents. They process documents in smaller segments and use a dynamically updated external cache to track previous key-value pairs. These models employ an approximate k-nearest-neighbor (kNN) lookup over this cache, merging dense self-attention on the current context with external-attention over retrieved key-value pairs, thus effectively extending the context length (Wu et al., 2022).

Unlimiformer is a vector retrieval method, particularly suited for sequence-to-sequence models like BART (Lewis et al., 2020). It extends encoding length by using a kNN index over all input token hidden states, focusing on the top-k input tokens through kNN distance-based attention scores in each decoder layer's cross-attention head (Bertsch et al., 2023).

2.3 Neurocache

Neurocache is a vector retrieval method designed for processing long documents in large language models (LLMs). It employs a kNN strategy to efficiently retrieve compressed past states from an external vector cache. This approach contrasts with methods like Memorizing Transformers and Unlimiformer, particularly in terms of computational efficiency and cache size management.

Neurocache's notable features include storing compressed states to reduce cache size and performing a single retrieval operation per token, which accelerates inference speed. Additionally, it expands the retrieval window to include neighboring states, enhancing language modeling and downstream task performance.

Crucially, Neurocache shows adaptability with established pre-trained models like Llama2-7B and Mistral-7B, extending their maximum context length capabilities to 128K tokens. This adaptability demonstrates Neurocache's potential in improving long-document processing capabilities of current LLMs.

In this context, Neurocache presents a balanced approach to vector retrieval, combining efficiency and adaptability to enhance long-context processing in natural language processing models.

3 Method

3.1 Neurocache Overview

Neurocache addresses the challenge of processing long documents using Transformer decoders, leveraging a k-nearest-neighbor (kNN) search for efficient retrieval and integration of relevant past states. The process begins by segmenting the long text sequences into smaller segments, each containing n tokens, fitting the model's attention window size.

State Compression: Text segments are sequentially processed via a Transformer decoder stack (Vaswani et al., 2017). At the r^{th} layer of the decoder, hidden states $H^r \in \mathbb{R}^{n\times h}$ are acquired and subsequently projected into a compressed form $C \in \mathbb{R}^{n\times d}$ using a learned projection matrix W_p. This compression step enhances the efficiency for the subsequent kNN retrieval.

State Retrieval: For each compressed state $c \in \mathbb{R}^d$ within C, we identify the top-k most similar states $C_{ret} \in \mathbb{R}^{k\times d}$ from the cache $C_{cache} \in \mathbb{R}^{m\times d}$. This selection is based on the L2-distance between each state in C and the states in the cache.

Cache Updating: The cache C_{cache} is updated with the compressed states C, maintaining a fixed size of m entries. This is achieved by discarding the oldest n states, adhering to a First-In-First-Out strategy. The update occurs post-retrieval, reinforcing the commitment to retrieving only relevant past states.

Cache Augmented Layers: Using the states C_{ret} retrieved in the previous step. Starting from the $(r+1)^{th}$ layer, the cache-augmented layers L^j, where $j > r$, integrate a specialized attention mechanism. Each layer uses unique projection matrices W_k^j, W_v^j, and W_q^j to generate keys K_{ret}^j and values V_{ret}^j from C_{ret}, and queries Q^j from the hidden states H^j. The cache attention mechanism is defined as:

$$CA(Q, K_{ret}, V_{ret}) = \text{softmax}\left(\frac{QK_{ret}^T}{\sqrt{d_{key}}}\right) V_{ret}$$

In this formula, Q are the queries derived from H^j, while K_{ret}^j and V_{ret}^j are keys and values derived from C_{ret}, with d_{key} serving as a normalization factor. The output of cache attention is processed by an output matrix W_o^j before being combined with self-attention outputs through a residual connection.

Contextual Retrieval Window: When retrieving the top-k similar cached states, Neurocache also considers additional states surrounding these top-k states within a defined Retrieval Window. This expanded retrieval captures not only the most similar states but also their immediate neighbors, providing a richer context for the model's processing.

Consider the cached states $C_{cache} = [c_1, c_2, \ldots, c_m]$, and a query q for which the cached states c_i and c_j are identified as the top-2. With an even Retrieval Window size w, the retrieved set would include not just c_i and c_j, but also the cached states $[c_{i-(w/2)-1}, \ldots, c_{i+w/2}]$ and $[c_{j-(w/2)-1}, \ldots, c_{j+w/2}]$, truncated at the boundaries of the cache.

Extended Cache-Attention: We enhance the contextual awareness of each token during the cache-attention operation by granting access to the retrievals of preceding tokens. Similar to the contextual retrieval window, this feature broadens the current token's context.

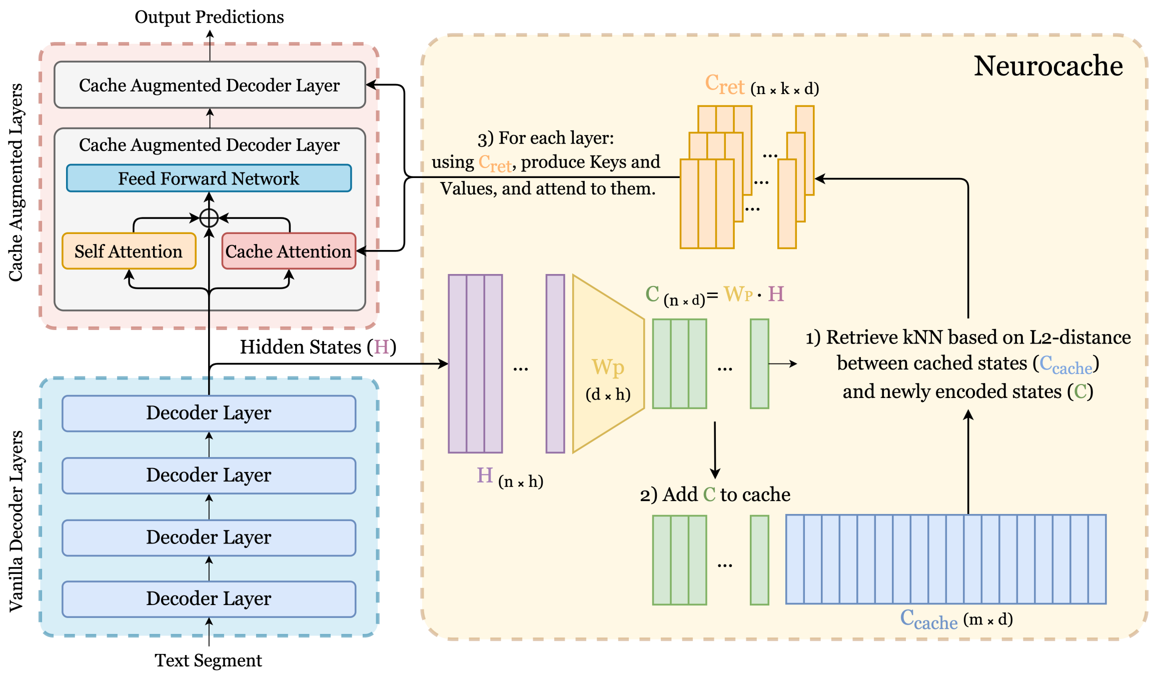

Figure 2: Documents are segmented into sequences of n tokens and processed sequentially through a Transformer decoder stack. For each text segment, mid-layer hidden states $H \in \mathbb{R}^{n\times h}$ are projected into a compact representation $C \in \mathbb{R}^{n\times d}$ using a learned weight matrix $W_p \in \mathbb{R}^{d\times h}$. This projection enhances the efficiency of kNN retrieval of the most relevant past states $C_{ret} \in \mathbb{R}^{n\times k\times d}$ from the cache C_{cache}. These states C_{ret} are used by cache-augmented layers to generate keys/values for cache attention. The output of cache attention is added to the self-attention output before being fed to the feed-forward network (FFN). Finally, the cache C_{cache} is updated to include C while maintaining a constant size of m entries.

Specifically, for a token positioned at i in a sequence, denoted as t_i, and with a predefined context size c, the cache-attention mechanism includes not only its own retrieved states C^i_{ret} but also the states retrieved for the preceding $c-1$ tokens. For example, if $c = 4$, the cache-attention for t_i would integrate the keys $K^{i-3:i}_{ret}$ and values $V^{i-3:i}_{ret}$ from tokens $t_{i-3:i}$.

Please refer to Appendix B for more detailed description on Neurocache.

3.2 Neurocache Adaptation

Adapting pre-trained decoder language models for Neurocache use is a straightforward process that significantly enhances their capability to efficiently process long documents. For the layers augmented with Neurocache, denoted as L^j where $j > r$, the adaptation involves initializing cache-attention weight matrices $(W^j_k, W^j_v, W^j_q, W^j_o)$ by duplicating weights from the corresponding self-attention layers of the pre-trained models. Simultaneously, the projection matrix W_p is randomly initialized to transform hidden states into compact forms suitable for Neurocache retrieval.

Furthermore, we integrate Low-Rank Adapters (LoRA) (Hu et al., 2022) into the feed-forward networks of the cache augmented layers. LoRA, introducing a minimal number of parameters, plays a key role in adapting the models to cache attention without compromising their original strengths.

During training, we freeze the original parameters of the pre-trained model and focus solely on training the newly added weights, specifically the LoRA weights, and the cache-attention weight matrices $(W^j_k, W^j_v, W^j_q, W^j_o)$, along with the projection matrix W_p. This training, using a causal language modeling objective on a corpus of long documents, enables the models to efficiently utilize the Neurocache system.

3.3 Retrieval Overhead

When analyzed per token, the computational overhead of retrieval in our method stems from the following components, which underline the primary computational efforts in the kNN retrieval.

Distance Computation: For each token, the relevance is assessed by calculating the L2-distance between the token's compressed hidden state $c \in \mathbb{R}^d$ and each of the m cached states, resulting in a complexity of $O(d \times m)$ per token, where d is the dimension of the compressed hidden state c and m is the total number of cached entries.

Method	Retrieval Frequency	Entry Size
Neurocache (Ours)	1	d
Memorizing Transformer	a	$2a \times f$
Unlimiformer	$l \times h$	e

Table 1: Space and time complexity of methods based on cache queries per token (Retrieval Frequency) and cache entry dimensions per token (Entry Size). Here, d is the compressed dimension in Neurocache, a the number of attention heads, f head size, e hidden size, and l layers with cache attention.

Top-k Search Over Distances: Identifying the top-k closest states from these distances involves a complexity of $O(m + k)$ for every token[1].

3.4 Comparative Analysis

The Neurocache model demonstrates computational advantage over alternatives like the Memorizing Transformer (Wu et al., 2022) and the Unlimiformer (Bertsch et al., 2023) by performing only one cache query per token. This approach significantly reduces the computational burden. In contrast, the Memorizing Transformer requires multiple cache queries for each token, specifically one for every attention head. Consequently, this leads to an a-fold increase in complexity per token, both for distance computation, $O(a \times d \times m)$, and top-k retrieval, $O(a \times (m + k))$, where a is the number of attention heads, and m is the cache size.

The Unlimiformer, needing $l \times a$ queries per token, further increases retrieval complexity. For instance, a Transformer with 24 layers and 12 attention heads in the Memorizing Transformer configuration would need 12 cache accesses per token. If the Unlimiformer uses half of its layers for augmentation, as per (Bertsch et al., 2023), the requirement rises to $12 \times 12 = 144$ cache accesses per token. Neurocache's strategy of one query per token significantly streamlines this process without compromising accuracy.

Table 1 outlines these models' retrieval frequency and cache entry size, emphasizing Neurocache's efficiency. The table compares the number of cache queries per token and each cache entry's size across the different methods.

4 Language Modeling

We assess Neurocache's effectiveness via two experimental approaches: pre-training language models from scratch and adapting established pre-trained models. For pre-training, TransformerXL (Dai et al., 2019) serves as our baseline, against which we compare Neurocache and Memorizing Transformer (Wu et al., 2022). In terms of adaptation, we focus on pre-trained models including OPT-1.3B, Llama2-7B, and Mistral-7B (Zhang et al., 2022; Touvron et al., 2023b; Jiang et al., 2023).

4.1 Datasets

Our experiments employ two distinct raw text corpora: PG-19, a well-established benchmark for long-form language modeling, and LongPile, a diverse dataset derived from the Pile.

PG-19: This corpus comprises a collection of books written in English and published before 1919, sourced from Project Gutenberg. It is recognized as a standard benchmark for evaluating models on long-form text (Rae et al., 2020; Wu et al., 2022; Hutchins et al., 2022).

LongPile: Extracted from the Pile corpus (Gao et al., 2020), LongPile features extensive documents from varied sources including "Books3," "Gutenberg (PG-19)," "OpenWebText2," "Pile-CC," and "Wikipedia (en)." The selection criterion ensures that each document surpasses 20K tokens, making it suitable for testing models' performance on longer texts.

4.2 Pre-training

Our baseline for pre-training is the TransformerXL model (Dai et al., 2019), which we compare against Neurocache and the Memorizing Transformer (Wu et al., 2022). In these experiments, both Neurocache and the Memorizing Transformer are configured with a fixed storage size of 16K during training, expanding to 128K for evaluation to assess their ability to generalize to larger storage sizes.

In Neurocache, we set the augmented layer threshold r at $3 * n_{layers}/4$, leading to the compression of outputs from the 9^{th} layer of a 12-layer model. The hidden states H, originally of size $h = 1024$, are compressed by a factor of 4, resulting in a reduced size of $d = 256$. We use a retrieval window $w = 2$ to fetch the top-k cached states and their right neighbors for cache-attention in layers 10 to 12. Extending cache-attention to include previous tokens' retrievals with $c = 2$, we set $k = 16$, resulting in 64 neighbors in total. This setup was determined through hyperparameter optimization

[1]We assume an algorithm with quicksort-style partitioning is used.

Model	Params	PG-19 16K	PG-19 128K	LongPile 16K	LongPile 128K
Training from scratch					
TransformerXL	184M	14.442	14.442	15.857	15.857
Memorizing Transformer	184M	13.636	13.494	14.966	14.818
Neurocache	184M	**13.511**	**13.352**	**14.425**	**14.110**
Neurocache adaptation					
Opt-1.3B	1.3B	12.199	12.199	19.446	19.446
+Neurocache	1.4B	**11.306**	**11.227**	**17.626**	**17.377**
Llama2-7B	6.7B	7.359	7.359	9.075	9.075
+Neurocache	7.1B	**7.117**	**7.078**	**8.401**	**8.308**
Mistral-7B	7.2B	7.863	7.863	9.380	9.380
+Neurocache	7.5B	**7.684**	**7.636**	**8.581**	**8.493**

Table 2: Comparison of token perplexity for different models and cache sizes on PG-19 and LongPile datasets. Neurocache outperforms Memorizing Transformer, and presents a significant reduction in perplexity across both pre-training and adaptation experiments, underscoring the adaptability to larger cache sizes.

(details in Appendix A). Additionally, we modify the FFN dimensionality of the Neurocache from 4096, consistent with the baseline and Memorizing Transformer, to 3776 to ensure parity in model sizes.

The Memorizing Transformer, adhering to its original design (Wu et al., 2022), caches key-value pairs from its 9^{th} layer. We align its retrieval setting with Neurocache by setting $k = 64$, thus retrieving the top-64 key-value pairs for each attention head per token.

The pre-training involves 100,000 steps with a batch size of 128 and a context size of 1,024. Adafactor (Shazeer and Stern, 2018) is used for optimization, with a learning rate warming up over the first 1,000 steps, peaking at 2×10^{-2}, and then decaying to 1×10^{-3}.

Neurocache's performance on the PG-19 and LongPile datasets surpasses that of the Memorizing Transformer, as evidenced by its lower token perplexities, detailed in Table 2. Additionally, we assess the scalability of Neurocache in comparison to Memorizing Transformers across various cache sizes. The results, illustrated in Figure 1, demonstrate Neurocache's computational advantage, maintaining its superior performance across different cache sizes.

4.3 Adaptation

We extend our adaptation strategy to pre-trained models such as OPT-1.3B, Llama2-7B, and Mistral-7B (Zhang et al., 2022; Touvron et al., 2023b; Jiang et al., 2023). The adaptation process is identical to that described in Section 3.2, ensuring a smooth integration of Neurocache with the pre-trained model weights.

We set the rank parameter r to 16, the scale parameter α to 32, and turn off bias in LoRA. Added weight matrices and adapter weights are trained on the PG-19 and LongPile datasets' training splits for 25,000 steps, employing the Adam optimizer (Kingma and Ba, 2015) with a decaying learning rate of 1×10^{-4}. We configured Neurocache using the same settings as the pre-training experiments. This adaptation process consumes approximately 200 Nvidia A100 GPU Hours per model.

The successful adaptation is evident in the significant improvement in token perplexity on both datasets, as detailed in Table 2. The subsequent section discusses the impact of these improvements on zero-shot performance in downstream tasks.

5 Downstream Evaluation

We assess the performance of models augmented with Neurocache, particularly Llama2-7B and Mistral-7B adapted on LongPile, using seven distinct downstream tasks from the LongBench suite (Bai et al., 2023). These tasks cover a range of scenarios, including single-document question-answering (QA), multi-document QA, and few-shot learning. We utilize a zero-shot evaluation approach for the single-document and multi-document QA tasks. Conversely, in the few-shot learning tasks, a small set of examples is provided to the models, serving as part of the extended context.

Method	*Single-doc QA*			*Multi-doc QA*		*Few-shot Learn.*	
	NQA F1	**QSP** F1	**MQA** F1	**HQA** F1	**MSQ** F1	**TREC** Acc.	**SAMS** R-L
Input avg. length	35.4K	5.3K	8.1K	17K	19K	7.8K	11.3K
LLAMA2-7B							
TRUNCATION	22.19	28.17	33.39	33.66	12.30	67.00	33.00
LONGLORA	21.92	27.58	30.10	29.17	11.05	69.50	30.29
TEXT RETRIEVAL	23.57	26.71	39.46	**38.51**	**18.89**	66.50	29.38
NEUROCACHE (Ours)	**23.62**	**28.32**	**41.23**	33.30	13.84	**72.00**	**42.77**
MISTRAL-7B							
TRUNCATION	15.64	27.58	40.21	35.22	13.17	68.00	26.47
TEXT RETRIEVAL	14.24	28.67	41.87	**40.92**	**21.17**	66.00	18.06
NEUROCACHE (Ours)	**20.08**	**31.01**	**44.15**	35.49	14.00	**70.00**	**35.86**

Table 3: Zero-shot performance comparison of LLAMA2-7B and MISTRAL-7B using various long document processing methods on the LONGBENCH benchmark tasks. Metrics include F1, Accuracy (Acc.), and Rouge-L (R-L). NEUROCACHE excels in *Single-doc QA* and *Few-shot Learning* but faces challenges in *Multi-doc QA* compared to text retrieval. Document lengths are provided for reference.

5.1 Datasets

The datasets in this evaluation present unique challenges, with average token lengths ranging from 5K to 35K, underscoring the need to process long texts effectively.

5.1.1 Single-document QA

NarrativeQA (NQA) is a question-answering dataset consisting of books from Project Gutenberg and movie scripts. It includes about 30 question-answer pairs per document, providing a robust test for QA systems (Kočiský et al., 2018).

Qasper (QSP) contains questions and answers extracted from NLP papers. This dataset offers diverse question types, such as abstractive, extractive, yes/no, and unanswerable questions, making it a comprehensive testbed for QA models (Dasigi et al., 2021).

MultiFieldQA (MQA) is designed to test a model's ability to understand long contexts across various fields, including legal documents, government reports, and academic papers. It poses a challenge with its questions dispersed throughout lengthy documents (Bai et al., 2023).

5.1.2 Multi-document QA

HotpotQA (HQA) is a multi-document, Wikipedia-based QA dataset. It requires reading and reasoning across multiple documents and includes questions necessitating sentence-level supporting facts for complex reasoning (Yang et al., 2018).

MuSiQue (MSQ) focuses on multihop reasoning in QA. It constructs multi-hop questions from simpler, single-hop ones, demanding a systematic approach and detailed control over the question formation process (Trivedi et al., 2022).

5.1.3 Few-shot Learning

SAMSum (SAMS) presents a dialogue summarization challenge with its dataset of messenger-like conversations and human-annotated summaries[2]. It tests a model's ability to condense conversational data into coherent summaries (Gliwa et al., 2019).

TREC serves as a dataset for few-shot learning tasks in question type classification. Models are tasked with categorizing questions into predefined categories, providing a test of their classification abilities (Li and Roth, 2002).

5.2 Models

In addition to Neurocache, our evaluation includes three distinct approaches for extending the input length of pre-trained language models. These approaches are Input Truncation, Text Retrieval, and Position Interpolation (PI).

Truncation: This approach employs the original Llama2-7B and Mistral-7B models without long-context-specific modifications. Here, inputs exceeding the maximum size of 4,096 tokens are truncated from the middle following (Bai et al., 2023). This baseline serves as a reference to evalu-

[2]We use the rouge package: `https://github.com/pltrdy/rouge`

ate the effectiveness of other methods in processing extended documents.

Text Retrieval: Contrasting with Neurocache, this approach involves selecting the most relevant text segments to include in the input, keeping the total length within the model's maximum input size. We divide the context into 200-word chunks, retrieving the top-7 chunks using Contriever (Izacard et al., 2022). These chunks, along with the input, are then processed by the model. Using the top-7 chunks balances performance and the 4K token limit. This method, used in previous work (Bai et al., 2023; Xu et al., 2023), differs from Neurocache, which dynamically integrates relevant information from the entire document via cache-augmented layers.

Position Interpolation (PI): PI (Chen et al., 2023a) linearly down-scales input position indices to fit the original context window size, avoiding high attention scores that could disrupt the self-attention mechanism. LongLoRA (Chen et al., 2023b), leveraging PI, offers an efficient fine-tuning method to expand the context size of pre-trained models. It uses a sparse local attention mechanism, enabling computation savings while retaining performance. The fully fine-tuned LongLoRA model[3], based on Llama2-7B, extends the maximum input length to 16K tokens, aiming to assess the effectiveness of efficient full-attention methods for longer documents.

Neurocache: We utilize the Neurocache-adapted Llama2-7B and Mistral-7B models in our evaluation. These adaptations follow the configuration detailed in Section 4 for pre-training. The models operate with a fixed cache size of 16K, accommodating the length of most datasets in our study. We split the documents into 2,048-token segments, processing them sequentially to populate the cache. Subsequently, the input, embedded within the prompt, is fed to the model, which then generates the corresponding answer.

5.3 Evaluation Setting

All evaluated models in this study are only pre-trained and not fine-tuned on the downstream tasks. They are assessed in a zero-shot setting, employing greedy decoding for output generation.

As outlined by LongBench (Bai et al., 2023), the model's task is to produce an answer given input and context sequences. In single-doc QA tasks, the input is a question paired with the document as context. For multi-doc QA, the input consists of multiple concatenated documents. In few-shot learning tasks, such as TREC and SAMSum, the context includes a set of examples, and the input is a question or dialogue, respectively. The input and answer are typically concise, while the context can be a long sequence extending to thousands of tokens.

If the combined length of input and context exceeds the model's maximum input capacity, only the context is truncated. This truncation is done from the middle of the context sequence, following the approach in (Bai et al., 2023). We utilize prompt templates provided by LongBench for consistency. Neurocache and LongLoRA operate with a maximum length of 16K tokens, truncating contexts longer than this limit. In contrast, the Text Retrieval method processes the entire context, regardless of length. To ensure comparability, all models are evaluated on identical hardware with a batch size of 1.

5.4 Results

The zero-shot evaluation results across various downstream tasks are summarized in Table 3. We compare the performance of Llama2-7B and Mistral-7B, in their original and Neurocache-adapted forms, against other long document processing methods.

Single-document QA: In tasks like NarrativeQA, Qasper, and MultiFieldQA, Neurocache-adapted models show superior performance, demonstrating their effectiveness in processing long contexts within single documents.

Multi-document QA: Performance in multi-doc QA tasks, such as HotpotQA, reveals a varied picture. While Neurocache-adapted models are competitive, they fall short of Text Retrieval methods. For instance, in HotpotQA, Text Retrieval with the Mistral-7B model achieves the highest F1 score of 40.92. This finding suggests that, despite Neurocache's effectiveness in single-doc scenarios, it may be less effective in multi-doc contexts compared to text retrieval approaches.

Few-shot Learning: In few-shot learning tasks like SAMSum and TREC, Neurocache shows strong performance, particularly indicated by improved Rouge-L scores in SAMSum. This under-

[3]https://huggingface.co/Yukang/Llama-2-7b-longlora-16k-ft

scores its capability to leverage few-shot examples for generating accurate summaries.

These findings illustrate the strengths and challenges of different methods in handling long documents in language models. Neurocache excels in single-document and few-shot learning scenarios, while Text Retrieval methods have an edge in multi-document tasks.

6 Conclusion

This paper introduced Neurocache, an approach designed to improve long document processing in language models. Neurocache employs a k-nearest-neighbor (kNN) strategy for integrating relevant past states from compressed hidden representations, thus extending the context window of Transformer decoders. Notably, Neurocache enhances the maximum context length of models like Llama2-7B and Mistral-7B to 128K tokens.

Our findings indicate that Neurocache offers improvements in inference speed and language modeling accuracy. It demonstrates proficiency in single-document question-answering and few-shot learning, though it faces challenges in multi-document scenarios. Neurocache's competitive performance and adaptability highlight its potential utility in various applications.

In summary, Neurocache contributes to the field by enabling more efficient handling of extended contexts in existing language models. Future work may explore further optimizations for multi-document tasks and the extension of Neurocache to different model architectures and domains.

Acknowledgment

Ali Safaya was supported by the KUIS AI Center fellowship. Deniz Yuret was partially supported by HyperBee.ai. Moreover, parts of the results reported in this paper were performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

Ali Safaya dedicates his work to the People of Gaza.

7 Limitations

While Neurocache demonstrates progress in long document processing with language models, several limitations should be noted. Our evaluation is confined to datasets like PG-19 and LongPile, and tasks from the LongBench suite. These datasets, despite their diversity, might not fully represent all long-context scenarios. Performance may vary in specialized domains like technical documents or source code, which have distinct content characteristics.

A notable limitation is Neurocache's performance in multi-document scenarios, suggesting potential challenges in contexts that require integration of information from multiple sources. This aspect is crucial for applications involving comprehensive data synthesis from various documents.

In terms of bias, Neurocache depends on the underlying language models and datasets for training and evaluation. Consequently, any inherent biases in these components could influence Neurocache's outputs. An explicit analysis of model biases was not conducted in this study, highlighting an area for future exploration.

Another critical point is our reliance on a zero-shot setting for evaluation. The performance of Neurocache might differ if fine-tuning on downstream tasks or instruction datasets was employed. This limitation suggests that our current findings may not fully capture the model's adaptability and efficiency in diverse application scenarios.

In conclusion, while Neurocache presents a step forward in handling long documents in natural language processing, its effectiveness is influenced by the nature of the data, model architecture, and specific task requirements. Understanding these limitations is vital for assessing its practical applicability and guiding future improvements.

References

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. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *Computing Research Repository*, arXiv:2308.14508.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *Computing Research Repository*, arXiv:2004.05150. Version 2.

Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R Gormley. 2023. Unlimiformer: Long-range transformers with unlimited length input. In *Advances in Neural Information Processing Systems*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2021. Improving language models by retrieving from trillions of tokens. *Computing Research Repository*, arXiv:2112.04426.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023a. Extending context window of large language models via positional interpolation. *Computing Research Repository*, arXiv:2306.15595.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023b. Longlora: Efficient fine-tuning of long-context large language models. *Computing Research Repository*, arXiv:2309.12307.

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

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, Online. Association for Computational Linguistics.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800gb dataset of diverse text for language modeling. *Computing Research Repository*, arXiv:2101.00027.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 3929–3938.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yunpeng Huang, Jingwei Xu, Zixu Jiang, Junyu Lai, Zenan Li, Yuan Yao, Taolue Chen, Lijuan Yang, Zhou Xin, and Xiaoxing Ma. 2023. Advancing transformer architecture in long-context large language models: A comprehensive survey. *Computing Research Repository*, arXiv:2311.12351.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. 2022. Block-recurrent transformers. In *Advances in Neural Information Processing Systems*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Computing Research Repository*, arXiv:2310.06825.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *Computing Research Repository*, arXiv:2307.03172.

Erik Nijkamp, Tian Xie, Hiroaki Hayashi, Bo Pang, Congying Xia, Chen Xing, Jesse Vig, Semih Yavuz, Philippe Laban, Ben Krause, Senthil Purushwalkam, Tong Niu, Wojciech Kryściński, Lidiya Murakhovs'ka, Prafulla Kumar Choubey, Alex Fabbri, Ye Liu, Rui Meng, Lifu Tu, Meghana Bhat, Chien-Sheng Wu, Silvio Savarese, Yingbo Zhou, Shafiq Joty, and Caiming Xiong. 2023. Xgen-7b technical report. *Computing Research Repository*, arXiv:2309.03450.

OpenAI. 2023. Gpt-4 technical report. *Computing Research Repository*, arXiv:2303.08774.

Ofir Press, Noah Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 4596–4604. PMLR.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *Computing Research Repository*, arXiv:2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *Computing Research Repository*, arXiv:2307.09288.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, page 6000–6010. Curran Associates, Inc.

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing transformers. In *International Conference on Learning Representations*.

Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Retrieval meets long context large language models. *Computing Research Repository*, arXiv:2310.03025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, pages 17283–17297. Curran Associates, Inc.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models. *Computing Research Repository*, arXiv:2205.01068.

A Optimizing Neurocache Hyperparameters

Effective processing of long documents in Neurocache depends on the optimal tuning of various retrieval hyperparameters. To this end, we conduct a comprehensive hyperparameter search focused on language modeling performance using the Project Gutenberg-19 (PG) dataset.

Our exploration encompasses a range of values for key hyperparameters: the number of retrieved neighbors (k) with values in the set $[None, 8, 16, 32, 64, 128, 256]$, the retrieval window size (w) tested with $[1, 2, 4]$, the cache-attention context size (c) evaluated at $[1, 2, 4]$, and the encoding dimension of hidden states (d) explored across $[1024, 512, 256, 128, 64]$. This systematic investigation aims to identify the optimal configurations that enhance Neurocache's efficiency and effectiveness in handling large-scale textual data.

Number of Retrieved Neighbors (k): The influence of k, the number of retrieved neighbors, on model performance is examined. Table 4 shows that increasing k generally leads to a decrease in perplexity, indicating improved performance. However, the computational cost also increases proportionally with k. We pick $k = 64$ due to the diminishing returns and the increasing cost of larger values.

k	PG (16K)	PG (64K)
None	14.739	14.739
8	14.362	14.398
16	14.242	14.259
32	14.186	14.190
64	14.117	14.118
128	14.069	14.037
256	14.052	13.988

Table 4: Perplexity for varying number of neighbors k.

Retrieval Window Size (w): Adjusting w while fixing the total number of neighbors at 64, we find that a window size of $w = 2$ is optimal, as per Table 5. This setting likely benefits the model's causal processing by including both the top-k entry and the subsequent one. We fix $w = 2$ for the subsequent experiments.

k	w	PG (16K)	PG (64K)
64	1	14.117	14.118
32	2	**13.720**	**13.578**
16	4	13.745	13.596

Table 5: Perplexity for varying retrieval window size w.

Attention Context Size (c): Table 6 shows that the cache-attention context size $c = 2$ achieves the lowest perplexity, indicating optimal performance when extending cache-attention to both the current and previous tokens' retrievals. We fix $c = 2$ for the subsequent experiments.

k	c	PG (16K)	PG (64K)
32	1	13.720	13.578
16	2	**13.704**	**13.564**
8	4	13.791	13.661

Table 6: Perplexity for varying context size c.

Encoding hidden states (d): Finally, we assess the impact of encoding hidden states into smaller dimensions d, as compared to the original $h = 1,024$. Table 7 demonstrate performance degradation as smaller sizes of compression are used.

d	PG (16K)	PG (64K)
1024	13.704	13.564
512	13.740	13.594
256	13.779	13.641
128	13.853	13.730
64	13.983	13.891

Table 7: Perplexity for varying d.

B Neurocache Algorithm

- The cache C_{cache} is initialized to store compact representations, with a maximum capacity of m entries.
- The long document D is segmented into sequences of n tokens each.

Algorithm 1 Neurocache Processing

```
Require: Long document $D$, Segment size $n$, Number of transformer layers $L$, Number of lower layers $r$, Cache memory size
    $m$, Projection dimensions $h$ to $d$, Number of nearest states $k$
Ensure: Updated cache $C_{cache}$ after processing each segment
 1: Initialize cache $C_{cache}$ with size $m \times d$
 2: Divide document $D$ into segments $S = (s_1, s_2, \ldots)$
 3: for each segment $s \in S$ do
 4:     $H^r \leftarrow$ Process $s$ through lower $r$ standard decoder layers
 5:     $C \leftarrow W_p \cdot H^r$                      ▷ Project hidden states to compact representation
 6:     $C_{ret} \leftarrow$ Retrieve top-$k$ nearest states from $C_{cache}$ based on L2-distance to $C$
 7:     for $j \leftarrow r + 1$ to $L$ do
 8:         $Q^j \leftarrow W_q^j \cdot H^j$                ▷ Generate queries for cache attention
 9:         $K_{ret}^j \leftarrow W_k^j \cdot C_{ret}$          ▷ Generate keys for cache attention
10:         $V_{ret}^j \leftarrow W_v^j \cdot C_{ret}$          ▷ Generate values for cache attention
11:         $CA \leftarrow$ Apply cache attention using $Q^j, K_{ret}^j, V_{ret}^j$
12:         $CA \leftarrow CA \cdot W_o^j$                ▷ Apply output projection for cache attention
13:         $H^j \leftarrow$ Combine $CA$ with self-attention outputs and $H^j$
14:     end for
15:     Update cache $C_{cache}$ with $C$, discard oldest if cache exceeds $m$
16: end for
```

- Each segment s_i undergoes sequential processing through the transformer decoder layers.
- At the middle r^{th} layer, the hidden states H^r are converted into a compact representation C.
- The nearest cached states C_{ret} to C are retrieved from C using a k-nearest-neighbor (kNN) method.
- In each augmented layer $j > r$, cache attention is calculated using the generated queries Q^j, keys K_{ret}^j, and values V_{ret}^j.
- The output of cache attention CA is merged with the self-attention outputs and subsequently processed through a feed-forward network (FFN).
- After processing each segment, the cache C is updated with the new compact representation C, and the oldest entries are discarded as needed to maintain the cache size.

The cache-attention mechanism in the augmented layers is designed to focus on the most relevant information retrieved from the cache, akin to the approach in Memorizing Transformers (Wu et al., 2022). Cache-attention implementation is given in Figure 3.

```python
def cache_attention(
    ret_keys,
    ret_vals,
    queries
  ):
  # Attention computation over states retrieved from the cache.
  # ret_keys: Retrieved keys (bsz, n_queries, n_heads, n_neighbors, head_dim)
  # ret_vals: Retrieved values (bsz, n_queries, n_heads, n_neighbors, head_dim)
  # queries: Queries (bsz, n_queries, n_heads, head_dim)

  # Calculate attention weights.
  ret_attn = einsum("...qhd,...khd->...qk", queries, ret_keys)
  ret_attn = softmax(ret_attn, dim=-1)

  # Compute the weighted sum of extended values.
  attn_output = einsum("...qk,...khd->...qhd", ret_attn, ret_vals)
  return attn_output
```

Figure 3: This implementation showcases the cache-attention computation in the model. It calculates the attention weights through a dot product between the queries and keys, applies a softmax to obtain probabilities, and then computes the weighted sum of the extended values to generate the final attention output.