Dynamic Memory Compression: Retrofitting LLMs for Accelerated Inference

Piotr Nawrot [*Q V] **Adrian Łańcucki** [*Q K] **Marcin Chochowski** [Q] **David Tarjan** [Q] **Edoardo M. Ponti** [V]

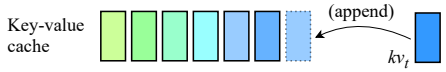
[Q]NVIDIA [K]University of Wrocław [V]University of Edinburgh

Abstract

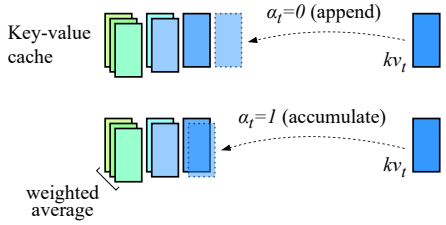
Transformers have emerged as the backbone of large language models (LLMs). However, generation remains inefficient due to the need to store in memory a cache of key–value representations for past tokens, whose size scales linearly with the input sequence length and batch size. As a solution, we propose Dynamic Memory Compression (DMC), a method for on-line key–value cache compression at inference time. Most importantly, the model learns to apply different compression ratios in different heads and layers. We retrofit pre-trained LLMs such as Llama 2 (7B, 13B and 70B) into DMC Transformers, achieving up to ~3.7× throughput increase during auto-regressive inference on an NVIDIA H100 GPU. DMC is applied via continued pre-training on a negligible percentage of the original data without adding any extra parameters. We find that DMC preserves the original downstream performance with up to 4× cache compression, outperforming up-trained grouped-query attention (GQA) and key–value eviction policies (H_2O, TOVA). GQA and DMC can be even combined to obtain compounded gains. As a result DMC fits longer contexts and larger batches within any given memory budget. We release the DMC code and models at https://github.com/NVIDIA/Megatron-LM/tree/DMC.

1. Introduction

Transformer Large Language Models (LLMs) are the state of the art in generative and conversational AI (Touvron et al., 2023; Jiang et al., 2023). Their deployment, however, is curtailed in part by their inefficiency. This is not only due to the quadratic complexity of attention layers (Bahdanau et al., 2015; Vaswani et al., 2017): during generation, Transformers store the keys and values of past tokens in memory to avoid recomputing them multiple times. Since this key–value (KV) cache grows linearly with the sequence length and batch size, generation with Transformers quickly becomes prohibitive due to the excessive memory load. This issue emerges even more clearly with long-context generation (e.g., in dialogues and stories) or when serving large numbers of user queries.



(a) Regular key–value cache with items kv_i depicted as boxes. New items are always appended.



(b) Dynamic Memory Compression (DMC) chooses whether to accumulate or append current items, resulting in a smaller key–value cache.

Figure 1: Key–value cache update mechanisms.

A widespread solution to increase the memory efficiency of Transformers during inference is Grouped Query Attention (GQA; Ainslie et al., 2023; Shazeer, 2019), which uses a number of key and value heads inferior to the number of query heads through parameter sharing. As an alternative, the number of tokens in memory can be reduced through token merging (Zhang et al., 2018; Liu et al., 2018; Bolya et al., 2023) or cache eviction policies (Zhang et al., 2023; Oren et al., 2024). Nevertheless, these methods often pay the price of a severe degradation in downstream performance. On the other hand, hardware/IO-aware (Dao et al., 2022; Kwon et al., 2023) and sub-quadratic algorithms for attention (Beltagy et al., 2020; Choromanski et al., 2021) do not alleviate the memory load of the KV cache.

[*]Equal contribution . Correspondence to: Piotr Nawrot <piotr.nawrot@ed.ac.uk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

In our work, we aim to adaptively compress the KV cache of LLMs, retaining their performance while reducing their memory load. To this end, we propose Dynamic Memory Compression (DMC). As shown in Figure 1, during every time step, DMC decides whether to append the current key and value representations to the cache or to perform a weighted average of them with the top item on the cache. Note that memory grows sub-linearly in DMC, which therefore falls halfway between vanilla Transformers and state space language models (Fu et al., 2023; Gu & Dao, 2023), where memory is constant.

In our experiments, we equip pre-existing LLMs—such as Llama 2 (Touvron et al., 2023) 7B, 13B, and 70B—with DMC by retrofitting them on a negligible percentage of the original pre-training data (~2% for 2× compression and ~4% for 4× compression) and without adding any extra parameters to the original LLM. We evaluate our DMC models on a series of downstream tasks such as MMLU (Hendrycks et al., 2021) for factuality, QA datasets for common-sense reasoning, and HumanEval (Chen et al., 2021) for code. We find that DMC LLMs retain a downstream performance similar to the original LLM, whereas baselines—such as GQA, H_2O, and TOVA—incur significant degradation at high compression ratios. Finally, we show that DMC can be hybridized with GQA such that their compression ratios are compounded. For Llama 2 70B, which is pre-trained with GQA $8\times$, DMC $2\times$ achieves a total compression of $16\times$.

We verify that KV cache compression translates into more efficient generation in practice. We measure that DMC $4\times$ increases the inference throughput between 340% and 370% for Llama 2 7B and 13B on NVIDIA H100 or A100 GPUs without loss in performance. In fact, it allows us to fit larger batches and longer sequences into a given memory budget. Finally, compression schemata learned by DMC provide insights into the internal structure of the LLMs, revealing a preference for compressing heads in higher layers.

2. Background

2.1. Multi-Head Self-Attention

Let $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathbb{R}^{n \times d}$ denote the input sequence of hidden states of a Transformer layer, where n stands for the number of tokens in a sequence, and d for the hidden state dimension. A Multi-head Self-Attention (MHSA) layer divides the embeddings into n_h different heads. Afterwards, the self-attention process is applied to each head separately. This enables the model to focus on different parts of the input, capturing various types of relationships in the data. For each head h, different weight matrices $W_q^h, W_k^h, W_v^h \in \mathbb{R}^{d/n_h \times d/n_h}$ are used to project the input sequence into queries Q^h, keys K^h, and values V^h:

$$Q^h = (W_q^h \mathbf{x}_1, \ldots, W_q^h \mathbf{x}_n) \quad (1)$$

$$K^h = (W_k^h \mathbf{x}_1, \ldots, W_k^h \mathbf{x}_n) \quad (2)$$

$$V^h = (W_v^h \mathbf{x}_1, \ldots, W_v^h \mathbf{x}_n). \quad (3)$$

The attention scores and outputs for the i-th token are then computed as

$$a_{ij}^h = \frac{\exp(\mathbf{q}_i^{h\top} \mathbf{k}_j^h / \sqrt{d_h})}{\sum_{t=1}^{i} \exp(\mathbf{q}_i^{h\top} \mathbf{k}_t^h / \sqrt{d_h})}, \quad \mathbf{o}_i^h = \sum_{j=1}^{i} a_{ij}^h \mathbf{v}_j^h. \quad (4)$$

Finally, the outputs from all heads are concatenated and linearly transformed to produce the final output $O \in \mathbb{R}^{n \times d}$:

$$O = (W_o[\mathbf{o}_1^1, \ldots, \mathbf{o}_1^{n_h}], \ldots, W_o[\mathbf{o}_n^1, \ldots, \mathbf{o}_n^{n_h}]) \quad (5)$$

where $W_o \in \mathbb{R}^{d \times d}$.

2.2. KV Caching During Inference

In a Transformer decoder, the generation of sequences is auto-regressive: each new token is predicted based on the previously generated ones. To avoid redundant computation, it is common to store the keys and values of previously computed tokens in the KV cache. For each time step i, only the keys and values for the current token $\mathbf{x}_i$ are computed whereas those for $\mathbf{x}_{<i}$ are retrieved from the cache. Thus for each head h:

$$K^h = [K_{1:i-1}^h, W_k^h \mathbf{x}_i] \quad (6)$$

$$V^h = [V_{1:i-1}^h, W_v^h \mathbf{x}_i] \quad (7)$$

Note that this process is not necessary for queries as only the query for the current token $\mathbf{x}_i$ is needed at each inference time step.

As a consequence, the KV cache grows linearly with each new token. This progressive expansion leads to substantial memory consumption and increased latency, especially for long input sequences. This issue is further exacerbated when serving multiple requests concurrently, as each inference process requires its own KV cache, significantly straining the system's resources.

3. Method: Dynamic Memory Compression

Inference with LLMs tends to be memory bound rather than compute bound, as we explain in Appendix A in detail. This means that latency scales linearly with the size of the KV cache. We tackle the problem of reducing the size of KV cache which brings two immediate benefits (Zhang et al., 2023): 1) it lowers the latency of auto-regressive generation,

and 2) improves GPU utilization by allowing for larger batch sizes and sequence lengths, leading to increased throughput.

In this section we introduce Dynamic Memory Compression (DMC), a simple and inexpensive method for on-line compression of the KV cache at inference time. In what follows, we first describe the inference-time operation of DMC, which constitutes our end goal. Next, we illustrate how to teach a pre-trained LLM such behavior through short, continued pre-training.

3.1. Inference

Consider the forward pass of an attention layer during autoregressive inference (Section 2.1). In a vanilla Transformer, at every time step t, both $\mathbf{k}_t$ and $\mathbf{v}_t$ are appended to the KV cache (Section 2.2). In DMC, on the other hand, the KV cache update procedure is different, as detailed in Algorithm 1. First, a decision variable $\alpha_t \in \{0, 1\}$ and importance variable $\omega_t \in [0, 1]$ are predicted. In order to avoid adding new parameters, we reuse the first neuron from $\mathbf{k}_t$ and $\mathbf{q}_t$, respectively, to extract the two scores.[1]

Algorithm 1 Single-head KV cache update with Dynamic Memory Compression (DMC)

Require: $K, V, \mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t$
$\alpha_t \leftarrow \text{round}(\text{sigmoid}(\mathbf{k}_t[0]))$
$\omega_t \leftarrow \text{sigmoid}(\mathbf{q}_t[0])$
$\mathbf{k}_t[0], \mathbf{q}_t[0] \leftarrow 0, 0$
if $\alpha_t = 1$ **then** ▷ ACCUMULATE
  $z_t \leftarrow z_{t-1} + \omega_t$
  $K \leftarrow [K_{1:l-1}, (K_l z_{t-1} + \mathbf{k}_t \omega_t)/z_t]$
  $V \leftarrow [V_{1:l-1}, (V_l z_{t-1} + \mathbf{v}_t \omega_t)/z_t]$
else ▷ APPEND
  $z_t \leftarrow \omega_t$
  $K \leftarrow [K_{1:l}, \mathbf{k}_t]$
  $V \leftarrow [V_{1:l}, \mathbf{v}_t]$
end if
return $K, V, \mathbf{q}_t, \mathbf{k}_t$

Based on α_t, a decision is made whether KV representations $\mathbf{k}_t$ and $\mathbf{v}_t$ are appended to the cache or accumulated with its last element (Figure 1).

In particular, for accumulation we perform a weighted average based on a predicted importance scores ω for the current token and the running sum of importance scores z_t for all the tokens since the last time step $\alpha = 0$ was predicted.

In fact, the α variable effectively segments the input sequence: each decision determines if the current segment should continue ($\alpha = 1$) or a new segment should be opened ($\alpha = 0$). As a result, after the update, the cache length for DMC is $l = \sum_{t=1}^{n}(1 - \alpha_t) \leq n$, whereas in vanilla Transformers it is always $l = n$. In what follows, we will refer to the ratio n/l between the length n of an uncompressed cache and the compressed length l as the Compression Ratio (CR).

Finally, multi-head self-attention is calculated similarly to vanilla Transformer using KV cache sequences, with the exception that KV sequences for different heads might have different lengths. Algorithm 1 is applied to every MHSA layer and head independently. Note that Algorithm 1 can be implemented efficiently without the if-then-else statement conditioned on α_t, instead multiplying the previous $\mathbf{k}_i$, $\mathbf{v}_i$ and z_i by α_t like in Equation (9).

3.2. Training

The DMC inference algorithm switches between accumulating and appending tokens to the KV cache. In order to endow LLMs with DMC, we continue pre-training them on a negligible amount of their pre-training data, gradually increasing the compression ratio towards a target. However, this poses serious challenges. First, we opt for end-to-end learning via gradient descent and continuous relaxation of the decision variables. As a result, we have to define an operation for KV cache updating when $0 < \alpha < 1$, resulting in partly aggregated, partly accumulated key and value states. Second, to avoid training–inference mismatch, we must simulate the DMC behavior at inference time while parallelizing training across a sequence of tokens: as a consequence the length of K and V is *not* reduced through compression during training; rather, all intermediate states of keys and values are explicitly kept in memory and an auxiliary (gradually discretized) mask regulates interactions between queries and keys. We elaborate on our solutions to these challenges below.

Gradient Estimation for Discrete Decisions The decision whether to accumulate or append at inference time is a discrete one; however, rounding $\text{sigmoid}(\mathbf{k}[0])$ to the nearest integer for training would result in a non-differentiable operation with zero gradients. Hence, we resort to stochastic reparametrization of the decision variable during training

$$\alpha_t \sim \text{Gumbel-sigmoid}(\mathbf{k}[0] - c, \tau) \in [0, 1], \qquad (8)$$

where τ is the temperature[2] and c is a constant subtracted so that every $\alpha \approx 0$ at training step 0. Similarly, we add c to the importance variables ω_t so that every $\omega_t \approx 1$ at the beginning. This ensures that DMC initially performs no compression and starts the training behaving just like the vanilla Transformer.

[1] Note that this choice is arbitrary as we could use any two neurons from $\{\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t\}$.

[2] Low temperatures sharpen α_t into almost-discrete values which accurately mimics inference behavior.

Partial accumulations As we relax our discrete decisions, we now must define a mechanism to update the KV cache that generalizes Algorithm 1 to continuous α. Hence, we define partially accumulated states for $\alpha \in [0, 1]$ as:

$$\begin{aligned} z_0 &\leftarrow \omega_0, & z_i &\leftarrow z_{i-1}\alpha_i + \omega_i, \\ \mathbf{k}_0 &\leftarrow \mathbf{k}_0, & \mathbf{k}_i &\leftarrow \frac{\alpha_i \mathbf{k}_{i-1} z_{i-1} + \mathbf{k}_i \omega_i}{z_i}, \\ \mathbf{v}_0 &\leftarrow \mathbf{v}_0, & \mathbf{v}_i &\leftarrow \frac{\alpha_i \mathbf{v}_{i-1} z_{i-1} + \mathbf{v}_i \omega_i}{z_i}. \end{aligned} \tag{9}$$

Note that when $\alpha \in \{0, 1\}$, Equation (9) defaults to Algorithm 1.

Intermediate compression steps Aside from key and value computations shown in Equation (9), the rest of the forward pass can be performed in parallel for all tokens in the sequence. Nonetheless, this creates a mismatch between training and evaluation, since during training all intermediate states of keys and values are accessible in self-attention.

To illustrate this issue, consider the example of a KV cache during DMC inference shown in Figure 2 for the sequence of decision scores $\alpha_{1:5} = (1, 1, 0, 1, 0)$ (importance scores ω have been omitted for clarity). The last element of the KV cache changes at every time step. In order to properly simulate the inference-time evolution of the KV cache during training, we keep all unrolled intermediate KV cache items. In lieu of an auto-regressive 'causal' mask, however, we use an additive mask based on the sequence of α values to modify the attention scores a_{ij}^h from Equation (4), which is shown in Figure 3. During training, the α values 1) naturally converge towards 0 or 1 as the model strives to satisfy the language modeling criterion and reduce uncertainty; 2) are intentionally pushed to almost-discrete states by the Gumbel noise and low temperature setting. Such binarization of α values significantly impacts the attention scores - it strengthens the queries interactions with last elements of every key–value segment, and weakens those with the intermediate elements, which are discarded during inference.[3] In fact, when $\alpha \in \{0, 1\}$, the matrix is filled with either 0 or $-\infty$ values, and exactly corresponds to the inference time query-to-key attendance pattern.

Training objective The model is incentivized to compress the KV cache to a certain CR, and thus increase the predicted α values. Instead of matching the desired rate for each append-or-accumulate decision α, we calculate a *global* one-sided loss as the difference between the sum of all decisions and the expected sum of KV tokens across all layers l, heads h and time steps t under the desired Compression Ratio (CR), normalized by $(n_l n_h n)$:

[3] See Appendix H for details on the mask implementation.

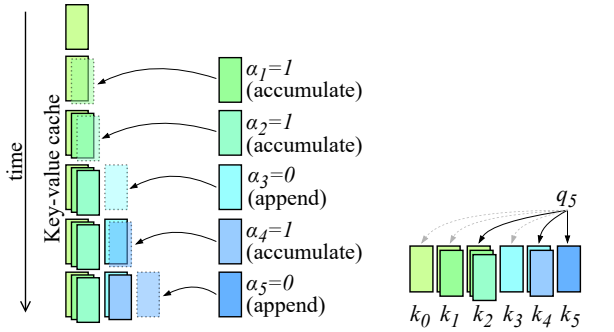


Figure 2: An example of KV cache growth in DMC during inference (left). During training (right), we retain all intermediate states seen during inference and gradually block access to some of them (gray arrows)

$$\begin{array}{c} \\ \mathbf{q_0} \\ \mathbf{q_1} \\ \mathbf{q_2} \\ \vdots \\ \mathbf{q_n} \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{k_0} & \mathbf{k_1} & \mathbf{k_2} & \dots & \mathbf{k_n} \end{array} \\ \begin{bmatrix} 0 & -\infty & -\infty & \dots & -\infty \\ \log(1 - \alpha_1) & 0 & -\infty & & -\infty \\ \log(1 - \alpha_1) & \log(1 - \alpha_2) & 0 & & -\infty \\ \vdots & & & \ddots & \vdots \\ \log(1 - \alpha_1) & \log(1 - \alpha_2) & \log(1 - \alpha_3) & \dots & 0 \end{bmatrix} \end{array}$$

Figure 3: Additive mask applied during training to the normalized attention scores in order to block queries from attending intermediate KV states (as well as future states).

$$\ell_{\text{CR}} = \frac{1}{n_l n_h n} * max(0, \sum_{l=1}^{n_l} \sum_{h=1}^{n_h} \sum_{t=1}^{n} (1 - \alpha_{lht}) - \frac{n_l n_h n}{\text{CR}}). \tag{10}$$

It is added to the language modeling loss term $\ell_{\text{LM}} = -\sum_{t=1}^{n} \log p_{\boldsymbol{\theta}}(\mathbf{x}_t \mid \mathbf{x}_{<t})$, with the final objective of the training being:

$$\arg\min_{\boldsymbol{\theta}} \ell_{\text{LM}} + \ell_{\text{CR}}. \tag{11}$$

Importantly, the training procedure is designed for slowly ramping up the target CR and taking ready-to-use DMC checkpoints along the way. This is possible because all hyperparameters, like Gumbel-sigmoid sampling temperature and learning rate, are not decayed and remain constant throughout training. A practical use case of this DMC property is to produce a series of DMC checkpoints with different CRs within a single run, and then choose the one with the desired efficiency–performance trade-off.

3.3. Practical Considerations

Storing a variable-length KV cache in memory DMC allows every head to learn a custom compression, which

results in KV cache sequences with variable lengths across heads. This poses difficulties in storing these sequences efficiently in an n-dimensional tensor, considering that they will be extended during auto-regressive generation by uneven amounts of tokens due to the adaptive compression rate of DMC. However, such sequences can be easily stored in memory with little overhead using PagedAttention (Kwon et al., 2023), where new pages are allocated on demand for every head separately. In Section 5.2 we present latency and throughput measured with an implementation based on FlashAttention (Dao et al., 2022) and PagedAttention.

Window grouping approximation The calculation of partial accumulations, during training of DMC models Equation (9), for a sequence of n tokens requires $O(n)$ sequential steps, therefore considerably slowing down the training. In order to improve the time complexity, we calculate Equation (9) at every time step t independently over a short window of the last w elements up to time t (e.g., $w = 12$). This enables us to reduce the span of computation to $O(w)$, provided that at least n threads can execute the computation in parallel for each of the n positions. During inference, this speed-up also applies to the prompt phase. However, the sliding window comes at a disadvantage that it needs to be cached during inference for heads that tends to accumulate more than w tokens.

4. Experimental Setup

Baselines In our experiments, we evaluate strategies to retrofit a state-of-the-art Large Language Model (LLM), Llama 2 (Touvron et al., 2023) [4], into a more efficient model across various sizes: 7B, 13B, and 70B. In addition to comparing the downstream performance of DMC with the original model, we also use Grouped Query Attention (GQA) as a main baseline, as it constitutes the most widespread strategy to ensure KV cache efficiency (Jiang et al., 2023).

We also compare DMC with two KV-Cache eviction policies that do not require model retrofitting: Token Omission Via Attention (TOVA; Oren et al., 2024) and Heavy-Hitter Oracle (H_2O; Zhang et al., 2023). H_2O keeps in memory a fixed window of the most recent tokens, as well as additional heavy-hitter (H_2) tokens. H_2 tokens are chosen dynamically: specifically, they correspond to the tokens with the highest aggregated attention scores throughout the sequence. On the other hand, TOVA retains the top-k tokens based on the attention weights of the last token only. This means that for a given attention layer, at each decoding step, the token with the lowest attention score is omitted.

Checkpoint adaptation To equip the original Llama 2 with GQA, we perform the standard checkpoint conversion proposed by Ainslie et al. (2023): the key and value projection matrices are split by head. Then the resulting sub-matrices corresponding to heads in the same group are merged via averaging. Note that this results in a fixed compression during training.

As for DMC, we avoid the introduction of new parameters by re-purposing the first dimension from both the $\mathbf{q}_t$ and $\mathbf{k}_t$ representations, in order to predict segmentation decisions α_t and importance scores ω_t. Setting $\mathbf{q}_t$ and $\mathbf{k}_t$ to zero triggers a significant increase in language modeling loss, by disrupting the attention scores. To counteract this spike in loss, we pre-train the model to disregard the first dimension of $\mathbf{q}_t$ and $\mathbf{k}_t$ in the attention calculations. Specifically, we load the raw pre-trained weights and up-train the model for 1 billion tokens (250 steps), annealing the values of $\mathbf{q}_t$ and $\mathbf{k}_t$ to 0 according to the following formula:

$$\begin{aligned}\mathbf{q}_t[0] &\leftarrow \mathbf{q}_t[0] \times (1 - (t/n_t)) \\ \mathbf{k}_t[0] &\leftarrow \mathbf{k}_t[0] \times (1 - (t/n_t))\end{aligned} \tag{12}$$

where t is the current step and $n_t = 250$. After this initial phase, which allows the model to ignore the first dimension of keys and values for attention calculations, we start the main retrofitting phase, where the model learns to compress the KV representations.

Training hyperparameters We strictly follow the training hyperparameters outlined by Touvron et al. (2023). We employ the AdamW optimizer with parameters $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 1e-5$, in conjunction with a weight decay of 0.1 and gradient clipping of 1.0. The batch size is 1024 with a sequence length of 4096. We apply a constant learning rate identical to the final rate from the original Llama 2 pre-training phase: 3×10^{-5} for the 7B and 13B models, and 1.5×10^{-5} for the 70B model. We set the constant from Equation (8) as $c = 5$ which in practice results in $\alpha_t = 0.0067$ and $\omega_t = (1 - 0.0067)$. Empirically, $c = 5$ is a high enough value so that we do not experience a spike in language modeling loss at the start, yet low enough to be easily changed by learning $\mathbf{q}_t[0]$ and $\mathbf{k}_t[0]$ through gradient optimization. Finally, we set the window size (Section 3.3) to 12, and keep the Gumbel-sigmoid temperature constant at 0.1 throughout the entire training. We do not perform any exhaustive searches for these values - we believe that the DMC retrofitting procedure is robust to a wide range of sensible hyperparameter choices.

Training schedule The volume of data for continued pre-training of DMC is contingent on the targeted KV cache compression ratio; a larger CR necessitates more data. We use a linear training schedule with 24B, 48B, and 72B tokens for training to $2\times$, $3\times$, and $4\times$ compression, respectively. In Appendix F we include an ablation where we use a schedule with twice less data.

[4] Obtained from https://huggingface.co/meta-llama.

For DMC, we discovered that the annealing strategy was crucial. Starting the training without compression and its measured increase helps to preserve the original perplexity. Through extensive ablations (see Appendix F), we found that any significant increase of perplexity, even if recovered during continued pre-training, prevents the model from regaining its performance on downstream tasks. The target CR is linearly increased from $1\times$ to $4\times$ for the 7B and 13B models, and from $1\times$ to $2\times$ for the 70B model.[5].

Upon achieving the target compression ratio, we initiate a final solidifying phase wherein we: 1) continue up-training for an additional 8B tokens, 2) maintain a fixed compression ratio, and 3) implement a cosine learning rate schedule, annealing it down to 10% of the initial value. This phase aims at stabilizing the model with a specific, fixed compression ratio. Resource-wise, all retrofitting stages requires roughly 4k, 8k and 28k GPU hours of NVIDIA H100 for Llama 2 7B, 13B and 70B respectively.

Evaluation Following Touvron et al. (2023), we evaluate models on a series of downstream tasks, including MMLU (Hendrycks et al., 2021) for factuality, HumanEval (Chen et al., 2021) for Python code generation, and several question-answering datasets for common-sense reasoning: PIQA (Bisk et al., 2020), BoolQ (Clark et al., 2019), Arc-C and Arc-E (Clark et al., 2018), HellaSwag (Zellers et al., 2019), and WinoGrande (Sakaguchi et al., 2020). We report the 5-shot performance on MMLU, average pass@1 scores for HumanEval, and average 0-shot performance on common-sense benchmarks (CS-QA). For pass@1 scores we use a temperature of 0.1 and nucleus sampling (Holtzman et al., 2020) with top-p $= 0.95$.

We adapted TOVA and H_2O to our codebase based on publicly released code. For a given CR, the total budget for both policies is calculated as $(1/\text{CR}) \times n$ for MMLU and CS-QA tasks, where n is the input length. For HumanEval, which instead involves generating more than one token, the initial budget is also $(1/\text{CR}) \times n$ but then increases dynamically as we generate the answer. For H_2O, the budget is split equally between the local window and heavy-hitter tokens.

5. Results

5.1. Main Results

We report the performance of the original LLM (equivalent to $1\times$ CR) and efficient variants (DMC, GQA, TOVA, and H_2O) in Table 1. For the original LLM we use results reproduced in our codebase as described in Appendix B.

DMC vs Original First, comparing DMC with the original LLM, we note that it even *increases* the performance in MMLU and CS-QA at $2\times$ CR for 7B and 13B and in HumanEval for all model scales. We speculate that this is due to the additional up-training steps, which expose LLMs to new examples. For the other combinations of downstream tasks and scales at $4\times$ CR, DMC incurs negligible degradation: -0.7 in MMLU and -0.3 in CS-QA for 7B, -0.3 in MMLU and CS-QA for 13B. This encouraging finding suggests that DMC is robust across different model scales even for $4\times$ CR. Overall, the fact that DMC is in general close or superior to the original performance makes it suitable as a drop-in replacement for KV caching to achieve higher inference efficiency.

DMC vs GQA Moreover, Table 1 allows for comparing DMC with GQA for equivalent CRs ($2\times$ and $4\times$). Overall, DMC *surpasses* GQA applied through up-training, in both

Scale	Method	CR	MMLU	CS-QA	Human Eval
7B	–	–	44.6	70.5	14.0
	GQA	2×	39.8	68.9	12.8
	H_2O		**45.2**	67.5	9.8
	TOVA		44.9	70.0	6.1
	DMC		**45.2**	**70.8**	**15.2**
	GQA	4×	34.7	68.3	14.0
	H_2O		41.1	56.8	4.9
	TOVA		43.4	64.2	1.8
	DMC		**43.9**	**70.2**	**16.5**
13B	–	–	54.5	73.5	17.5
	GQA	2×	50.2	72.7	15.9
	H_2O		54.1	70.3	16.5
	TOVA		54.4	72.8	12.2
	DMC		**54.8**	**74.2**	**20.7**
	GQA	4×	48.6	72.2	16.5
	H_2O		50.7	60.0	8.5
	TOVA		53.3	67.8	2.4
	DMC		**54.2**	**73.2**	**22.0**
70B*	–	8×*	68.8	78.0	29.6
	H_2O	16×*	68.7	74.1	18.3
	TOVA		68.1	77.6	**29.9**
	DMC		**68.8**	**77.9**	**29.9**

Table 1: MMLU accuracy, Commonsense Question Answering (CS-QA) accuracy averaged across 6 tasks, and HumanEval Pass@1 for several scales (7B, 13B, and 70B) and compression ratios (CRs; $1\times$, $2\times$, $4\times$) of Llama 2. (*) Unlike the 7B and 13B models, the 70B model was trained with GQA which compresses the KV cache $8\times$. We apply additional $2\times$ DMC compression during up-training to obtain a total compression of $16\times$.

[5]For Llama 2 70B, we do not up-train to $4\times$ because this LLM is already pre-trained with GQA $8\times$.

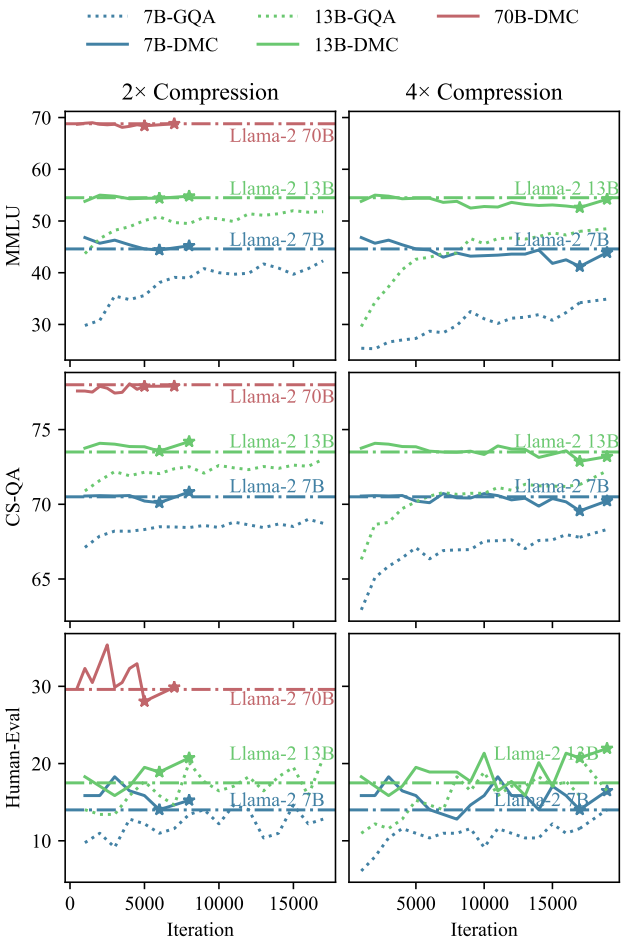
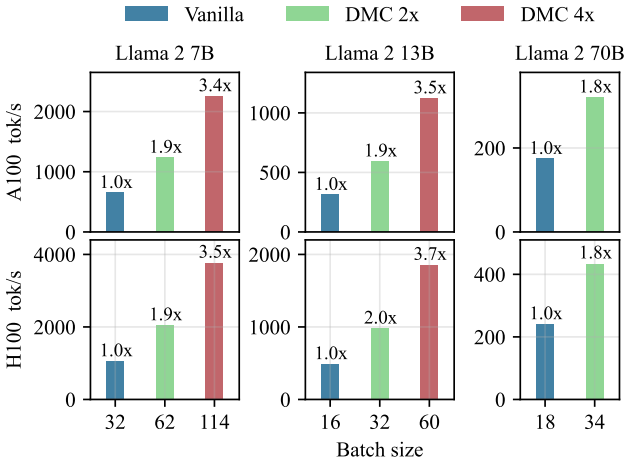
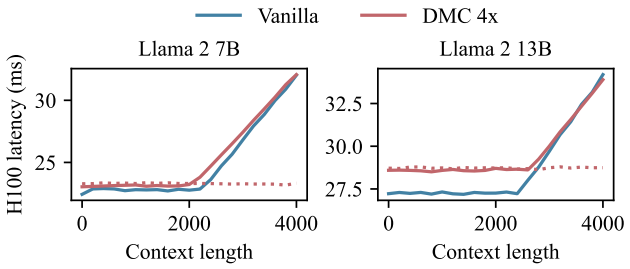


Figure 4: Sample efficiency of DMC and GQA. Horizontal lines correspond to the performance of the original Llama 2. Every DMC model was trained first with increasing CR, then with constant CR for the last 2K steps (marked with $\star$).



(a) Inference throughput averaged over the generation of 1K tokens with 3K tokens of context (up to 4K in total). On the x-axis, we show the maximum batch size that fits in memory on a single GPU (7B and 13B) or two GPUs with tensor parallelism (70B) for the Vanilla, DMC 2×, and DMC 4× models.



(b) Latency of next token generation. Solid lines denote measurements with the maximum batch size that fits on a single GPU. Dotted lines denote DMC 4× with the same batch size as the vanilla LLM.

Figure 5: Efficiency measurements with the Megatron-LM framework on NVIDIA A100 80GB and H100 GPUs.

CRs and in both scales (7B and 13B). For MMLU, the gap widens when we increase the CR. This holds true both at the smaller 7B scale (from +5.4 for 2× CR to +9.2 for 4× CR) and at the larger 13B scale (from +4.6 for 2× CR to +5.6 for 4× CR). For CS-QA and Human-Eval, on the other hand, we observe comparable gains over GQA across CRs and model scales. These findings illustrate that DMC should be preferred to GQA for retrofitting LLMs into variants that are more efficient at inference time.

DMC vs Cache Eviction Policies Both H2O and TOVA evict tokens from memory according to the post-softmax attention scores, which makes them incompatible with efficient attention implementations like FlashAttention. In fact, this requires materialising the n^2-sized tensor of attention scores. As a result, while reducing the KV cache size, these policies may slow down inference. Moreover, comparing the performance of both cache eviction policies with DMC in Table 1, it emerges how they are almost comparable at CR 2× in MMLU and CS-QA; however, they drop significantly in accuracy with a higher CR of 4×. The gap is even more dramatic for HumanEval pass@1 at all CRs and scales, except only for 70B TOVA.

70B: DMC *and* GQA Many widely adopted LLMs were pre-trained with GQA which leads to the question of whether *DMC and GQA can be used together to reap compounded benefits*. To investigate this, we retrofit Llama 2 70B, which has been pre-trained with 8× GQA. We further compress its KV cache with DMC to 2× CR: this is equivalent to a cache 16× smaller than a vanilla LLM with neither GQA nor DMC. We observe that the performance remains unchanged, and conclude that DMC and GQA can be easily and successfully combined.

Sample efficiency To shed light on the sample efficiency of DMC and GQA, we report their performance on MMLU,

CS-QA, and HumanEval across retrofitting steps in Figure 4. First, for a target CR of $2\times$, it emerges how GQA cannot achieve the performance that DMC obtains after fine-tuning (at 8K steps) even after more than double the amount of fine-tuning (at 17K steps). This applies to both 7B and 13B scales. Figure 4 also reveals the importance of the fine-tuning phase in DMC: a limited amount of extra steps with a fixed CR recovers a significant amount of the original performance (especially for higher target CRs such as $4\times$).

5.2. Throughput and Latency Measurements

To verify whether increased CRs result in concrete efficiency gains, in Figure 5 we present the performance properties of DMC, estimated within the NVIDIA Megatron-LM framework (Narayanan et al., 2021). Specifically, we run measurements on a single GPU (NVIDIA A100 80GB SXM or H100 SXM) in bfloat16 precision for Llama 7B and 13B. For Llama 70B, we run the same measurements on two GPUs of the same type with tensor parallelism. We feed the model with 2K tokens of English text, and generate additional 2K tokens in an auto-regressive manner to ensure that the model properly compresses its own generations. The reported throughput consists of the average over the last 1K tokens. We limit the sequence to 4K tokens to avoid issues with context length extrapolation, as this is the maximum length observed by Llama 2 during pre-training.

In order to maximize the GPU utilization, we increase the batch size to the maximum that fits into memory (see Appendix A for details). The compression of the KV cache with DMC allows for substantial increases in batch size and thus significant throughput gains. As shown in Figure 5a, DMC $2\times$ translates into an effective increase in tokens per second compared to the original LLM of $> 1.8\times$ for 7B, 13B, and 70B on both A100 and H100 GPUs. Similarly, DMC $4\times$ translates into gains between $3.4\times$ and $3.7\times$. This means that the efficiency boost observed in practice is close to the theoretical limit. In addition, the extra memory saved with DMC could also be used to cache longer contexts.

Finally, we compare the latency of the original LLM with DMC $4\times$ in Figure 5b. When we fit the largest possible batch size for either model, after generating approximately 2200 tokens, the inference time begins to scale linearly with the context size due to the increasingly dominating cost of reading the KV cache from the high bandwidth memory (HBM). On the other hand, if we choose to maintain the same batch size as for the original LLM also for DMC $4\times$, the memory footprint of the KV cache is reduced and latency for longer contexts improves significantly.

While we acknowledge that the behavior of LLMs at inference time depends on a multitude of factors and implementation details, our measurements in Figure 5 offer evidence that DMC increases throughput and reduces the latency of autoregressive generation with LLMs. We speculate that in the future, DMC might be used to grow the KV cache sublinearly, which would provide an alternative between vanilla Transformers and State Space Models, where memory is constant (Fu et al., 2023; Gu & Dao, 2023).

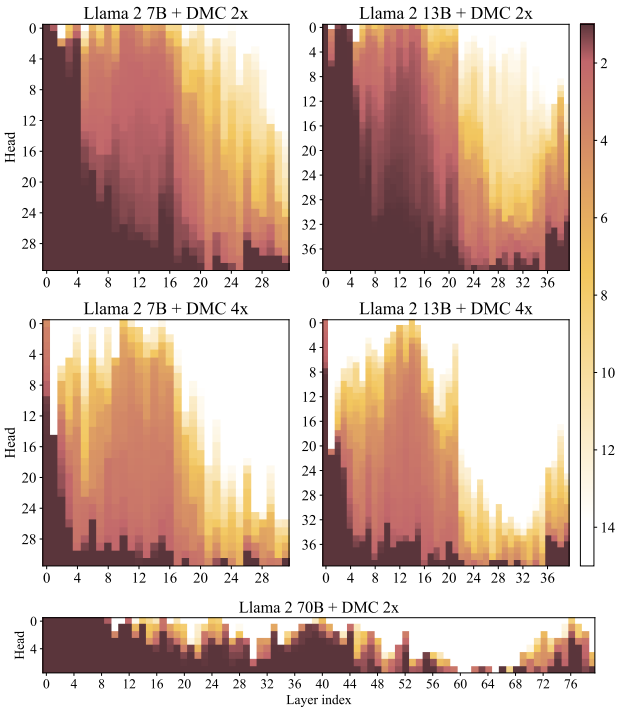


Figure 6: Heatmaps of average compression ratios across layers (X-axis) and heads (Y-axis). Heads are arranged from the highest compression to the lowest top-down for clarity.

5.3. Per-Head Learned Compression Ratios

Since the training loss does not enforce any compression schema *a priori*, as it just requires to match a certain *global* CR, we can investigate what schema the model discovers in practice. In Figure 6, we report the CR for each layer and head for different scales (7B, 13B, and 70B) and CRs ($2\times$ and $4\times$). From all schema, it emerges how compressing deeper layers (> 16 for 7B, > 22 for 13B, > 44 for 70B) is universally the most popular strategy. However, the very final layers are compressed to a somewhat reduced degree. Fascinatingly, $4\times$ DMC achieves extremely high CRs for several heads also in the few layers after the very first one. This is counter-productive as token representations are not contextualized yet, which makes taking the correct decision (whether to append or accumulate) hard.

This same pattern (in terms of relative preference for compressing the certain ranges of layers) also corresponds to how the distribution of CRs across layers changes throughout training steps, as we push the model towards the target

CR with auxiliary loss ℓ_{CR} . Figure 7 in Appendix D illustrates how CR increases first in deeper layers, then in some non-contiguous intermediate ranges.

6. Related Work

Efficient inference in Transformer models is a subject of extensive research, with detailed overviews provided by several surveys (Pope et al., 2022; Treviso et al., 2022). This section narrows its focus to advancements in Transformer inference efficiency through KV cache size reduction.

Grouped Query Attention (GQA; Ainslie et al., 2023) represents the most widespread strategy, evolving from Multi Query Attention (MQA; Shazeer, 2019). GQA reduces the number of KV heads by allocating shared KV representations across subsets of query heads. Multi-Head Latent Attention (DeepSeek-AI, 2024) compresses the size of token's KV state via low-rank projection which is shared across all query heads. Prior efforts in token merging (Zhang et al., 2018) condensed the entire past context into a single token, while (Liu et al., 2018; Rae et al., 2020) employed strided convolution and mean pooling kernels to reduce the number of KV tokens. Sliding window attention techniques (Beltagy et al., 2020; Child et al., 2019) restrict attention to a maximum of w preceding tokens. Quantization-based approaches (Sheng et al., 2023; Liu et al., 2024; 2023b; Hooper et al., 2024) reduce the KV precision to a smaller number of bits. Though effective in limiting KV cache, such methods perform *fixed-size* compression, unlike the presented *dynamic* DMC, which adapts the compression schema based on the input sequence. Such adaptation yields superior results, as we prove in an ablation in Appendix G.

Previous learnable compression methods (Anagnostidis et al., 2023, *inter alia*) decide which tokens to drop from the KV cache. DMC takes a different approach as instead of dropping tokens it merges them. Hence, it preserves cached information more faithfully. Moreover, the DMC compression mechanism has constant complexity with respect to the context length, while the one proposed by Anagnostidis et al. (2023) is linear. Mu et al. (2023) instead compresses prompts through costly generation which limits their inference benefits. Moreover, this method is only applicable to compressing the model input while DMC compresses *on-the-fly* the entire sequence, including both the model input and the generated output.

Non-learnable cache eviction strategies (Zhang et al., 2023; Sheng et al., 2023; Liu et al., 2023a; Wang et al., 2020; Ge et al., 2023; Oren et al., 2024) utilize attention scores or token properties to filter tokens in the KV cache. These approaches, while bypassing additional training, rely on heuristics and lack the ability to learn the compression mechanisms. In contrast, DMC integrates compression into its learning objective in an end-to-end manner, where compression is synergistic to language generation.

Finally, DMC draws inspiration from Dynamic Token Pooling (Nawrot et al., 2023), which introduces a learnable boundary predictor to merge the representations of groups of tokens in intermediate layers. DMC improves upon this idea by applying it to the KV cache of and introducing a continuous relaxation of the pooling decision during training. Moreover, it enables retrofitting pre-trained LLMs with minimal extra steps rather than training language models from scratch.

7. Conclusions and Future Work

We proposed Dynamic Memory Compression, a method to reduce the length of the KV cache in Transformers, which enhances the memory efficiency and speed of LLMs at inference time. For every new token, DMC learns end-to-end whether to append its KV representations to the cache or merge them with the top element in the cache. We show how to retrofit LLMs such as Llama 2 at different scales (7B, 13B, and 70B) into efficient DMC versions with a negligible amount of extra data and without extra parameters. DMC LLMs with $2\times$ and $4\times$ compression ratios (CRs) retain (or even improve upon) the performance of the original LLM. In addition, we demonstrate that, for comparable CRs, DMC has significantly higher continued pre-training performance and sample efficiency than Grouped Query Attention (GQA), a widespread method for KV cache size reduction.

Acknowledgements

The authors would like to thank Mostofa Patwary for sharing the data blend we used to retrofit the models, Szymon Migacz for his assistance with the computing cluster, as well as Przemysław Strzelczyk, Daniel Korzekwa, and Bryan Catanzaro for helpful discussions and support in releasing this paper. This work was supported in part by the UKRI Centre for Doctoral Training in Natural Language Processing, funded by the UKRI (grant EP/S022481/1) and the University of Edinburgh, School of Informatics and School of Philosophy, Psychology & Language Sciences.

Impact Statement

Dynamic Memory Compression in Large Language Models (LLMs) like Llama 2 results in better computational efficiency, reducing both operational costs and environmental impact (Patterson et al., 2021). By enabling higher throughput and lower latency, DMC democratizes access to advanced AI, making state-of-the-art models suitable for a broader range of hardware. This may not only accelerate

innovation across diverse sectors but also promote AI development and applications in an environmentally conscious manner.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. GQA: training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://doi.org/10.18653/v1/2023.emnlp-main.298.

Anagnostidis, S., Pavllo, D., Biggio, L., Noci, L., Lucchi, A., and Hofmann, T. Dynamic context pruning for efficient and interpretable autoregressive transformers. In *Advances in Neural Information Processing Systems 36*, 2023.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1409.0473.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *ArXiv*, abs/2004.05150, 2020. URL https://arxiv.org/abs/2004.05150.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020. URL https://doi.org/10.1609/aaai.v34i05.6239.

Bolya, D., Fu, C., Dai, X., Zhang, P., Feichtenhofer, C., and Hoffman, J. Token merging: Your vit but faster. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/pdf?id=JroZRaRw7Eu.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Ponde, H., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D. W., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Babuschkin, I., Balaji, S., Jain, S., Carr, A., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M. M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *ArXiv*, abs/1904.10509, 2019.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *9th International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Clark, C., Lee, K., Chang, M., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019. URL https://doi.org/10.18653/v1/n19-1300.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model, 2024.

Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Re, C. Hungry Hungry Hippos: Towards language modeling with state space models. In *The International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=COZDy0WYGg.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms. *ArXiv*, abs/2310.01801, 2023.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *ArXiv*, abs/2312.00752, 2023.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. In *8th International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

Hooper, C., Kim, S., Mohammadzadeh, H., Mahoney, M. W., Shao, Y. S., Keutzer, K., and Gholami, A. KVQuant: Towards 10 million context length llm inference with kv cache quantization. *ArXiv*, abs/2401.18079, 2024.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mistral 7B. *ArXiv*, abs/2310.06825, 2023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. URL https://doi.org/10.1145/3600006.3613165.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., and Shazeer, N. Generating wikipedia by summarizing long sequences. In *6th International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hyg0vbWC-.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems 36*, 2023a.

Liu, Z., Oğuz, B., Zhao, C., Chang, E., Stock, P., Mehdad, Y., Shi, Y., Krishnamoorthi, R., and Chandra, V. LLM-QAT: Data-free quantization aware training for large language models. *ArXiv*, abs/2305.17888, 2023b.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. *ArXiv*, abs/2402.02750, 2024.

Mu, J., Li, X., and Goodman, N. D. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems 36*, 2023.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V. A., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., and Zaharia, M. A. Efficient large-scale language model training on gpu clusters using megatron-lm. *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021. URL https://doi.org/10.1145/3458817.3476209.

Nawrot, P., Chorowski, J., Łańcucki, A., and Ponti, E. M. Efficient transformers with dynamic token pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://aclanthology.org/2023.acl-long.353.

Oren, M., Hassid, M., Adi, Y., and Schwartz, R. Transformers are multi-state RNNs. *ArXiv*, abs/2401.06104, 2024.

Patterson, D. A., Gonzalez, J., Le, Q. V., Liang, C., Munguía, L.-M., Rothchild, D., So, D. R., Texier, M., and Dean, J. Carbon emissions and large neural network training. *ArXiv*, abs/2104.10350, 2021.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *ArXiv*, abs/2211.05102, 2022. URL https://doi.org/10.48550/arXiv.2211.05102.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. In *8th International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020. URL https://doi.org/10.1609/aaai.v34i05.6399.

Shazeer, N. M. Fast transformer decoding: One write-head is all you need. *ArXiv*, abs/1911.02150, 2019. URL http://arxiv.org/abs/1911.02150.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Fu, D. Y., Xie, Z., Chen, B., Barrett, C. W., Gonzalez, J., Liang, P., Ré, C., Stoica, I., and Zhang, C. High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/sheng23a.html.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S.,

and Scialom, T. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288, 2023. URL https://arxiv.org/abs/2307.09288.

Treviso, M. V., Ji, T., Lee, J.-U., van Aken, B., Cao, Q., Ciosici, M. R., Hassid, M., Heafield, K., Hooker, S., Martins, P. H., Martins, A. F. T., Milder, P., Raffel, C., Simpson, E., Slonim, N., Balasubramanian, N., Derczynski, L., and Schwartz, R. Efficient methods for natural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 11, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, H., Zhang, Z., and Han, S. Spatten: Efficient sparse attention architecture with cascade token and head pruning. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2020. URL https://doi.org/10.1109/HPCA51647.2021.00018.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://aclanthology.org/P19-1472.

Zhang, B., Xiong, D., and Su, J. Accelerating neural transformer via an average attention network. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018. URL https://aclanthology.org/P18-1166.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z. A., and Chen, B. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36*, 2023.

Appendix

A. Memory-Bound Operations in Transformers

During autoregressive generation with a KV cache, the sequence length during every forward pass is $n = 1$. The vast majority of time is spent on calculations for linear layers and multi-head self-attention. For linear layers, the ratio of FLOPs to input bytes improves as the batch size increases. For small batch sizes, the operations are memory bounded on reading the weight matrices from HBM. On the other hand, for MHSA layers during inference the ratio of FLOPs to input bytes does not change and MHSA layers are always memory bounded on current GPUs. The impact of KV cache compression is two-fold as it 1) allows us to decrease the latency of MHSA layers, and 2) allows us to fit larger batch sizes in HBM, resulting in better throughput and better utilization of GPUs during the calculations of linear layers.

B. Replicating the Original Results

To make sure that our implementation is correct, for each downstream task we compare the performance reported in the original Llama 2 paper (Touvron et al., 2023) with those obtained from the Hugging Face Hub checkpoints. Furthermore, we evaluate the impact of using our internal data mixture for up-training, acknowledging that variations in data proportions and preprocessing methodologies can influence model behavior. In particular, we up-train the vanilla pretrained Llama 2 checkpoint for 200 training steps, amounting to 1B tokens, in accordance with the original Llama 2 training schedule. We compute the average and standard deviation of checkpoints after 50, 100, 150, 200 steps. In our experiments, we replicate the results reported by the Llama 2 paper almost exactly, as shown in Table 2. Furthermore, we observe that retrofitting the Llama 2 checkpoint on our data mixture has little effect on the model's performance on the downstream benchmarks.

C. Retrofitting Data

The retrofitting corpus comprised sections of The Pile, including BookCorpus2, Books3, Pile-CC, Gutenberg (PG-19), NIH ExPorter, OpenWebText2, Stack Exchange, and Wikipedia (en). Additional datasets included in the corpus were ArXiv, Pushshift Reddit, mC4 (multilingual C4), Common Crawl (CC) dumps from 2017-2023, CC-News, PubMed Central, BigScience Workshop datasets, and The Stack dataset (BigCode project).

D. Analysis of the Compression Schema Learned by DMC

Evolution throughout the training In Figure 7, we illustrate how the CR changes for each layer of the Llama 2 7B model throughout the training from $1\times$ up to $4\times$ global CR. Each subplot corresponds to a different global CR which occurs at different stages of the training, going from the smallest (1.17) at the top, to the highest (4.16) at the bottom. There is a clear trend such that, for a smaller global Compression Ratio (i.e. at the beginning of the training), the model emphasizes compression in the later layers. As the global Compression Ratio increases, the model keeps on compressing in final layers but also starts to compress the earlier layers. We hypothesize that the token representations in the initial layers do not contain sufficient information to perform any meaningful grouping. Conversely, token representations in the subsequent layers are more defined and, possibly, after several attention layers, already contain redundant/shared information.

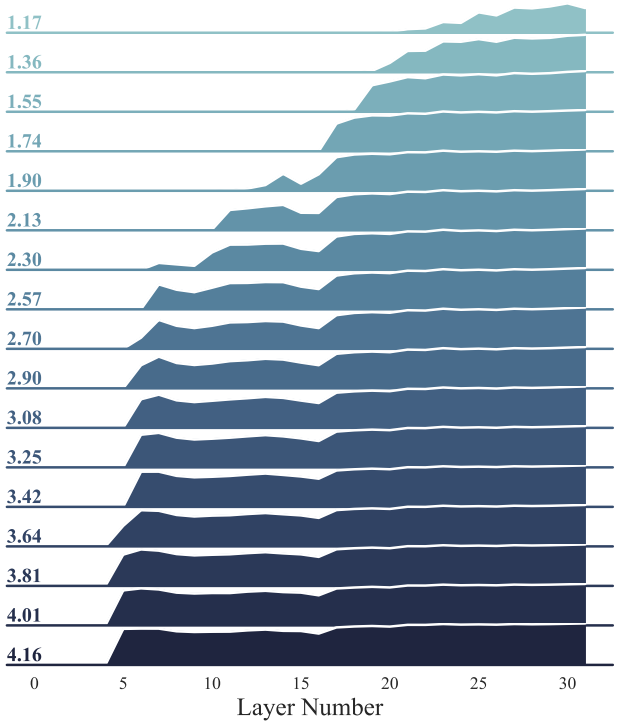


Figure 7: Compression distribution across layers at different stages of retrofitting a Llama 2 7B model. We adhere to the convention where, for a given subplot, a larger space above a given layer indicates greater compression at that layer.

Sequence Length versus Compression Ratio Do DMC models compress sequences with a uniform CR independent from their total length? We find that this is not the case. As show by Figure 8, the CR increases logarithmically as we

	CS-QA			MMLU			Human-Eval		
	7B	**13B**	**70B**	**7B**	**13B**	**70B**	**7B**	**13B**	**70B**
Paper	70.6	73.7	78.5	45.3	54.8	68.9	12.8	18.3	29.9
Checkpoint	70.6	73.7	78.5	45.7	55.1	69.1	13.4	17.7	30.5
Up-trained	70.5 ± 0.2	73.5 ± 0.1	78.0 ± 0.1	44.6 ± 0.6	54.5 ± 0.3	68.8 ± 0.3	14.0 ± 1.2	17.5 ± 1.5	29.6 ± 1.6

Table 2: Replicating the original up-training results.

increase the total sequence length. This holds true across all global CRs (including both 2× and 4×).

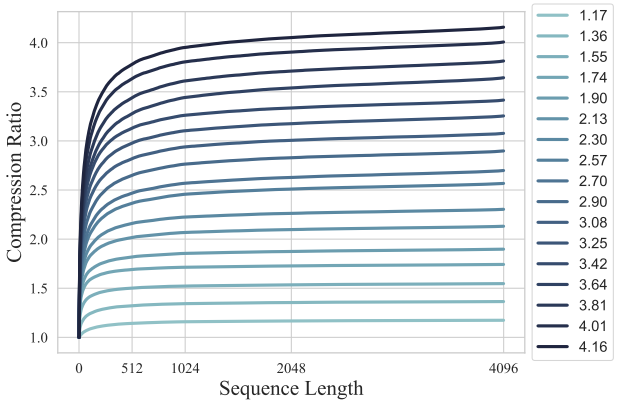


Figure 8: CR achieved by Llama 2 7B for particular sequence lengths across various global CRs.

Absolute Position versus Compression Decision Do DMC models learn a fixed compression schema, or do they exhibit position biases? In Figure 9, we plot the average value of the decision variable α across positions in the sequence (0 to 4096). Our observations reveal that the average value of the decision variable α is independent of a token's position in the input sequence which demonstrates that the model does not follow some fixed pattern. This persists across both 2× and 4× compression ratios (CR).

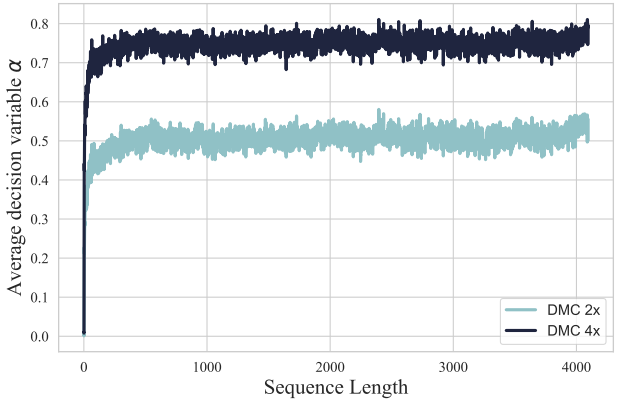


Figure 9: Average value of the decision α for positions (0, 4096) averaged over 128 samples, heads and layers.

Interpretability A natural question arises, whether the compression schema that the model learns is somehow aligned with human intuitions about text segmentation. We analyzed the outputs of Llama 2 13B DMC with CR 4 and noticed that some heads compress according to the boundaries of linguistic units, such as words or syntactic phrases. Figure 10 shows the compression schema learned by head 14 in layer 0. In this case, the model merges the subwords back into words *reverting* the tokenizer. Interestingly, some groupings of tokens correspond to semantic units, e.g., "*1 9 th century*"', "*5 0 percent*", or "*a week back later*". Yet, we also stress that many heads and layers are not interpretable as their behavior does not overlap with linguistic units.

More generally higher layers merge longer tokens sequences, in line with Figure 6. For instance, Figure 11 shows the decisions of layer 24 head 2. We leave a more in-depth analysis of compression schemata learned by DMC to future work.

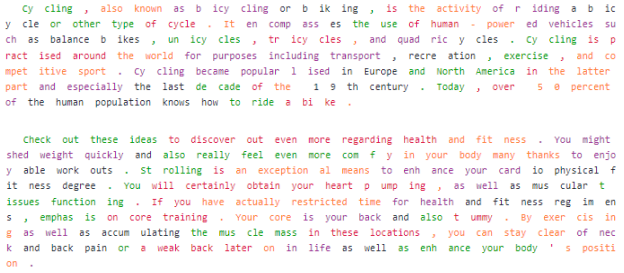


Figure 10: Compression schema found by Llama 2 13B DMC 4× in layer 0, head 14. Tokens that are merged in the KV cache are marked with the same color.

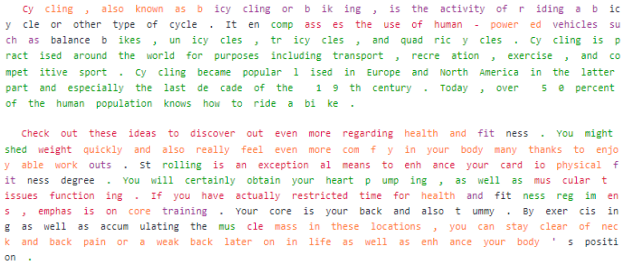


Figure 11: Compression schema found by Llama 2 13B DMC 4× for layer 24, head 2. Tokens that are merged in the KV cache are marked with the same color.

Scale	Method	CR	MMLU	CS-QA	Human Eval
7B	–	–	44.6	70.5	14.0
	GQA	2×	39.8	68.9	12.8
	DMC	2×	45.2	**70.8**	**15.2**
	DMC-C	2×	**45.5**	70.6	14.6
	GQA	4×	34.7	68.3	14.0
	DMC	4×	**43.9**	**70.2**	**16.5**
	DMC-C	4×	38.2	69.6	14.6
13B	–	–	54.5	73.5	17.5
	GQA	2×	50.2	72.7	15.9
	DMC	2×	**54.8**	**74.2**	**20.7**
	DMC-C	2×	**54.8**	73.9	18.3
	GQA	4×	48.6	72.2	16.5
	DMC	4×	**54.2**	**73.2**	**22.0**
	DMC-C	4×	52.4	72.9	18.3
70B*	–	8×*	68.8	78.0	29.6
	DMC	16×*	**68.8**	**77.9**	**29.9**
	DMC-C	16×*	67.4	78.2	31.1

Table 3: MMLU accuracy, Commonsense Question Answering (CS-QA) accuracy averaged across 6 tasks, and HumanEval Pass@1 for several scales (7B, 13B, and 70B) and compression ratios (CRs; 1×, 2×, and 4×) of Llama 2. Here, we include an extra DMC variant - DMC-C, which does not require custom implementation. (*) The 70B model was trained with GQA which compresses the KV cache 8×.

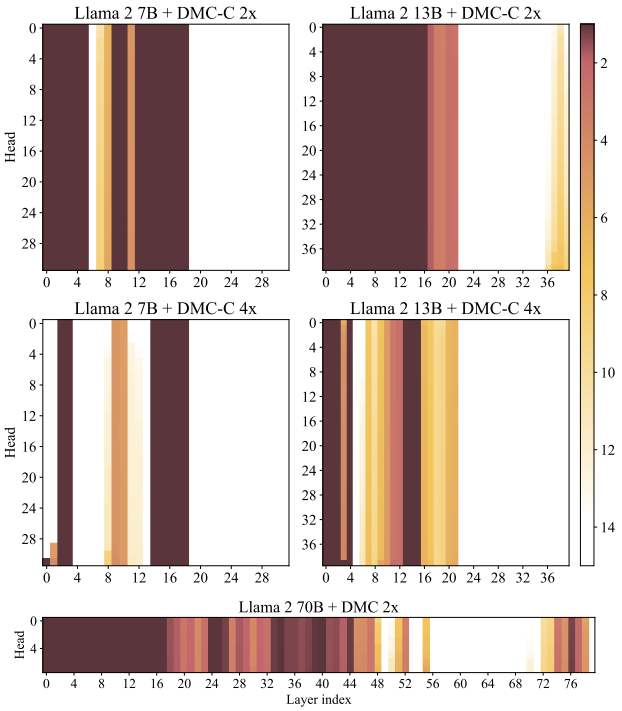


Figure 12: Heatmaps of average compression ratios across layers (X-axis) and heads (Y-axis) for DMC-C. Heads are arranged from the highest compression to the lowest top-down for clarity.

E. Similar Per-layer Compression Rates

In order to alleviate the problem of efficiently storing and retrieving the KV cache of variable-length sequences, we also study a Constrained variant of our algorithm (DMC-C) in which we force all heads in a given layer to maintain a similar compression ratio. A similar compression ratio allows us to store key and value sequences naïvely as padded tensors with minimal padding. To this end, we add an extra loss term to Equation (10)

$$\ell_{\text{H}} = \sum_{l=1}^{n_l} \sum_{h=1}^{n_h} \sum_{t=1}^{n} \left| \alpha_{lht} - \sum_{h=1}^{n_h} \alpha_{lht} \right| . \tag{13}$$

Table 3 compares DMC with its Constrained variant DMC-C. In general, while remaining superior to GQA, DMC-C displays a significant degradation in several configurations, most notably 7B 4× where it records a drop of -6.4 in MMLU compared to the ceiling. On the other hand, DMC recovers all performance loss in DMC-C. When combined with custom attention implementations that do not require excessive padding, standard DMC should therefore be vastly preferred, as it retains the original LLM performance while fully reaping the advantages in memory efficiency.

Compression Schemata learned by DMC-C Studying the compression schema in Figure 12 learned by DMC-C, we find a very different pattern compared to DMC, due to the auxiliary loss forcing the model to compress similarly across heads in the same layer. Nevertheless, we observe a similar global preference for compressing deeper layers.

F. Training Ablations

Training Steps per Increase in CR Another advantage of DMC is its high flexibility. In fact, based on the availability of resources, different regimes can be chosen when annealing the CR to the target during up-training. In Figure 13, we compare a SHORT and a LONG regime for the constrained variant of DMC (DMC-C), which continuously increase the CR by 1 every 3K and 6K steps (12B and 24B tokens), respectively. It is evident how there exists a trade-off between training steps (hence, time) and performance. Additionally, Figure 13 showcases another aspect of the higher flexibility DMC affords: it is compatible with arbitrary real-valued CRs, as opposed to integer CRs divisible by 2 as in GQA.

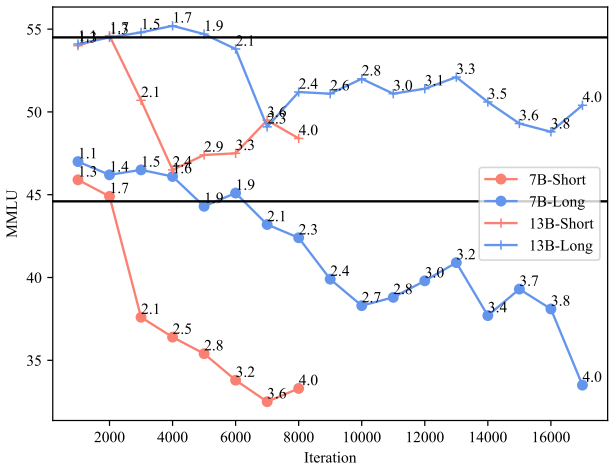


Figure 13: Different up-training regimes for DMC-C: Short (red) increases CR by 1 every 6K steps, Long (blue) increases CR by 1 every 3K steps. Horizontal lines correspond to the performance of the original Llama 2.

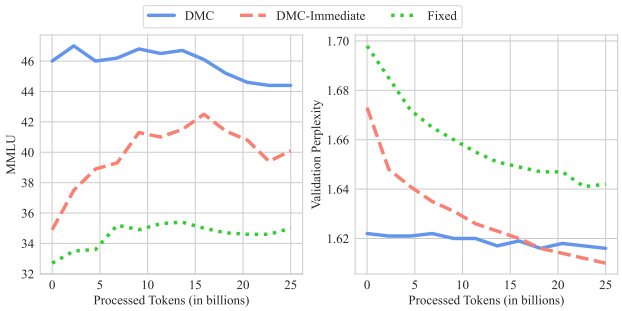


Figure 14: Validation perplexity and MMLU accuracy vs training steps for Fixed Memory Pooling and a variant of DMC where the auxiliary loss CR is immediately set to the target on the onset of training. All models follow the regular training schedule and are trained up to 2× CR.

Schedules of Target CR Additionally, we explore how different schedules for the target CR impact the model performance. In the standard setup, this is linearly annealed from 1 to the target CR throughout the duration of training. Here, we compare it with a setup where the CR used in the auxiliary loss for compression is set to the target from the start (DMC-immediate). We show the results in Figure 14. As expected, DMC-immediate has a perplexity spike at the beginning when the model quickly increases the CR due to the auxiliary loss. While perplexity is recovered during training, even to a lower point than DMC with annealing, downstream accuracy on MMLU benchmark is degraded across the board. This showcases why avoiding perplexity spikes is fundamental to successfully retrofit an LLM.

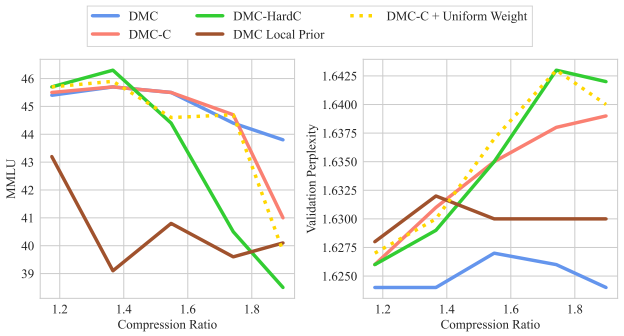


Figure 15: Validation perplexity and test MMLU accuracy vs compression ratio for different types of compression priors, in-layer relaxations, and importance scores. All models follow the SHORT training regime and are trained up to 2× CR.

G. DMC Ablations

Fixed vs Learned Memory Compression We assessed the importance of dynamically learning compression decisions in DMC by comparing it with Fixed Memory Pooling, which reduces the overall number of tokens in memory by deterministically averaging every n tokens, where n in this case is identical to the compression ratio. The results, shown in Figure 14, demonstrate that the dynamic component of DMC is crucial to achieve lower perplexity as well as higher downstream accuracy.

Global vs Local (Layer-wise) Compression Prior We compare two approaches to compression: a *Local* Prior, which enforces a pre-specified compression ratio (CR) in each layer independently, requiring every layer to compress approximately the same amount, and a *Global* Prior used by default DMC, which applies a pre-specified CR across all layers, giving the model the freedom to apply different CRs in each layer, provided that their average compression equals the global CR. Figure 15 clearly indicates that the Global Prior (DMC in Figure 15) improves MMLU performance compared to the Local Prior.

In-layer Relaxation We then compare three strategies to determine how similar compression schemata for heads within each layer should be (assuming a global prior):

1. DMC: There are no constraints on the decision and importance scores, except for the global loss nudging the model towards a pre-defined CR.

2. DMC-C: Different heads can have varying decision and importance scores within each layer. However, an auxiliary loss encourages the model to maintain similar CRs among all heads within the layer.

3. DMC-HardC: Decision scores α_t and importance scores ω_t are shared across heads, leading to the same shortening schema within each layer across heads.

As per Figure 15, the default DMC strategy shows a consistent MMLU performance across varying CRs, while both DMC-C and DMC-HardC exhibit a sharp drop in MMLU as the compression reaches 1.9×. Moreover, in Table 3 we report a more thorough comparison between DMC and DMC-C. In general, while remaining superior to GQA, DMC-C displays a significant degradation in several configurations when compared to regular DMC.

Importance Scores Finally, we assess the impact of predicting importance scores for accumulation as opposed to uniformly weighting each token in a group. Figure 15 shows that DMC-C with Uniform Weighting is worse than learned weighting DMC-C.

H. Masking Implementation Details

We mask the unnormalized attention score for the pair (i, j) as follows:

$$\hat{a}_{(i,j)} = \underbrace{\frac{\mathbf{q}_i[1:d_h]^\top \mathbf{k}_j[1:d_h]}{\sqrt{d_h}}}_{\text{attention score}} + \underbrace{\log(1-\alpha_{j+1})}_{\text{attention mask}}.$$

We rely on the memory efficient implementation of MHSA from PyTorch, which allows adding arbitrary masks to the attention scores before softmax. Notwithstanding this, at inference time DMC remains compatible with efficient libraries for attention such as Flash Attention (Dao et al., 2022). The $\log(1-\alpha_{j+1})$ term is calculated as log-sigmoid$(-\alpha_{j+1})$ for better numerical precision.

I. Limitations

This paper is focused on retrofitting existing LLMs into DMC variants. In our preliminary experiments with *pre-training LLMs with DMC from scratch* we obtained negative results when compared to the training curve of GQA. We speculate that this is due to the mutual dependency of modeling and segmenting data: when token representations are not of sufficient quality, boundary decisions are unreliable. Vice versa, incorrect boundary decisions may lead to poor token representations. This creates a vicious cycle which may be broken by techniques that facilitate convergence, such as an Expectation Maximization-style alternation between modeling and segmenting.