NoMAD-Attention: Efficient LLM Inference on CPUs Through Multiply-add-free Attention

Tianyi Zhang
Dept. of Computer Science,
Rice University
xMAD.ai
Houston, TX
tz21@rice.edu

Jonah Yi
Dept. of Computer Science,
Rice University
xMAD.ai
Houston, TX
jwy4@rice.edu

Bowen Yao
Dept. of Computer Science,
Rice University
Houston, TX
by18@rice.edu

Zhaozhuo Xu
Dept. of Computer Science,
Stevens Institute of Technology
xMAD.ai
Hoboken, NJ
zxu79@stevens.edu

Anshumali Shrivastava
Dept. of Computer Science, Rice University
Ken Kennedy Institute
ThirdAI Corp.
xMAD.ai
Houston, TX
anshumali@rice.edu

Abstract

Large Language Model (LLM) inference on Central Processing Units (CPU) is challenging due to the vast quantities of Multiply-Add (MAD) matrix operations in the attention computations. This paper highlights a rare gem in modern CPUs, Single-Instruction-Multiple-Data (SIMD) registers, which allows for ultra-low-latency lookups in a batch. We leverage this unique capability to propose NoMAD-Attention, an efficient attention algorithm that replaces MAD operations with in-register lookups. Through hardware-aware algorithmic designs, NoMAD-Attention achieves the computation of attention scores using repeated fast accesses to SIMD registers. NoMAD-Attention works with pre-trained attention-based LLMs without model finetuning. Extensive empirical evaluations demonstrate that NoMAD-Attention maintains the quality of the original LLMs well and speeds up the 4-bit quantized LLaMA-7B-based model by up to 2× at 16k context length.

1 Introduction

Auto-regressive transformer-based Large Language Models (LLM) demonstrate remarkable abilities across a wide range of natural language processing tasks without finetuning [35] and exhibit emergent abilities [49] for solving complex problems.

The Need for Deploying LLM on CPUs. Despite the potential of LLMs, their deployment is costly [27]. Serving LLMs with billion-scale parameters requires specialized hardware such as NVIDIA A100 Graphics Processing Units (GPUs) [55]. However, mainstream personal devices are predominately equipped with only Central Processing Units (CPUs) [43]. As a result, making LLM-related services accessible to everyone remains a major challenge. Reducing the LLM inference latency on CPUs would significantly influence its accessibility and adoption.

Expensive Multiply-add Operations for Attention in LLM Inference. LLM inference on CPUs is compute-bound, and the primary computational bottleneck is the calculation of attention scores [21]. Attention, a mechanism that models token interactions through all-pair dot products, heavily

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

 https://doi.org/10.52202/079017-3581

relies on the multiply-add (MAD) kernel on processors. The MAD operation involves computing the product of two numbers and adding that product to an accumulator [44]. MAD plays a crucial role in determining the attention score between tokens and subsequently blending their embeddings within the attention mechanism. The computational cost of attention grows quadratically with the sequence length due to the cumulative MAD operations. Since CPUs have limited parallel cores, they are inefficient for handling highly repetitive and parallel workloads. The extensive MAD operations required by the attention mechanism thus become the primary bottleneck during inference.

Opportunities and Challenges from Modern CPUs: In-Register Lookups. The memory hierarchy of modern CPUs has undergone significant evolution, introducing a new type of registers optimized for Single-Instruction-Multiple-Data (SIMD) operations. The SIMD registers vary in size, ranging from 128 bits to 512 bits [41], and support specialized SIMD instructions for high-throughput parallel processing [59]. SIMD registers have become a standard feature in commodity hardware, including laptops and mobile devices [13]. In this context, an in-register lookup (or shuffle) refers to the low-latency random access of information stored within SIMD registers. Storing information such as dot-product lookup tables (LUT) within SIMD registers, as opposed to cache memory, can accelerate LLM inference [4]. However, the limited size of SIMD registers poses challenges to fitting the computational paradigm of existing models.

Our Proposal: MAD-Free Attention with In-Register Lookups. Our paper demonstrates a new approach for speeding up LLM inference by leveraging the unique hardware capability of CPUs. We show how the vast quantities of MAD operations in attention computation can be replaced with in-register lookups to mitigate the quadratic computational bottleneck of LLM inference on CPUs. NoMAD-Attention significantly speeds up LLM inference without sacrificing model quality and is compatible with pre-trained attention-based transformers without finetuning.

We summarize our contributions as follows:

1. We identify the extensive MAD operations in attention as the bottleneck of CPU LLM inference and we replace them with fast in-register lookups.
2. We introduce NoMAD-Attention, a MAD-free framework of attention computation for pre-trained attention-based LLMs. NoMAD-Attention leverages hardware-aware algorithmic designs to enable accurate and fast in-register lookup-based estimations of query-key dot products despite the limited capacity of SIMD registers. NoMAD-Attention preserves model quality while yielding considerable speedups over MAD-based attention.
3. Our extensive experiments demonstrate that NoMAD-Attention achieves up to $2\times$ speedup on LLaMA-7B-based models with 4-bit weights at a context length of 16k while maintaining the predictive performance of the original model.

2 LLM Inference on CPUs

This section introduces the attention mechanism used in LLMs and the key-value (KV) caching technique for avoiding redundant attention computations. We also discuss the CPU memory hierarchy, which motivates the use of fast in-register lookups.

2.1 LLM Attention

Most LLMs are decoder-only attention-based models that are pre-trained on a next token prediction objective. LLMs use masked self-attention, which allows LLMs to cache key and value embeddings to avoid future recomputations. However, this comes at the cost of memory overhead. The autoregressive generation of LLMs consists of two phases: 1. *prompt processing*: the sequence of token embeddings in the prompt is fed through by the model, and their key-value embeddings are cached by the model; and 2. *decoding*: a new token is sampled based on the output embedding of the last token, and the embedding of the new token is fed through the model, the output of which becomes the basis for sampling the next token. The decoding process continues until an end-of-sequence token `<EOS>` is sampled.

At the decoding step t, a single-head masked self-attention computes its output in the following way. The embedding of the current token e^t is transformed into key, query, and value embeddings through distinct transformations $k^t = f_K(e^t), q^t = f_Q(e^t), v^t = f_V(e^t)$.

https://doi.org/10.52202/079017-3581

Algorithm 1 Attention Score Computation in LLM

1: **Input:** query q^t, key k^t, key cache K_{cache}^{t-1}
2: let $K_{\text{cache}}^t \leftarrow \begin{bmatrix} K_{\text{cache}}^{t-1} \\ k^t \end{bmatrix}$ ▷ Append the current key to key cache
3: return $\text{softmax}(\frac{q^t (K_{\text{cache}}^t)^\top}{\sqrt{d}})$

Then, the key and value embedding of the current token are appended to the key and value cache, respectively. The KV cache $K_{\text{cache}}^{t-1}, V_{\text{cache}}^{t-1}$ of the step $t-1$ contains the key/value embeddings of all previous tokens, and after appending, the KV cache become

$$K_{\text{cache}}^t = \begin{bmatrix} K_{\text{cache}}^{t-1} \\ k^t \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ \dots \\ k^t \end{bmatrix}, V_{\text{cache}}^t = \begin{bmatrix} V_{\text{cache}}^{t-1} \\ v^t \end{bmatrix} = \begin{bmatrix} v^1 \\ v^2 \\ \dots \\ v^t \end{bmatrix}$$

Finally, the attention output is computed as

$$\text{attention}(e^t) = \text{softmax}\left(\frac{q^t (K_{\text{cache}}^t)^\top}{\sqrt{d}}\right) V_{\text{cache}}^t$$

where d is the dimensionality of q^t. We will refer to the result of $\text{softmax}(\frac{qK^\top}{\sqrt{d}})$ as the attention scores since they dictate how much "attention" each token pays to other tokens. Computations in the prompt processing phase are similar to those in the decoding phase, except all the prompt tokens are computed in batch. LLMs use multi-head attention, which transforms the concatenation of the outputs of multiple single-head attentions to form an output embedding.

MAD-based Attention. The attention mechanism models the interaction between tokens by performing all-pair dot products, where each dot product is computed via d Multiply-Add (MAD) operations. Since attention computes the interaction between all pairs of tokens, the amount of MAD operations scales quadratically with the sequence length, quickly overwhelming the computing capability of CPUs. CPUs are designed to handle complex workloads with granular control, while GPUs are optimized for processing simple and repetitive tasks in high throughput. Hence, the success of attention has largely been fueled by the development of highly parallel throughput-oriented processors, such as GPUs [12].

MAD-based Attention as Bottleneck of LLM Inference. The computation of attention scores becomes the bottleneck of LLM inference as the sequence length increases (see our analysis in Figure 4). At the t-th step of the decoding phase, the time complexity of computing attention score with MAD is $O(t)$ due to t dot products, while all other components of LLMs such as MLP, skip connections, and normalization have a time complexity of $O(1)$. We will focus on optimizing the efficiency of attention score computations in our proposed approach. Algorithm 1 presents the pseudocode for attention score computation, including key caching, for a single-head masked self-attention in LLM. This algorithm will serve as a point of comparison in our proposed approach.

2.2 Memory Hierarchy of Modern CPUs

CPU memory is organized into a pyramidal hierarchy, as shown in Figure 1, with faster memory significantly smaller than slower memory. The memory unit with the fastest access speed is the registers. Each compute core accesses its dedicated registers in just 1–2 CPU cycles, but these registers are limited in size, usually not exceeding 64 bits. Modern processors include a new type of registers optimized for Single-Instruction-Multiple-Data (SIMD) operations. These SIMD registers range from 128 bits to 512 bits in size and support a limited set of SIMD instructions for throughput-oriented parallel processing. SIMD registers are common on commodity hardware, including laptops and mobile devices. Using SIMD operations can speed up deep learning models on CPUs by parallelizing matrix multiplications. However, due to the limited number of cores in a CPU, its efficiency in deep learning is still considerably worse than GPU. Prior works [42, 9] have resorted to sparsity and sampling-based approaches, but they require training models from scratch and may not apply to all architectures. Our approach exploits the SIMD registers to shift the computation paradigm from MAD to in-register lookups, which demonstrates significant speedup over MAD-based models.

 https://doi.org/10.52202/079017-3581

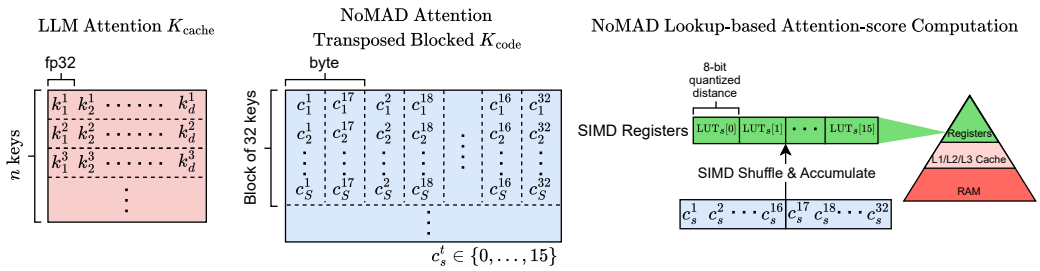

Figure 1: A comparison of memory layouts of the key cache of LLM attention (left) and the key-code cache of NoMAD-Attention (middle), and an illustration of how attention scores are computed through in-register lookups in NoMAD-Attention (right).

When describing our proposed algorithm, we assume SIMD registers are 128 bits wide. Some systems with wider SIMD registers support more parallelism, e.g., 256-bit registers in AVX-2 and 512-bit registers in AVX-512. However, the most universal form of SIMD registers uses 128 bits, which is supported by Arm NEON and AVX-compatible processors.

3 Methodology

This section describes our proposed approach, NoMAD-Attention, which replaces MAD operations with in-register lookups to enable faster attention computations on CPUs. NoMAD-Attention utilizes three techniques to enable lookup-based attention: 1. transforming dot product computations to memory lookups through product quantization, 2. compressing lookup tables into SIMD registers for low-latency access, 3. reorganizing the memory layout of key cache for batch parallel dot product lookups.

3.1 Transforming Dot-products into Lookups

Previous works have shown that inexact attention scores in transformers work well for sequence modeling [53]. NoMAD leverages Product Quantization (PQ) [23] to compute high-quality estimations of dot products through in-register lookups. PQ, originally designed for compressing high-dimensional vectors, quantizes a floating-point vector into discrete codes. It makes use of *sub-quantizers*; for a d-dimensional vector space, a vector is divided evenly in dimension into S sub-vectors, where each sub-vector has dimension $d_{\text{sub}} = \frac{d}{S}$, and each sub-vector space is quantized independently. We use $\pi_s(e)$, where $s \in \{1 \ldots S\}$, to denote the function that maps a d-dimensional vector e to its d_{sub}-dimensional sub-vector of the s-th sub-quantizer. *Codebooks* are used to quantize sub-vectors to codes, which are collections of cluster centroids learned from a set of calibration vectors. We use $b_{s,c}$ to denote the c-th centroid in the codebook of the s-th sub-quantizer. For a given vector e, the product-quantized codes of e, denoted $c_1, \ldots, c_S$, are the indexes of the nearest centroid of each sub-quantizer, i.e.,

$$\text{PQ}(e) = [c_1 \ldots c_S], \text{where } c_s = \arg\min_c \left\| \pi_s(e) - b_{s,c} \right\|$$

Once base vectors have been product-quantized to codes, PQ leverages asymmetric distance computation to keep estimation errors low. In the computed distances, the original query vector is used while the quantized base vectors are used, hence the asymmetry. For a given query q, the distances to the centroids of each sub-quantizer are computed and stored in a lookup table (LUT). Then, the corresponding distances in the LUT are looked up based on the codes of base vectors and accumulated to produce the final distance estimation. More concretely, denoting the distance between query q and the c-th centroid for the s-th sub-quantizer using $\text{LUT}_s[c] = \text{dist}\big(\pi_s(q), b_{s,c}\big)$, then the estimated distance between query q and a product-quantized base vector e, where $\text{PQ}(e) = [c_1 \ldots c_S]$, is

$$\widetilde{\text{dist}}(q, e) = \sum_{s=1}^{S} \text{LUT}_s[c_s]$$

Building upon previous work [7], we employ product quantization to approximate dot products within the attention mechanism. We product-quantize the key vectors in attention to produce key codes, which will be stored in place of the key cache of LLM attention. The codebooks are learned by performing clustering on a set of key vectors from a calibration set. The key vectors are quantized to the nearest centroid with respect to L2 distance. For a given query, the query-dependent LUT is computed to hold dot products with respect to centroids. Dot products of sub-vectors are retrieved from the LUT based on key codes and accumulated to produce the final dot product estimates. This procedure allows us to compute attention scores through lookups.

3.2 Compressing Lookup Tables into SIMD Registers

Estimating dot products through PQ mostly eliminates the use of MAD kernels in the computation of attention scores. However, this approach yields limited speedup over dot-product attention since a high proportion of the CPU cycles are wasted due to cache/memory access stalling. L1-cache-resident LUT is not enough to offer high-performance PQ [3]. The full potential of lookup-based attention can only be unlocked by having the LUT stored in registers, which take only 1-2 CPU cycles to access. However, the highly limited size of registers poses a challenge to fitting the LUT. In PQ, each sub-quantizer commonly uses 256 centroids, which translates to 8-bit codes. Combined with 32-bit floating-point (FP32) dot products, the LUT for each sub-quantizer consumes 8192 bits of memory, while the SIMD registers are only 128 bits wide. We leverage hardware-aware techniques proposed in [4] to enable low-latency retrieval from register-resident LUT.

8-bit Quantized Dot Products in LUT Due to the 128-bit width of SIMD registers, the FP32 representation of dot product is too costly to store. Adopting FP32 dot products in LUT implies that each codebook can only contain up to 4 centroids, which will lead to significant quantization errors. Therefore, we adopt the 8-bit dynamically quantized representation of dot products. Compressing beyond 8-bit is infeasible since most SIMD instruction sets do not support parallel lookups below 8 bits. The quantization is done dynamically for each query to minimize quantization errors. For a given query and sub-quantizer, dot products to centroids are first computed in full FP32 precision. Then, the quantization range is determined by the minimum and maximum dot products to the centroids. Finally, the range is evenly divided into 2^8 buckets, and dot products are quantized to the bucket they fall into. More formally, suppose $\mathrm{dp}_{\min} = \min_c(\pi_s(q) \cdot b_{s,c})$ and $\mathrm{dp}_{\max} = \max_c(\pi_s(q) \cdot b_{s,c})$ are the minimum and maximum dot products of the query q to the centroids of the s-th sub-quantizer, then the LUT stores the quantized dot products to centroid c as

$$\mathrm{LUT}_s[c] = \left\lfloor \frac{(\pi_s(q) \cdot b_{s,c}) - \mathrm{dp}_{\min}}{(\mathrm{dp}_{\max} - \mathrm{dp}_{\min})/(2^8 - 1)} \right\rfloor \tag{1}$$

The quantization and de-quantization process can be done efficiently without much computational overhead, and the quantization error is kept low thanks to dynamic query-dependent quantization (we analyze its effects in Table 3).

Constrained Codebook Size By adopting 8-bit quantized dot products in LUT, we can fit 16 dot products on 128-bit SIMD registers. This implies that the codebook size of each sub-quantizer is constrained to 16 centroids, and evidence suggests this limited size may work well with attention: it has been shown that the output of attention loses rank extremely quickly [15], implying that the intermediate embeddings of transformers may exhibit clear clustering structures.

3.3 Reorganizing Key Cache Memory Layout

Quantized dot products and constrained codebooks enable LUT to be stored in SIMD registers, but the layout format of the key cache needs to be reorganized for SIMD instructions. The original key cache in LLM attention stores each key vector contiguously in a row to optimize single vector reads. NoMAD-Attention uses the key-code cache in place of the key cache, which stores the quantized codes of keys. To allow fast lookups of LUT entries based on key codes, we store the key codes in a transposed blocked format. A comparison between the LLM key cache and the NoMAD key-code cache is given in Figure 1.

The storage format of the NoMAD key-code cache is *transposed*: stored in column-major order instead of row-major, and *blocked*: with 32 keys as a block. The SIMD instruction `shuffle`, leveraged for performing low-latency batch lookups, takes a batch of byte-size integers as input and

 https://doi.org/10.52202/079017-3581

Algorithm 2 NoMAD-Attention Score Computation

Input: query q^t, key k^t, key-code cache K_{code}^{t-1}
let $c_s^t \leftarrow \arg\min_{c \in \{0,\ldots,15\}} \|\pi_s(k^t), b_{s,c}\|_2$ for $s = 1 \ldots S$ ▷ Compute codes for the current key
let $K_{\text{code}}^t \leftarrow$ insert c_s^t into K_{code}^{t-1} for $s = 1 \ldots S$
▷ Insert codes of the current key into the key-code cache
let $\text{LUT}_s[c] \leftarrow \text{quantize}(\pi_s(q^t) \cdot b_{s,c})$ for $s = 1 \ldots S, c = 0 \ldots 15$
▷ Store 8-bit quantized dot products (Equation 1) in LUT
let $\text{accu}[1 \ldots t] \leftarrow 0$ ▷ Initialize accumulators
for $i \leftarrow 1 \ldots \lceil \frac{t}{32} \rceil$ **do** ▷ Perform in-register lookups in batch of 32 keys
  for $s \leftarrow 1 \ldots S$ **do**
    `simd_load`(LUT_s) ▷ Load LUT into registers
    $\text{accu}[32i - 31 \ldots 32i] \leftarrow$
      `simd_add`$\big(\text{accu}[32i - 31 \ldots 32i],$ `simd_shuffle`$(\text{LUT}_s, K_{\text{code}}^{32i-31\ldots32i,s})\big)$
  end for
end for
return $\text{softmax}(\frac{\text{dequantize}(\text{accu}[1\ldots t])}{\sqrt{d}})$

retrieves the values held in the registers corresponding to the integer indices. The original storage format of the key cache stores all dimensions of a key contiguously, which precludes efficient use of `shuffle`. To maximize the usage of the LUT held in registers, we store key codes belonging to the same sub-quantizer contiguously in rows of 32 codes. Since `shuffle` performs lookups in a batch size of 16, the keys within the same block are stored in alternating order. Each quantized code occupies half a byte as there are 16 centroids in a codebook, while the `shuffle` instruction uses each byte as an input argument. By performing SIMD bit-shifting and bit-masking on a block of alternating keys, we obtain the key codes in the original order, ready for use with `shuffle`. Section A in the Appendix describes additional details on how `shuffle` is performed on each block of key-code cache, and provides pseudocode.

3.4 NoMAD-Attention

By combining these three techniques, NoMAD-Attention achieves fast MAD-free attention score computations through SIMD in-register lookups. For a given query, first, LUTs with 8-bit quantized dot products are computed for each sub-quantizer. Then, an LUT is loaded into registers, followed by SIMD `shuffle` instructions to retrieve dot products in the LUT in batch based on key codes. The loading and lookup are repeated for all sub-quantizers, and the retrieved dot products are accumulated in batch through SIMD `add`. Finally, the quantized dot products accumulated over all sub-quantizers are de-quantized, scaled, and fed through softmax to produce the attention scores. The pseudocode for NoMAD-Attention score computations is given in Algorithm 2.

3.5 Learning Key Compression

Compressing each segment of key activation embeddings into 4-bit codes, without degrading model quality, is challenging. The straightforward approach is to first cache key activation embeddings on a calibration dataset, and then learn the centroids through k-means clustering [29] on the embeddings. However, this first-cut approach significantly degrades model quality, especially for $d_{\text{sub}} > 1$ (we analyze its effects in Section 4.2). The first-cut approach performs sub-optimally since it is uninformed; clustering is performed with the aim to minimize the reconstruction error with all tokens. However, as shown previously [60, 28], the key cache of certain tokens are more pivotal for preserving model quality. Therefore, we leverage the Fisher Information Matrix (FIM) to bias the centroids towards important activations. Specifically, we approximate the Hessian using diagonals of the FIM, which is the element-wise square of the gradient, and use it to weigh the reconstruction errors. Using the FIM to minimize reconstruction error was first proposed in [26]. Our optimization objective for

Table 1: Perplexity on WikiText-2 and C4 and accuracy on 6 benchmarks of LLMs with Attention and NoMAD-Attention.

		WT-2↓	C4↓	SciQ↑	Arc-E↑	Arc-C↑	Hellaswag↑	WinoGrande↑	PIQA↑	**Avg.↑**
LLaMA-7b	Attention	5.68	7.08	94.6	75.21	41.89	56.93	70.09	78.67	69.57
	NoMAD-Attention ($d_{\text{sub}} = 1$)	5.74	7.14	94.9	75.34	41.81	56.57	70.56	78.56	69.62
	NoMAD-Attention ($d_{\text{sub}} = 2$)	6.11	7.56	93.3	73.65	38.65	54.45	67.56	77.86	67.58
	NoMAD-Attention ($d_{\text{sub}} = 4$)	9.23	12.66	84.4	66.41	32.59	46.74	59.75	74.32	60.70
LLaMA-13b	Attention	5.09	6.61	95.0	77.40	46.42	59.93	72.85	79.16	71.79
	NoMAD-Attention ($d_{\text{sub}} = 1$)	5.14	6.65	95.1	77.15	46.67	59.74	72.85	79.22	71.79
	NoMAD-Attention ($d_{\text{sub}} = 2$)	5.44	6.96	94.8	76.43	44.37	58.05	71.82	78.18	70.61
	NoMAD-Attention ($d_{\text{sub}} = 4$)	8.19	10.66	89.1	73.53	37.71	51.19	60.93	76.71	64.86
LLaMA-2-7b	Attention	5.47	6.97	94.0	76.30	43.43	57.16	69.06	78.07	69.67
	NoMAD-Attention ($d_{\text{sub}} = 1$)	5.53	7.02	93.9	75.80	42.92	56.71	69.30	77.97	69.43
	NoMAD-Attention ($d_{\text{sub}} = 2$)	5.97	7.54	93.2	73.48	39.76	54.71	67.96	76.93	67.67
	NoMAD-Attention ($d_{\text{sub}} = 4$)	10.19	13.24	85.1	68.77	33.70	47.11	56.51	74.27	60.91
LLaMA-2-13b	Attention	4.88	6.47	94.6	79.42	48.46	60.07	72.30	79.00	72.31
	NoMAD-Attention ($d_{\text{sub}} = 1$)	4.92	6.50	94.8	79.00	47.87	59.81	71.51	78.94	71.99
	NoMAD-Attention ($d_{\text{sub}} = 2$)	5.24	6.85	94.2	77.61	45.39	58.46	70.72	77.75	70.69
	NoMAD-Attention ($d_{\text{sub}} = 4$)	8.22	10.86	90.8	72.81	37.80	51.24	55.96	76.33	64.16

learning the centroids for the s-th sub-quantizer is

$$b^\star_{s,0} \dots b^\star_{s,15} = \underset{b_{s,0} \dots b_{s,15} \in \mathbb{R}^{d_{\text{sub}}}}{\arg\min} \sum_{c=0}^{15} \sum_{i=1}^{n} w_i^c \left\| \pi_s(k_i) - b_{s,c} \right\|_2^2,$$
$$\text{where } w_i^c = \begin{cases} \overbrace{\text{grad}(\pi_s(k_i))^\top \text{grad}(\pi_s(k_i))}^{\text{partial sum of diagonals of FIM}} & \text{if } c = \arg\min_a \|\pi_s(k_i) - b_{s,a}\|_2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $k_1 \dots k_n$ are cached key activation embeddings on the calibration dataset. We use weighted k-means++ [5] to optimize the objective.

4 Experiments

This section evaluates the effectiveness of our proposed NoMAD-Attention in preserving model quality and speeding up LLM inference on CPUs. We first introduce the software, hardware, models, benchmarks, and baselines, then provide detailed results and discussions, and finally perform an ablation study to validate each component of our proposal.

Software and Hardware Our implementation of NoMAD-Attention is built in C and C++, based on the open-source projects llama.cpp [20] and FAISS [16]. We also built a GPU implementation of NoMAD-Attention for quick prototyping and key-compression learning, which is based on PyTorch [33] and HuggingFace Transformers [51]. Experiments for latency and throughput are performed on a Linux server equipped with an Intel Xeon E5-2695 V3 14-core CPU, which supports AVX2 SIMD instructions, and 512GB of DDR4 RAM. Experiments for accuracy and perplexity are performed on two NVIDIA A100-40GB GPUs.

Models and Benchmarks We evaluate the quality of NoMAD-Attention with 4 popular LLMs: 1. LLaMA-7b 2. LLaMA-13b [45] 3. LLaMA-2-7b 4. LLaMA-2-13b [46]. We measure the model quality with perplexity on WikiText-2 [30] and C4 [14] at the context length of 2048, and zero-shot accuracy (using the default configurations of lm-evaluation-harness [19]) on SciQ [50], Arc Easy (Arc-E), Arc Challenge (Arc-C) [11], Hellaswag [54], WinoGrande [38], and PIQA [6]. The centroids for key compression of NoMAD-Attention are learned on a calibration set of 16 sequences from WikiText-2, each with 2048 tokens. To test the model efficiency, we benchmark the latency and throughput of CodeLlama-7b [37] (with 16-bit weights and 4-bit q4_0 quantized weights), which has a longer context length of 16,384 than the LLaMA family of models. We compare the efficiency of NoMAD-Attention-based models (with $d_{\text{sub}} \in \{1, 2, 4\}$) against Attention-based models with a llama.cpp-based implementation.

https://doi.org/10.52202/079017-3581

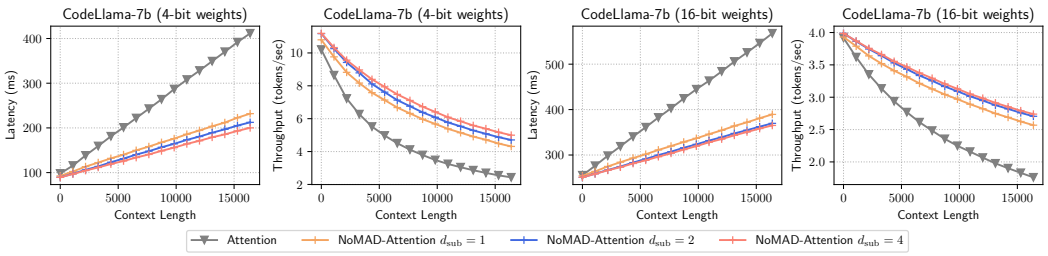

Figure 2: Latency and throughput of decoding for CodeLlama-7b (4-bit and 16-bit weights) with Attention and NoMAD-Attention. NoMAD-Attention achieves 1.78–2.07× higher throughput than Attention with 4-bit CodeLlama-7b, and 1.46–1.56× higher throughput with 16-bit CodeLlama-7b.

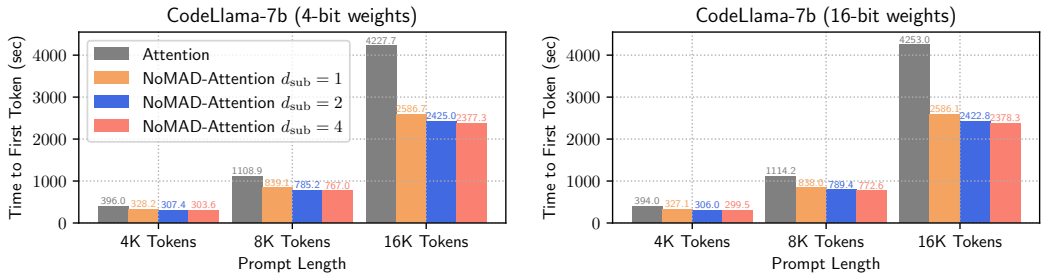

Figure 3: Time for processing prompts of different lengths for CodeLlama-7b (4-bit and 16-bit weights) with Attention and NoMAD-Attention. NoMAD-Attention achieves 1.63–1.79× speedup over Attention to process a prompt of 16k tokens.

4.1 Results

Model Quality Table 1 presents the perplexity and accuracy of NoMAD-Attention on different benchmarks with comparisons to Attention. NoMAD-Attention ($d_{\text{sub}} = 1$) incurs negligible perplexity and accuracy loss, $d_{\text{sub}} = 2$ incurs minimal degradation, and $d_{\text{sub}} = 4$ incurs some degradation due to the high key compression factor.

Decoding Efficiency The latency and throughput for decoding 16,384 tokens for NoMAD-Attention-based and Attention-based CodeLlama-7b are presented in Figure 2. For 4-bit CodeLlama-7b, NoMAD-Attention achieves 1.78×, 1.95×, and 2.07× higher throughput (tokens per second) than Attention for $d_{\text{sub}} = 1, 2, 4$, respectively. For 16-bit CodeLlama, NoMAD-Attention achieves 1.46×, 1.54×, and 1.56× higher throughput (tokens per second) than Attention for $d_{\text{sub}} = 1, 2, 4$, respectively.

Prompt Processing Efficiency We investigate the efficiency of NoMAD-Attention for prompt processing. We generate prompts of length 4,000, 8,000, and 16,000 tokens and record the time until the first token is decoded. Figure 3 presents prompt processing time for CodeLlama-7b. At the prompt length of 16k tokens, NoMAD-Attention speeds up prompt processing time by 1.63 – 1.78× for the 4-bit model, and 1.64 – 1.79× for the 16-bit model. Prompt processing times are similar for 16-bit and 4-bit model, since weight quantization is ineffective for speeding up batch processing.

4.2 Ablation Study

We perform a set of ablative experiments to study the latency breakdown, and the effects of FIM-informed clustering and 8-bit LUT quantization on model quality.

Latency Breakdown We investigate the makeup of latency in each decoding step. Figure 4 presents the latency breakdown of 4-bit CodeLlama-7b for decoding 16,384 tokens, and Figure 7 in the appendix shows the latency breakdown for 16-bit CodeLlama-7b. The latency of linear projections

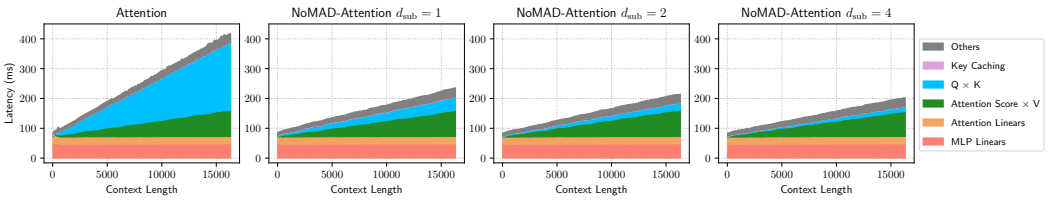

Figure 4: The breakdown of decoding latency of 4-bit CodeLlama-7b for Attention and NoMAD-Attention. NoMAD-Attention effectively reduces the latency of computing query-key dot-products by 5.24–14.77× over Attention.

in MLP and Attention stays constant as context length increases, and the latency of key caching is insignificant compared to other components. The latency of dot-products between queries and keys, as well as the multiplications between attention scores and values, grows linearly with increasing context length. In the original Attention, computing attention scores quickly becomes the latency bottleneck as context length increases. NoMAD-Attention mitigates this bottleneck by speeding up query-key dot-product computation by 5.24×, 9.75×, and 14.77× for $d_{\text{sub}} = 1, 2, 4$, respectively.

FIM-Informed Clustering We study the effects of FIM-informed centroid learning for preserving model quality. Table 2 presents the perplexity of NoMAD-Attention-based LLaMA-7b on WikiText-2 with uninformed centroids and FIM-informed centroids. FIM-informed centroids consistently achieve better model quality, shown by the lower perplexity.

LUT Quantization We examine the effects of 8-bit quantized dot products in LUT on model quality. Table 3 presents the perplexity of NoMAD-Attention-based LLaMA-7b on WikiText-2 with 8-bit quantized LUT and 32-bit unquantized LUT. 8-bit quantization of LUT incurs negligible loss in perplexity.

Table 2: Ablative experiments on the effects of FIM-informed centroid learning on the perplexity of LLaMA-7b on WikiText-2.

	Perplexity ↓	
	Uninformed	FIM-informed
NoMAD-Attention ($d_{\text{sub}} = 1$)	5.76	5.74
NoMAD-Attention ($d_{\text{sub}} = 2$)	7.05	6.11
NoMAD-Attention ($d_{\text{sub}} = 4$)	21.39	9.23
Attention	5.68	

Table 3: Ablative experiments on the effects of quantized LUT on the perplexity of LLaMA-7b on WikiText-2.

	Perplexity ↓	
	8-bit LUT	32-bit LUT
NoMAD-Attention ($d_{\text{sub}} = 1$)	5.74	5.74
NoMAD-Attention ($d_{\text{sub}} = 2$)	6.11	6.10
NoMAD-Attention ($d_{\text{sub}} = 4$)	9.23	9.23
Attention	5.68	

5 Related Works

Approximate Attention and Efficient Transformer Since the introduction of attention in transformers [47], there has been a body of work on approximating the attention mechanism for efficient training and inference. Dynamically sparse attention was achieved using LSH [24], Nyström method [52], and random sampling [53]. Low-rank attention has also been extensively explored [48, 10, 8] and shown to have compute- and memory-efficiency advantages. Hardware-aware attention mechanisms such as FlashAttention [12] propose to mitigate the IO bottleneck in GPUs. In large language models, multiple approaches [60, 28, 58] have been proposed to reduce the high memory overhead of the KV cache. For CPU-only environments, [39] proposes to speed up LLM inference through weight quantization. Weight quantization [18, 56] is an effective approach for accelerating LLM inference and fine-tuning [57] by mitigating the IO bottleneck.

Matrix Multiplication Optimization and Compression Approximate matrix multiplication is applicable in a wide range of computational problems, and its optimization has been a topic of interest for years [32]. Modern researchers have begun optimizing matrix multiplication around the specific limitations of computers, including mitigating the IO bottleneck between the CPU and main memory [25]. Compression techniques were developed to accelerate large-scale matrix multiplications and

https://doi.org/10.52202/079017-3581

scale up the size of multiplication [31, 7, 25, 1]. However, many of these algorithms still had limitations that make them too inaccurate or costly for approximating matrix multiplications in LLMs [7, 25].

6 Conclusion

This study addresses the challenges of large language model inference on Central Processing Units (CPUs), particularly the difficulties associated with the expensive Multiply-Add (MAD) matrix operations in attention mechanisms. The investigation highlighted the untapped potential of Single-Instruction-Multiple-Data (SIMD) registers and their fast in-register lookup capabilities within CPUs. The proposed NoMAD-Attention algorithm serves as an efficient alternative to traditional MAD-based approaches, leveraging in-register lookups and optimizing memory access to SIMD registers. The implementation of NoMAD-Attention resulted in a significant acceleration of LLaMA-7B-based model inference, achieving up to a $2\times$ speedup on CPUs.

Acknowledgements

This work was supported by National Science Foundation SHF-2211815, Ken Kennedy Institute, and grants from Adobe and VMware.

Limitations and Broader Impacts

Our proposed method is targeted towards CPUs with SIMD capabilities, and may not generalize to other types of processors. We make LLMs more accessible on commodity hardware, which contributes to the democratization of artificial intelligence and reduces the carbon footprints. Other than the negative societal impacts already presented by LLMs, we expect no additional negative impacts from our work.

References

[1] Ahmed F AbouElhamayed, Angela Cui, Javier Fernandez-Marques, Nicholas D Lane, and Mohamed S Abdelfattah. Pqa: Exploring the potential of product quantization in dnn hardware acceleration. *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

[2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, 2023.

[3] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *42nd International Conference on Very Large Data Bases*, volume 9, page 12, 2016.

[4] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Accelerated nearest neighbor search with quick adc. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pages 159–166, 2017.

[5] David Arthur, Sergei Vassilvitskii, et al. k-means++: The advantages of careful seeding. In *Soda*, volume 7, pages 1027–1035, 2007.

[6] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[7] Davis Blalock and John Guttag. Multiplying matrices without multiplying. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 992–1004. PMLR, 18–24 Jul 2021.

[8] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34:17413–17426, 2021.

[9] Beidi Chen, Tharun Medini, James Farwell, Charlie Tai, Anshumali Shrivastava, et al. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *Proceedings of Machine Learning and Systems*, 2:291–306, 2020.

[10] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[11] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[12] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[13] Ganesh Dasika, Mark Woh, Sangwon Seo, Nathan Clark, Trevor Mudge, and Scott Mahlke. Mighty-morphing power-simd. In *Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems*, pages 67–76, 2010.

[14] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

[15] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International Conference on Machine Learning*, pages 2793–2803. PMLR, 2021.

[16] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

[17] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[18] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Optq: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2022.

[19] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023.

[20] Georgi Gerganov. GitHub - ggerganov/llama.cpp: LLM inference in C/C++ — github.com. `https://github.com/ggerganov/llama.cpp`, 2023.

[21] Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David P Woodruff, and Amir Zandieh. Hyperattention: Long-context attention in near-linear time. *arXiv preprint arXiv:2310.05869*, 2023.

[22] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[23] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

 https://doi.org/10.52202/079017-3581

[24] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[25] Sudhindra Gopal Krishna, Aditya Narasimhan, Sridhar Radhakrishnan, and Richard Veras. On large-scale matrix-matrix multiplication on compressed structures. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 2976–2985. IEEE, 2021.

[26] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.

[27] Zheng Lin, Guanqiao Qu, Qiyuan Chen, Xianhao Chen, Zhe Chen, and Kaibin Huang. Pushing large language models to the 6g edge: Vision, challenges, and opportunities. *arXiv preprint arXiv:2309.16739*, 2023.

[28] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023.

[29] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[30] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[31] Michael Nelson, Sridhar Radhakrishnan, and Chandra N Sekharan. Billion-scale matrix compression and multiplication with implications in data mining. In *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 395–402. IEEE, 2019.

[32] Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)*, 5(3):1–17, 2013.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[36] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

[37] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[38] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[39] Haihao Shen, Hanwen Chang, Bo Dong, Yu Luo, and Hengyu Meng. Efficient llm inference on cpus. *arXiv preprint arXiv:2311.00502*, 2023.

[40] Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. Language models are multilingual chain-of-thought reasoners, 2022. *URL https://arxiv. org/abs/2210.03057*.

[41] Seo-Ran Shin, Se-Yeon Choo, and Joon-Sang Park. Accelerating random network coding using 512-bit simd instructions. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1099–1103. IEEE, 2019.

[42] Ryan Spring and Anshumali Shrivastava. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 445–454, 2017.

[43] Yifan Sun, Nicolas Bohm Agostini, Shi Dong, and David Kaeli. Summarizing cpu and gpu design trends with product data. *arXiv preprint arXiv:1911.11313*, 2019.

[44] Seunghwan Sung, Sujin Hur, Sungwoo Kim, Dongho Ha, Yunho Oh, and Won Woo Ro. Mad macce: Supporting multiply-add operations for democratizing matrix-multiplication accelerators. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 367–379, 2023.

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

[46] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

[48] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[49] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[50] Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

[51] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[52] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14138–14148, 2021.

[53] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

[54] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[55] Hengrui Zhang, August Ning, Rohan Prabhakar, and David Wentzlaff. A hardware evaluation framework for large language model inference. *arXiv preprint arXiv:2312.03134*, 2023.

[56] Tianyi Zhang and Anshumali Shrivastava. Leanquant: Accurate and scalable large language model quantization with loss-error-aware grid. *arXiv preprint arXiv:2407.10032*, 2024.

[57] Tianyi Zhang, Junda Su, Oscar Wu, Zhaozhuo Xu, and Anshumali Shrivastava. Spallm: Unified compressive adaptation of large language models with sketching. *arXiv preprint arXiv:2410.06364*, 2024.

 https://doi.org/10.52202/079017-3581

[58] Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *arXiv preprint arXiv:2405.03917*, 2024.

[59] Weihua Zhang, Zhaofeng Yan, Yuzhe Lin, Chuanlei Zhao, and Lu Peng. A high throughput b+ tree for simd architectures. *IEEE Transactions on Parallel and Distributed Systems*, 31(3):707–720, 2019.

[60] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

Appendix / Supplemental Material

A Details Regarding SIMD Instructions

The SIMD `shuffle` presented in Algorithm 2 is a simplification of the actual hardware implementation. We give full details of lines 5 to 11 in Algorithm 2 in Algorithm 3. Keys are stored in blocks of 32, in which keys are stored in an alternating order (see Figure 1 for an illustration). After a LUT is loaded into registers, the row of key codes in the block corresponding to the sub-quantizer is used to perform `shuffle`. First, each byte in the row is bit-shifted to the right by 4 bits via a SIMD instruction, which produces the codes of the first 16 keys in the block. The codes are fed to `shuffle` to retrieve the quantized dot products of the first 16 keys from the LUT. Then, the first 4 bits of each byte in the row are masked out via a SIMD instruction, which produces the code of the last 16 keys in the block. They are similarly used to retrieve the quantized dot products from the LUT. The retrieved quantized dot products of 32 keys are accumulated in the accumulator. Since quantized dot products are 8 bits wide, accumulating them in 8-bit accumulators easily results in overflows. Therefore, 16-bit accumulators are used to accumulate quantized dot products.

Algorithm 3 NoMAD Dot-Product Lookup Accumulation Loop

1: let $\text{accu}[1 \dots t] \leftarrow 0$ ▷ Initialize 16-bit unsigned accumulators
2: **for** $i \leftarrow 1 \dots \lceil \frac{t}{32} \rceil$ **do**
3: **for** $s \leftarrow 1 \dots S$ **do**
4: `simd_load`(LUT_s) ▷ Load LUT into registers
5: let $K_{\text{cache}}^{32i-31 \dots 32i-16,s} \leftarrow$`simd_bitwise_right_shift`$(K_{\text{cache}}^{32i-31 \dots 32i,s}, 4)$ ▷ Obtain the first 15 key codes through bit shifting
6: $\text{accu}[32i-31 \dots 32i-16] \leftarrow$ `simd_add`$\big($ $\text{accu}[32i-31 \dots 32i-16]$, `simd_shuffle`$(\text{LUT}_s, K_{\text{cache}}^{32i-31 \dots 32i-16,s})\big)$
7: let $K_{\text{cache}}^{32i-15 \dots 32i,s} \leftarrow$`simd_bitwise_and`$(K_{\text{cache}}^{32i-15 \dots 32i,s}, \texttt{0xf})$ ▷ Obtain the last 15 key codes through bitwise and
8: $\text{accu}[32i-15 \dots 32i] \leftarrow$ `simd_add`$\big($ $\text{accu}[32i-15 \dots 32i]$, `simd_shuffle`$(\text{LUT}_s, K_{\text{cache}}^{32i-15 \dots 32i,s})\big)$
9: **end for**
10: **end for**

B Visual Explanations on K_{cache} and Lookup Table (LUT) Construction

Figure 5 illustrates the process of mapping and compressing key vector k^t to construct K_{cache}^t. For an input key vector k^t, functions π_s, where $s \in 1 \dots S$, split the vector into sub-vectors $k^t = (\pi_1(k^t), \pi_2(k^t), \dots, \pi_S(k^t))$. Subsequently, each sub-quantizer $\pi_s(k^t)$ is mapped to its nearest centroid c_s^t by referencing the codebook b_s, where $i \in 1 \dots S$, among 16 centroids in the codebook. The resulting values are then stored in the key cache K_{cache}^t.

Similarly, Figure 6 illustrates the process of mapping and compressing query vector q^t to construct the Look-up Tables (LUT). Given a query vector q^t, functions π_s, where $s \in 1 \dots S$, first split the query into sub-queries $q = (\pi_1(q^t), \pi_2(q^t), \dots, \pi_S(q^t))$. Subsequently, the distances between each sub-query $\pi_s(q^t)$ and the 16 centroids from the codebook b_s are computed and then quantized to values within the range of 0–255. Lastly, the quantized vectors are converted into 8-bit codes and stored in LUT_s^t.

C Overhead of Centroid Learning and Storage

Table 4 details the time overhead for learning centroids, which involves saving activations and gradients, and weighted k-means on the saved embeddings. NoMAD-Attention has low learning overheads and can easily scale to larger models.

https://doi.org/10.52202/079017-3581

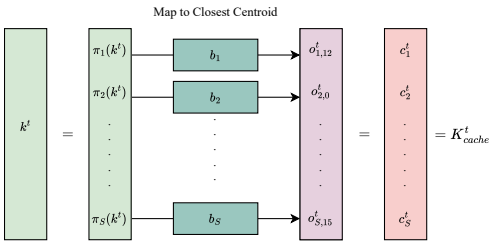

Figure 5: Illustration demonstrating the mapping of an input key k^t to its s-th sub-quantizer using $\pi_s(k^t)$, where $s \in 1 \dots S$. Subsequently, each sub-quantizer maps to its closest centroid c^t_i, where $i \in 1 \dots S$, and the results are stored in the key cache K^t_{cache}.

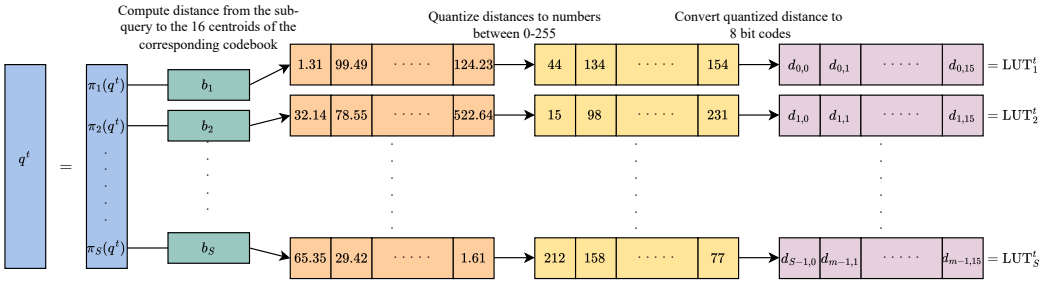

Figure 6: Illustration depicting the mapping of a query vector q to its s-th sub-quantizer using $\pi_s(q)$, where $s \in 1 \dots S$. Subsequently, the distance between $\pi_s(q)$ and 16 centroids is computed. This distance is quantized to a value within the range of 0-255, and the resulting quantized distance is further converted into 8-bit codes, which are stored in LUT_s.

The learned centroids of NoMAD-Attention introduce a small amount of memory overhead. The storage overhead of centroids can be calculated as $l \times h \times d \times 16 \times 4$ bytes, where l represents the number of layers, h the number of attention key heads, d the dimensionality of each attention head, 16 the number of centroids, and 4 the number of bytes for storing each centroid parameter. Therefore, the codebook memory overhead for LLaMA-7b/LLaMA-2-7b with $d_{\text{sub}} = \{1, 2, 4\}$ is 8.4MB, and the memory overhead for LLaMA-13b/LLaMA-2-13b with $d_{\text{sub}} = \{1, 2, 4\}$ is 13.1MB, which is a modest memory footprint compared to the overall model size.

Table 4: Overhead of learning centroids for key compression.

Model	NoMAD Config.	Saving Activations & Gradients	Weighted K-means
LLaMA-7b	$d_{\text{sub}} = 1$	4 mins	27 mins
	$d_{\text{sub}} = 2$	4 mins	14 mins
	$d_{\text{sub}} = 4$	4 mins	7 mins
LLaMA-13b	$d_{\text{sub}} = 1$	8 mins	42 mins
	$d_{\text{sub}} = 2$	8 mins	22 mins
	$d_{\text{sub}} = 4$	8 mins	11 mins
LLaMA-2-7b	$d_{\text{sub}} = 1$	4 mins	27 mins
	$d_{\text{sub}} = 2$	4 mins	14 mins
	$d_{\text{sub}} = 4$	4 mins	7 mins
LLaMA-2-13b	$d_{\text{sub}} = 1$	8 mins	42 mins
	$d_{\text{sub}} = 2$	8 mins	23 mins
	$d_{\text{sub}} = 4$	8 mins	11 mins

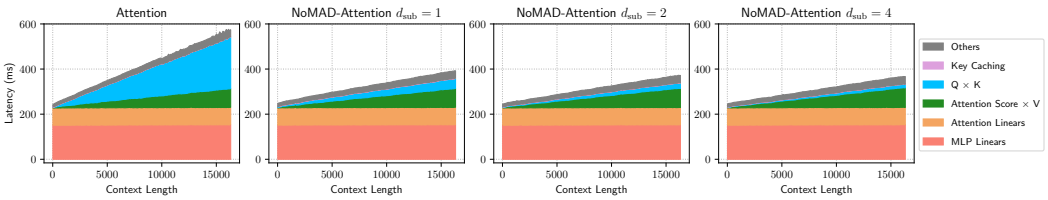

Figure 7: The breakdown of decoding latency of 16-bit CodeLlama-7b for Attention and NoMAD-Attention. NoMAD-Attention effectively reduces the latency for computing dot-products of queries and keys.

D Latency Breakdown

Figure 7 presents the latency breakdown of CodeLlama-7b with 16-bit weights.

E Comparison with Integer Quantization

We empircally compare NoMAD-Attention and Attention with integer quantized key cache. We test the accuracy of LLaMA-7b, using NoMAD-Attention ($d_{\text{sub}} = 1$) and 8-bit and 4-bit integer quantized key cache (q8_0 and q4_0 in llama.cpp). Furthermore, we test the decoding latency at a context length of 16K tokens, and measure the speedup compared to the full model. As shown in Table 5, NoMAD-Attention ($d_{sub} = 1$) demonstrates better model quality than INT4-quantized key cache and significantly higher speedup than INT4- and INT8-quantized key cache.

Table 5: Zero-shot accuracy and decoding latency (at a context length of 16K) comparison between NoMAD-Attention and Attention with integer quantized key cache, for LLaMA-7b.

	SciQ	Arc-E	Arc-C	Hellaswag	WinoGrande	PIQA	Avg	Latency Per Token (ms)	Speedup
Original Attention	94.6	75.21	41.89	56.93	70.09	78.67	69.565	572.497	-
INT8 Key Cache	94.7	75.29	42.15	57.00	70.09	78.63	69.643	562.377	1.018×
INT4 Key Cache	93.6	74.33	41.04	55.34	67.96	77.69	68.327	540.319	1.060×
NoMAD-Attention ($d_{\text{sub}} = 1$)	94.9	75.34	41.81	56.57	70.56	78.56	69.623	391.098	**1.464×**

F Results on Llama-3

We evaluate the effectiveness of NoMAD-Attention on the Llama-3-8b [17] model, which uses grouped query attention [2]. Table 6 presents the accuracy of Llama-3-8b with Attention and NoMAD-Attention across a range of downstream tasks. The results demonstrate that NoMAD-Attention with $d_{sub} = 1$ effectively maintains model quality for Llama 3.

Table 6: Accuracy of the Llama-3-8b model with NoMAD-Attention on downstream tasks.

	SciQ	Arc-E	Arc-C	Hellaswag	WinoGrande	PIQA
Attention	96.4	80.09	50.51	60.18	72.77	79.71
NoMAD-Attention ($d_{\text{sub}} = 1$)	96.1	80.05	49.49	59.86	73.16	79.65
NoMAD-Attention ($d_{\text{sub}} = 2$)	94.8	78.32	46.59	57.52	70.17	78.35
NoMAD-Attention ($d_{\text{sub}} = 4$)	86.3	70.08	37.54	47.38	57.38	76.61

G Additional Evaluations

We evaluate NoMAD-Attention on the more challenging MMLU [22], GPQA [36], and MGSM (English) [40] benchmarks. We use lm-evaluation-harness [19] for accuracy evaluation, and the task names are `mmlu_stem`, `mmlu_social_sciences`, `mmlu_humanities`, `mmlu_other`, `gpqa_main_zeroshot`, `mgsm_direct_en`. As shown in Table 7, NoMAD-Attention with $d_{sub} = 1$ effectively maintains model quality across these diverse challenging tasks.

https://doi.org/10.52202/079017-3581

Table 7: Accuracy of LLaMA models on the challenging MMLU, GPQA, and MGSM (English) benchmarks.

Model	Method	MMLU				GPQA	MGSM
		STEM	Social Sciences	Humanities	Other		
LLaMA-7b	Attention	26.39	29.57	29.73	33.15	20.98	4.8
	NoMAD-Attention ($d_{sub} = 1$)	27.31	29.12	29.44	32.8	23.66	4.0
	NoMAD-Attention ($d_{sub} = 2$)	25.25	24.99	26.82	30.09	20.08	2.0
	NoMAD-Attention ($d_{sub} = 4$)	25.21	22.98	24.85	26.94	25.67	1.6
LLaMA-13b	Attention	34.13	44.39	40.60	46.48	28.35	7.2
	NoMAD-Attention ($d_{sub} = 1$)	33.43	43.71	39.57	45.83	28.35	7.6
	NoMAD-Attention ($d_{sub} = 2$)	30.70	37.96	34.39	40.20	27.01	6.8
	NoMAD-Attention ($d_{sub} = 4$)	25.88	25.58	27.46	27.55	25.67	1.2

H Clarification on d_{sub}

NoMAD-Attention achieves speedup relative to full Attention when $d_{\text{sub}} = 1$ primarily due to two factors: 1. **Leveraging Lower-Latency, Higher-Throughput Instructions.** Unlike the vanilla multiply-add attention, which relies on batched multiplication and addition SIMD instructions (e.g., `_mm256_mul_ps` and `_mm256_add_ps` in AVX2), NoMAD-Attention utilizes the SIMD lookup instruction (`_mm256_shuffle_epi8`). This latter instruction operates on more elements at once (32 elements versus 8) and exhibits lower latency (1 cycle vs. 4 cycles on most architectures) [1], contributing significantly to the efficiency gains. 2. **Minimized Data Movement.** The product-quantized key cache employed in NoMAD-Attention effectively reduces the volume of data transferred between RAM and registers, hence speeding up computations.

I Generalizing to More Attention Types

NoMAD-Attention can generalize to other pretrained transformer models and attention variants such as grouped-query attention (GQA) [2] and Attention with Linear Biases (ALiBi) [34]. Given that GQA employs a shared key head across multiple query heads, we can adapt NoMAD-Attention to reuse the same key codes for performing in-register lookups. The ALiBi method adds a linear bias term to the query-key dot products, a process fully compatible with the NoMAD-Attention approach.

[1] `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html`

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction accurately reflect the scope and contribution shown in the LLM Inference on CPUs and Methodology sections and are supported by the Results and Ablation Study in the Experiments sections.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of our work were made clear throughout the paper, and we include a separate Limitations and Broader Impacts section detailing the limitations.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

https://doi.org/10.52202/079017-3581

Answer: [NA]

Justification: The paper does not include any new theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: All information needed to reproduce the main experimental results, including the parameters, algorithmic details, software, models, benchmarks, and baselines, are fully disclosed in the Methodology and Experiments sections.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The code is proprietary to xMAD.ai. However, anyone is able to reproduce our results using the algorithm and procedure we describe in the paper. We will provide a mechanism to reproduce our numbers in the future.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: All training and test details necessary to understand the results are specified in the Experiments section.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Error bars or other appropriate information about the statistical significance are correctly defined wherever applicable.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

 https://doi.org/10.52202/079017-3581

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The Experiments section details the computer resources needed to reproduce the experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: After reviewing the NeurIPS Code of Ethics, we can confirm that the paper conforms with the code in every respect.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We include a Limitations and Broader Impacts section to detail the potential societal impacts of our work.

    Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: With no new dataset or model, the paper poses no such risk.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The assets used in the paper were properly credited and their licenses and terms were listed and respected.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

 https://doi.org/10.52202/079017-3581

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: New assets introduced in the paper are documented wherever applicable.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor reserach with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

https://doi.org/10.52202/079017-3581