MULTIPOLE SEMANTIC ATTENTION: A FAST APPROXIMATION OF SOFTMAX ATTENTION FOR PRETRAINING

Rupert Mitchell[*1] **and Kristian Kersting**[1,2,3,4]
[1]Department of Computer Science, TU Darmstadt, Darmstadt, Germany
[2]Hessian Center for AI (hessian.AI), Darmstadt, Germany
[3]German Research Center for Artificial Intelligence (DFKI), Darmstadt, Germany
[4]Center for Cognitive Science, TU Darmstadt, Darmstadt, Germany

September 16, 2025

ABSTRACT

We present Multipole Semantic Attention (MuSe), an efficient approximation of softmax attention that combines semantic clustering with multipole expansions from computational physics. Our method addresses the quadratic computational complexity of transformers in the context length by clustering queries and keys separately in their learned representation spaces, enabling a hierarchical two-stage attention mechanism. Unlike prior clustering approaches that group only keys or use unified clustering, we maintain separate clusterings that respect attention's asymmetric treatment of these spaces. We augment centroid-based (monopole) approximations with dipole corrections that capture directional variance within clusters, preserving richer information during training. The method operates as a drop-in replacement for standard attention, requiring only hyperparameter specification without architectural modifications. Our approach achieves $\mathcal{O}(NCD)$ complexity for acausal attention with C clusters and $\mathcal{O}(NCD \log N)$ for causal attention. On isolated attention layers, we demonstrate 3× speedup over CUDNN Flash Attention at 8k context length, with relative squared errors below 20%. For causal attention, we develop a hierarchical block decomposition that combines exact local computation with efficient long-range approximation. In end-to-end pretraining of a 30M parameter model on book-length texts with 16k context, we achieve 12.2% runtime reduction with only 0.36% loss degradation, establishing the viability of multipole approximations for efficient transformer pretraining.

1 Introduction

The quadratic computational complexity of softmax attention remains the primary bottleneck limiting context length in transformers. While this $\mathcal{O}(N^2D)$ scaling enables the rich token interactions that underpin transformer capabilities, it renders long-context pretraining prohibitively expensive. Modern pretraining increasingly benefits from extended context—whether processing entire books, collections of related scientific papers, or complete code repositories—yet computational constraints force most models to train on artificially truncated sequences. Modern transformers partially mitigate these issues through the use of flash attention [Dao et al., 2022], which reduces memory complexity to linear but retains quadratic computational complexity. Hybrid architectures using sliding window attention for local interactions must still interleave quadratic-complexity global attention layers to maintain full-context understanding. Efficient approximations to global attention, which maintain training quality at these scales, become essential for next-generation models.

In this context, we present Multipole Semantic Attention (MuSe), combining semantic clustering with multipole expansions from computational physics to approximate softmax attention. Our goal is to establish the viability of this approach by demonstrating that (1) the approximation preserves sufficient quality for training convergence, (2)

---

[*]Institutional email: `rupert.mitchell@tu-darmstadt.de`   Correspondence: `mail@rupertmitchell.com`

practical speedups are achievable with current hardware at context lengths of 16k and beyond, and (3) the method integrates into existing training pipelines without architectural changes. We focus on validating the core algorithm at a scale where rapid iteration is feasible, leaving systematic exploration of scaling behavior and optimal hyperparameter selection to future work. More specifically, MuSe introduces three key elements: First, we cluster queries and keys separately in their learned representation spaces, enabling a two-stage mechanism where coarse query clusters attend to fine key clusters, then fine queries refine through cluster summaries. Second, we augment centroid-based (monopole) approximations with dipole terms that capture directional variance within clusters. Third, the method operates as a drop-in replacement for standard attention, requiring only hyperparameter specification (cluster counts, K-means iterations) without architectural modifications.

This distinction between the key and query latent spaces, whereby we cluster keys and queries independently is motivated as follows. Firstly, softmax attention is invariant under translation of the keys (as this change is absorbed by the normalizing constant), but not of the queries. More importantly, the queries live in the dual vector space to the keys, that is, softmax attention is further invariant to an arbitrary change of basis of the keys, so long as the queries are transformed inversely. The consequence of this is that, unless one has some way to choose a preferred basis, it is unclear what it would mean to say that some particular key and query occupied the same point in latent space.

We validate MuSe empirically through several microbenchmarks and end-to-end pretraining on book-length texts from Project Gutenberg. On isolated attention layers, we achieve 3× speedup over CUDNN Flash Attention at 8k context, reaching 20-30× at 64k tokens, with approximation errors below 20%. For causal attention, we develop a hierarchical block decomposition combining exact local computation with approximate long-range dependencies. In pretraining a 30M parameter model with 16k context—a scale enabling thorough experimentation—we demonstrate 12.2% runtime reduction with 0.36% loss degradation, suggesting that multipole approximations merit further investigation for efficient long-context pretraining.

To summarize, this paper makes the following contributions:

- A bidirectional semantic clustering approach that partitions queries and keys *separately* in their native representation space, enabling a hierarchical two-stage attention mechanism: coarse query clusters first attend to all fine key clusters to produce query-dependent summaries, then fine queries refine these summaries using their residual components—fundamentally different from methods that cluster only keys or use unified clustering

- Integration of dipole corrections that preserve directional variance within semantic clusters, extending beyond centroid-based approximations to maintain richer output information during training

- Empirical demonstration that the method can achieve meaningful speedups while maintaining acceptable approximation quality for pretraining

To this end, we proceed as follows. We start off by reviewing related efficient attention methods. Sections 3-4 develop the monopole and multipole formulations. Section 5 then presents the MuSe approach, which we extend to causal attention in Section 6. Before concluding, Section 7 presents the empirical evaluation.

2 Related Work

We organize existing efficient attention methods into two broad categories, those which restrict attention to only occur within certain subsets of tokens but do so exactly, and those which aim to approximate attention globally. We fall into the latter category, and together with Hooper et al. [2025a] are the only works using clustering therein.

2.1 Restricted Attention

Restricted attention methods reduce the complexity of attention by only computing attention between subsets of queries and keys.

Bucketed Attention   Reformers [Kitaev et al., 2020] bucket queries and keys according to random directions in key-query space and restrict attention to occur within buckets, on the principle that large magnitude queries and keys effectively perform a nearest neighbour lookup operation. If multiple bucketings are used, then this lookup operation can be simulated cheaply with high reliability, though in order to do so they force queries and keys to be the same for the same token, and lose the global aggregation behaviour of attention for smaller magnitude keys and queries.

Routing Transformers [Roy et al., 2021] learn a k-means clustering of the queries and keys during training rather than using random directions, and similarly restrict attention to only occur within buckets.

K-Means at Inference Time   Since late 2024 there has been significant work on the use of K-means clustering in key-space for the acceleration of long context inference in language models. Hooper et al. [2025b] use context-specific kmeans clustering on keys to perform exact attention on those keys most likely to be relevant to a given query. He et al. [2025] similarly use kmeans clustering in A2ATS to formulate inference attention as a lookup problem, and further use knowledge of the expected query distribution and position encodings to improve this representation, though they rely on a static offline clustering of keys. Tactic [Zhu et al., 2025] also uses a context-specific clustering of keys, but focus on adapting the number of tokens on which exact attention is performed to the temperature of the relevant attention patterns.

2.2   Approximate Attention

These methods allow all tokens to contribute to the output of attention, and instead look for ways to approximate the output of full softmax attention without the quadratic complexity.

Position Based   Wang et al. [2020] use random projections in the spatial domain to approximate the attention matrix in Linformer, exploiting the low rank structure of this matrix. Nyströmformer [Xiong et al., 2021] also constructs a low rank representation of the attention matrix, but this time in terms of "landmark" keys and queries, which they obtain by taking segment-wise means of spatially proximate keys and queries.

Fast Multipole Attention [Kang et al., 2023] and H-Transformer [Zhu and Soricut, 2021] instead rely on a recursive decomposition of the sequence dimension in the spirit of the Barnes-Hut approximation for N-body simulation [Barnes, 1986]. While H-Transformer uses a fixed spatial averaging of these subsets, Kang et al. [2023] allow the model to learn multiple convolutional summaries of each segment, somewhat like the Fast Multipole Method [Rokhlin, 1985] retaining multiple terms of the multipole expansion.

Learned Bottlenecks   Jaegle et al. [2021] observed that while self-attention for the full context is quadratic in context length, attention from a fixed latent sequence to the full context is linear. In applications such as image classification, the output domain has no need to retain the spatial structure of the input domain, so the self-attention complexity can be massively reduced by having all self attention occur between latent representations at artificial, latent positions, only using the original tokens for the source of cross-attention.

Ma et al. [2021] contemporaneously construct Luna for cases where the output spatial structure must be maintained, by adding a further step where the original token positions attend to the latent ones. They introduce a new architecture designed to allow the model to learn suitable latent representations at these latent tokens, and allow long-context language modeling by forcing all information to flow through this latent bottleneck.

Kernel Approximations   Performer [Choromanski et al., 2021] uses random features to approximate the exponential kernel, effectively routing attention through stochastic latent vectors in key-query space and thereby deriving an unbiased estimator of softmax attention. Their method has impressive provably small mean squared error in cases where both the queries and keys have low magnitude.

Semantic Clustering   Our work, MuSe, falls in the approximate attention category, specifically using semantic clustering of key-query representation space. While Hooper et al. [2025a] recently introduced the use of monopole (centroid) summaries of "far field" clusters at inference time as an augmentation of lookup-based approaches, we present the first training-time method with three key innovations:

1. **Separate query and key clustering.** While Routing Transformers uses unified clustering and all inference methods cluster only keys, we partition queries and keys independently in their respective representation spaces enabling a hierarchical two-stage attention mechanism where coarse query clusters attend to fine key clusters to produce query-dependent summaries, then fine queries refine these summaries using residual components.

2. **True Multipole approximations beyond monopoles.** Unlike Hooper et al. [2025a] and all other clustering methods that use only centroid (monopole) representations, we incorporate covariance-based dipole terms that capture directional variance within clusters, reducing approximation error while maintaining linear complexity.

3. **Training-time application with global aggregation.** In contrast to inference-only methods that restrict attention to top-k keys and training methods like Reformer that limit attention to within-bucket computation, our method approximates contributions from all keys during training, preserving gradient flow and global information aggregation critical for model learning.

3

These innovations work synergistically: separate clustering enables query-specific refinement in our two-stage mechanism, while dipole corrections recover information lost through clustering, together achieving practical speedups with acceptable training-time approximation quality.

3   Semantic Monopole Attention

Softmax attention is quadratic in the context length because every query must attend to every key.[2] The core idea of MuSe is to cluster the keys and queries such that substituting certain summary statistics of each cluster into the softmax attention algorithm may be taken as an approximation of using its full contents directly.

3.1   Clustering both queries and keys

Supposing that we have N target locations, or queries, and N source locations, or keys, naive attention has a cost of $\mathcal{O}(NND)$ where D is the head dimension, e.g., 64. The first thing one notices is that, if we partition our keys into C_k clusters, we can attend from our fine original queries to our coarse summarized clusters with $\mathcal{O}(NC_kD)$ cost, saving a factor of N/C_k. Interestingly, one could also attend from coarse summarized *queries* to fine original keys at $\mathcal{O}(C_qND)$ cost, saving a factor of N/C_q, supposing that we partitioned our queries into C_q clusters. (In practice we will use $C_k = C_q$ and refer instead to simply C where the distinction is unimportant.) The first core idea of our approach can be seen as finding a way to combine these two possibilities.

Suppose we decompose each original query q into the centroid of the coarse cluster to which it is assigned $\overline{q}$ plus some diff $\tilde{q}$. We first attend from each of the C_q coarse queries to the full contents of each of the C_k clusters of fine keys, constructing a query-dependent summary of the results, producing C_qC_k summaries with $\mathcal{O}(C_qND)$ cost. For each fine query, we then take the C_k summaries obtained by the coarse query of the cluster to which it belongs and attend to them with the residual query $\tilde{q}$ for this target location, with $\mathcal{O}(NC_kD)$ cost.[3]

In practice, we construct these clusterings by running K-means for between 1 and 5 iterations at $\mathcal{O}(NCD)$ cost.

3.1.1   Why two stages, intuitively?

Suppose we write a particular query $q = \overline{q} + \tilde{q}$ in terms of its assigned centroid $\overline{q}$ plus a residual $\tilde{q}$. Let us similarly decompose a key $k = \overline{k} + \tilde{k}$ where $\tilde{k}$ is the residual as before, though we will later find that the optimal choice of $\overline{k}$ is not always precisely the original centroid assigned to k.

The unnormalized attention weight from q to k is given by $\exp(q \cdot k)$. Attending only from coarse queries to fine keys would yield the approximation $\exp(q \cdot k) = \exp(\overline{q} \cdot k)\exp(\tilde{q} \cdot k) \approx \exp(\overline{q} \cdot k)$. Similarly, attending only from fine queries to coarse keys would correspond to the approximation $\exp(q \cdot k) \approx \exp(q \cdot \overline{k})$. Our two stage approximation combines these to instead effectively use

$$\exp(q \cdot k) = \exp(\overline{q} \cdot \overline{k} + \overline{q} \cdot \tilde{k} + \tilde{q} \cdot \overline{k} + \tilde{q} \cdot \tilde{k}) \approx \exp(\overline{q} \cdot \overline{k} + \overline{q} \cdot \tilde{k} + \tilde{q} \cdot \overline{k}) \tag{1}$$

where the term we neglect includes the dot product of two residuals instead of just one.

3.1.2   Stacking Attention

Take the following definitions of $M(q)$ and $V(q)$:

$$M(q) := \sum_k \exp(q.k), \quad V(q) := \frac{\sum_{k,v} \exp(q.k)v}{M(q)} \tag{2}$$

It can be seen that $V(q)$ is equivalent to softmax attention, and $M(q)$ is the normalizing term commonly returned in the form $\log M(q)$ for use in the backward pass.

If we partition our k, v pairs into C_k clusters then the attention computations compose as follows:

$$M_T(q) := \sum_{j \in C_k} M_j(q), \quad V_T(q) := \frac{\sum_{j \in C_k} M_j(q)V_j(q)}{\sum_{j \in C_k} M_j(q)} \tag{3}$$

---

[2]Making the attention causal saves up to a factor of two, but does not change this fundamental quadratic scaling.
[3]Note that while C_qC_k total summaries are produced, any particular output location will only attend to the C_k summaries corresponding to its assigned query cluster in this second step.

Algorithm: Softmax Attention (attn)

Input: $Q \in \mathbb{R}^{N \times D}, K \in \mathbb{R}^{N \times D},$
$\qquad V \in \mathbb{R}^{N \times D}, b \in \mathbb{R}^{N}$
Output: $Y \in \mathbb{R}^{C_q \times U \times D}, \mu \in \mathbb{R}^{C_q \times U}$

$S = \text{matmul}(Q, K^T) + \mathbf{1}b^T;$
$P = \text{softmax}(S, \text{axis} = -1);$
$\mu \leftarrow \text{logsumexp}(S, \text{axis} = -1);$
$Y \leftarrow \text{matmul}(P, V);$

Algorithm 1: Standard softmax attention with bias and returning logsumexp values for backward pass.

Algorithm: Semantic Monopole Attention

Input: $\overline{Q} \in \mathbb{R}^{C_q \times D}, \tilde{Q} \in \mathbb{R}^{C_q \times U \times D},$
$\qquad K \in \mathbb{R}^{C_k \times U \times D}, V \in \mathbb{R}^{C_k \times U \times D}$
Output: $Y \in \mathbb{R}^{C_q \times U \times D}, \mu \in \mathbb{R}^{C_q \times U}$

$\overline{V}, \overline{\mu} \leftarrow \text{attn}(\overline{Q}, K, V, \mathbf{0});$
$\overline{K}, \_ \leftarrow \text{attn}(\overline{Q}, K, K, \mathbf{0});$
$Y, \mu \leftarrow \text{attn}(\tilde{Q}, \overline{K}, \overline{V}, \overline{\mu});$

Algorithm 2: Semantic Monopole Attention written explicitly in terms of softmax attention (attn : $Q, K, V, b \rightarrow Y, \mu$ as seen in Algorithm 1).

where M_T and V_T are the attention results for the whole set of keys and values, and M_j and V_j are the attention results on each cluster j of keys and values separately.

In this way, clustering based approximations to softmax attention may be constructed by approximating $\hat{M}_j \approx M_j$ and $\hat{V}_j \approx V_j$ for each cluster in a way which does not require looking at their entire contents. A particularly simple class of approaches, which we shall refer to as "Monopole" approaches, looks like:

$$\hat{M}_j(q) := \exp(q \cdot \overline{k}_j), \quad \hat{V}_j(q) := \overline{v}_j \tag{4}$$

for some representative key and value vectors $\overline{k}_j$ and $\overline{v}_j$ for cluster j, e.g. the key and value centroids.

In the two level approach one breaks queries $q = \overline{q} + \tilde{q}$ into contributions from the cluster centroid $\overline{q}$ and a residual $\tilde{q}$. One then constructs approximations $\hat{M}_{ij}(\tilde{q})$ and $\hat{V}_{ij}(\tilde{q})$ for each query cluster i and key-value cluster j. We will additionally need the definition $K_j(q) := \frac{\sum_k \exp(q.k)k}{\sum_k \exp(q.k)}$ corresponding to replacing the values with the keys in $V_j(q)$. We then choose the approximations

$$\hat{M}_{ij}(\tilde{q}) := M_j(\overline{q}_i)\exp(\tilde{q} \cdot K_j(\overline{q}_i)), \quad \hat{V}_{ij}(\tilde{q}) := V_j(\overline{q}_i) \tag{5}$$

With C_q query clusters, the M_{ij}, K_{ij} and V_{ij} may be constructed in $C_q N$ time, and applying the approximation for every $\tilde{q}$ can be done in $C_k N$ time.

3.1.3 Semantic Monopole Algorithm

Suppose we define an extended softmax attention function attn : $Q, K, V, b \rightarrow Y, \mu$ as seen in algorithm 1. Here we make explicit that the attention function returns the logsumexp for each query location, as most efficient implementations do since it is useful in the backward pass. We further permit a bias b to be added to the attention scores for each key-value location.

We can then express the Monopole component of MuSeas seen in algorithm 2. Here we have written the algorithm for C equal clusters of size $N/C = U$, though in practice one may want to allow flexibility in the cluster size and work with the resulting ragged tensors.

4   Multipole Semantic Attention (MuSe)

Having desribed the monopole version of our algorithm, i.e., the version where the key-value clusters are represented by simple centroids, we now turn to more advanced approximations which capture additional structure in the key-value clusters. In particular we will end up introducing terms dependent upon the covariance between keys and values within a cluster. Since we are approximating a vector valued function, these covariance matrices constitute the linear, or dipole, component of the approximation.

Polynomial Expansion via Cumulant Generating Functions   The keys k of some cluster j can be considered as a probability distribution of equally weighted point masses at each k. They then have the moment generating function $\mathcal{M}_j(q) := \mathbb{E}_{k \in C_j} \exp(q \cdot k)$ whose nth derivative at the origin gives the nth moment as an n-index tensor. The cumulant generating function (CGF) $\mathcal{K}_j(q) := \ln \mathcal{M}_j(q)$ is given by the natural logarithm of the moment generating function, and has as its nth derivative the nth cumulant. For those unfamiliar with cumulants, these are familiar quantities such as the mean, variance, skewness, excess kurtosis, etc. which describe the shape of the distribution.
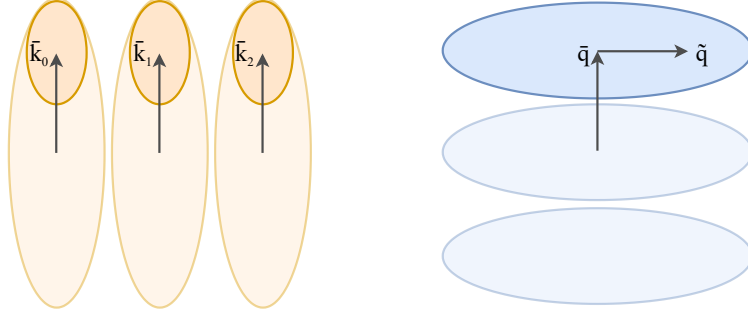
Figure 1: Idealized depiction of MuSe attention. For some particular query centroid $\overline{q}$, we exponentially tilt the key clusters, producing new centroids $\overline{k}_j$ and $\overline{v}_j$ for keys and values for each key-value cluster j. To the extent that these exponentially tilted clusters are narrow in the direction of a subsequent residual query $\tilde{q}$, the approximation will be accurate. Note that we therefore require only the product of the intra-cluster variances $\mathrm{Cov}(\tilde{q}, \tilde{q})$ and $\mathrm{Cov}(\tilde{k}, \tilde{k})$ to be small in any given direction, not both separately. (Formally, we require $\mathrm{Tr}[\mathrm{Cov}(\tilde{q}, \tilde{q})\,\mathrm{Cov}(\tilde{k}, \tilde{k})]$ to be small). Note further that we may make no assumption that any key-value clusters may be neglected or approximated with lower fidelity for any particular query centroid - the exponential tilting itself corresponds to the "select nearby regions to more accurately approximate" part of Fast Multipole-like methods.

Interestingly for our purposes, attenting to these keys with $\overline{q}_i$ corresponds to an exponential tilt of this distribution (i.e., a weighting by $\exp(\overline{q}_i \cdot k)$ for each point). The CGF $\mathcal{K}_{ij}(\tilde{q})$ as a function of $\tilde{q}$ of the resulting distribution after reweighting cluster j with query centroid $\overline{q}_i$ corresponds to simply evaluating $\mathcal{K}_j$ at $\overline{q}_i$ and shifting it so that it is zero at the origin (i.e., $\mathcal{K}_{ij}(\tilde{q}) = \mathcal{K}_j(\overline{q}_i + \tilde{q}) - \mathcal{K}_j(\overline{q}_i)$).

In order for this to be actually useful, we need to incorporate the values somehow, so let us define the CGF of the joint distribution of keys and values $\mathcal{K}_j(q, t) := \ln \mathbb{E}_{k,v \in C_j} \exp(q \cdot k + t \cdot v)$ for a key-value cluster j. Recalling that the first derivative of the CGF is the mean, we obtain first that the value centroid of cluster j is given by $\left. \frac{\partial \mathcal{K}_j(q,t)}{\partial t} \right|_{q=0, t=0}$. More interestingly, the output of standard softmax attention on this cluster $V_j(\overline{q}) = \left. \frac{\partial \mathcal{K}_j(q,t)}{\partial t} \right|_{q=\overline{q}, t=0}$ for the query centroid $\overline{q}$ is given by the derivative of $\mathcal{K}_j$ with respect to t at $q = \overline{q}$ and $t = 0$.

This means that if we decompose some particular query $q = \overline{q}_i + \tilde{q}$ in query cluster i into its centroid $\overline{q}_i$ and a residual $\tilde{q}$, we can approximate $V_j(q)$ by taking a polynomial expansion of V_j around $\overline{q}$. Recalling that the derivatives of the CGF are the mean, variance, skewness etc, we have

$$V_{ij}(\tilde{q}) = \left. \frac{\partial \mathcal{K}_j(q,t)}{\partial t} \right|_{q=\tilde{q}+\overline{q}, t=0} = \overline{v}_{ij} + \mathrm{Cov}_{ij}(v,k)\tilde{q} + \frac{1}{2}\tilde{q}^T \mathrm{Skew}_{ij}(v,k,k)\tilde{q} + ... \tag{6}$$

where $\overline{v}_{ij}$, $\mathrm{Cov}_{ij}(v,k)$ and $\mathrm{Skew}_{ij}(v,k,k)$[4] are statistics of the joint key-value distribution of key-value cluster j after exponentially tilting by the centroid $\overline{q}_i$ of query cluster i. Note that, since we are expanding in q, each successive term picks up an extra k index and contracts with $\tilde{q}$ an additional time.

In section 3.1.2 we required an approximation for the value $V_{ij}(\tilde{q})$ obtained by attending to key-value cluster j with a residual query in query cluster i, which we have now derived a polynomial expansion of. We further required an estimate of the unnormalized attention weight $M_{ij}(\tilde{q})$, which can be obtained similarly to above as follows

$$M_{ij}(\tilde{q}) = M_j(\overline{q}_i)\exp(\mathcal{K}_j(\tilde{q} + \overline{q}) - \mathcal{K}(\overline{q})) = M_j(\overline{q}_i)\exp\left(\tilde{q} \cdot \overline{k}_{ij} + \frac{1}{2}\tilde{q}^T \mathrm{Cov}_{ij}(k,k)\tilde{q} + ...\right) \tag{7}$$

where the values no longer feature, and the required statistics for the expansion are only of the key distribution after reweighting with $\overline{q}_i$.

Dipole Truncation   We choose to retain terms that are at most linear in $\tilde{q}$. The leading neglected terms are then quadratic in $\tilde{q}$ and, on inspection of the coefficients, also quadratic in $\tilde{k}$. For small $\tilde{q}$ we therefore have an error $\mathcal{O}(\tilde{q}^2\tilde{k}^2)$.

Most clearly, this constitutes dropping the quadratic terms in the expansions of $V_{ij}(\tilde{q})$ and $M_{ij}(\tilde{q})$. Since we are implicitly going to multiply these values, we can in fact also drop the linear term of $V_{ij}(\tilde{q})$ when multiplying by the

---

[4]Note that $\mathrm{Skew}_{ij}(v,k,k)$ is a three index tensor, where we contract $\tilde{q}$ with each k index, though we treat it as a matrix with the v index omitted in equation 6 for notational convenience.

6

Algorithm: MuSe: Initial Stage

Input: $\overline{Q} \in \mathbb{R}^{C_q \times D}, \overline{K} \in \mathbb{R}^{C_k \times U \times D},$
$\qquad V \in \mathbb{R}^{C_k \times U \times D}$

Output: $\overline{K} \in \mathbb{R}^{C_q \times C_k \times D}, \overline{V} \in \mathbb{R}^{C_q \times C_k \times D},$
$\qquad \overline{\mu} \in \mathbb{R}^{C_q \times C_k}, \overline{C} \in \mathbb{R}^{C_k \times D \times D}$

for $k \in [0..K)$ **do in parallel**
$\quad | \quad S \leftarrow \mathrm{matmul}(Q, K[k]^T);$
$\quad | \quad \overline{\mu}[:, k] \leftarrow \mathrm{logsumexp}(S, \mathrm{axis} = -1);$
$\quad | \quad P \leftarrow \mathrm{softmax}(S, \mathrm{axis} = -1);$
$\quad | \quad \overline{K}[:, k, :] \leftarrow \mathrm{matmul}(P, K[k]);$
$\quad | \quad \overline{V}[:, k, :] \leftarrow \mathrm{matmul}(P, V[k]);$
$\quad | \quad \overline{C}[k] \leftarrow \mathrm{Cov}(V[k], K[k]);$
end

Algorithm 3: Note that the loop over k is in practice parallelised. The covariance is computed over the U dimension, i.e., across tokens per cluster, and unweighted by P, and corresponds to $\mathrm{Cov}_j(v, k)$ - it is partially specialized to $\mathrm{Cov}_{ij}(v, k)$ later. $\overline{\mu}$ is $\log M_j(\overline{q}_i)$ from equation 7.

Algorithm: Semantic Dipole Attention: Final Stage

Input: $\tilde{Q} \in \mathbb{R}^{C_q \times U \times D}, \overline{K} \in \mathbb{R}^{C_q \times C_k \times D},$
$\qquad \overline{V} \in \mathbb{R}^{C_q \times C_k \times D}, \overline{\mu} \in \mathbb{R}^{C_q \times C_k},$
$\qquad \overline{C} \in \mathbb{R}^{C_q \times D \times D}$

Output: $Y : [Q, N, V]$

for $q \in [0..Q)$ **do in parallel**
$\quad | \quad S \leftarrow \mathrm{matmul}(\underline{Q}[q], \overline{K}[q]^T);$
$\quad | \quad S_{ij} \leftarrow S_{ij} + \overline{\mu}_j;$
$\quad | \quad P \leftarrow \mathrm{softmax}(S, \mathrm{axis} = -1);$
$\quad | \quad Y[q] \leftarrow \mathrm{matmul}(P, \overline{V}[q]) + \mathrm{matmul}(\underline{Q}[q], \overline{C}[q])$
end

Algorithm 4: Note that the loop over q is in practice parallelised. Adding $\overline{\mu}$ implements the $M_j(\overline{q}_i)$ scaling of equation 7 in log space. $\mathrm{matmul}(\underline{Q}[q], \overline{C}[q])$ is $\mathrm{Cov}_i{}_j(v, k)\tilde{q}$ from equation 6.

linear term of $M_{ij}(\tilde{q})$. This corresponds to the very convenient fact that we only need to compute a single dipole correction for every residual query rather than one for every cluster, since we can pre-merge the covariances matrices $\mathrm{Cov}_{ij}(v, k)$ for each query cluster i. This reduces the computational complexity of evaluating the dipole corrections from $\mathcal{O}(NC_kD^2)$ to $\mathcal{O}(ND^2)$.

The expected error over all queries can be expressed more precisely as $\mathcal{O}(\mathrm{Tr}(\mathrm{Cov}(\tilde{q}, \tilde{q})\,\mathrm{Cov}(\tilde{k}, \tilde{k}))$ where these are intra-cluster variances weighted by cluster size, i.e., exactly the quantity minimized by K-means, motivating our choice of clustering method. The intuition for this coupled dependency on key and query covariances is given in figure 1. Strictly speaking the queries are ideally clustered in a metric given by the intra-cluster variance of the keys and vice-versa, but we use the euclidean metric in practice for simplicity. Note that, as we increase the number of clusters C of queries and keys, these covariances approach zero, so we can drive error arbitrarily small as long as we can afford to use a large enough number of clusters.

Partially Specialized Dipoles   While computing the dipole corrections given the covariance matrices $\mathrm{Cov}_{ij}(v, k)$ is $\mathcal{O}(ND^2)$, actually computing the matrices themselves is still $\mathcal{O}(C_qND^2)$. In practice, to reduce the computational expense of calculating $\mathrm{Cov}_{ij}(v, k)$, we only calculate this term per cluster j but do not include the exponential tilt by $\overline{q}_i$ (though we use this tilt for the $\overline{v}_{ij}$ terms. We instead calculate a single $\mathrm{Cov}_j(v, k)$ term for each key-value cluster. This reduces the accuracy of the approximation from $\mathcal{O}(\tilde{q}^2\tilde{k}^2)$ to $\mathcal{O}(\overline{q}q\tilde{k}^2)$, but allows us to balance memory and compute expenses across the monopole and dipole terms. For the cluster counts we use in practice (32-128), $\overline{q}$ is not typically much larger than $\tilde{q}$, so this degradation is acceptable to us. In particular, this reduces the cost of computing the covariances from $\mathcal{O}(C_qND^2)$ to $\mathcal{O}(ND^2)$, preventing it from dominating the cost ($\mathcal{O}(C_qND)$) of computing the monopoles.

5   MuSe Algorithm

We now explicitly describe the resulting algorithms for the two stages of MuSe. It will be seen that the actual computations to be performed are in fact deceptively simple.

Clustering   We first cluster the queries and keys separately with a fixed number of iterations of K-means (e.g., 1 or 5) obtaining C_q and C_k clusters respectively, though in practice we choose $C_q = C_k = C$ for simplicity.[5] We initialize by sampling points proportional to their squared norm, as this is fast and equivalent to the first sample of K-means++ for zero mean vectors. It is convenient to enforce a maximum cluster size of, e.g., 1.5 times the average cluster size, to simplify handling of ragged tensors later on.

---

[5]We found $C_q = C_k = D = 64$ to be a sweet spot from an implementation perspective, since this results in square matrix-matrix multiplies and makes C and D interchangeable in complexities.

Initial Stage   The output of the first stage of MuSe consists of monopole and dipole terms. The monopole terms consist of $C_q \times C_k$ key and value centroids, i.e., each of the Q coarse query clusters gets its own specialized monopole approximation to the C_k coarse key-value clusters. The dipole term consists of C_k intra-cluster covariances between keys and values for each coarse key-value cluster, with no specialization per coarse query cluster. See algorithm 3.

Dipole Aggregation   Stage 1 gives us $C_q \times C_k$ monopole terms, with specialization for each of the C_q query clusters, together with C_k dipole terms but without specialization with respect to query clusters. We now produce a single, partially specialized, dipole term for each query cluster by aggregating the C_k dipole terms according to the $C_q \times C_k$ logsumexp values returned by stage 1, producing an array of dipole terms of shape $[C_q, D, D]$. This corresponds to a simple matrix multiplication with softmax($\overline{\mu}$, axis $= -1$), and is shown in algorithm 5. Note that this operation is very cheap because the attention scores are precomputed and the resulting attention matrix has small size ($C_q \times C_k$) and is independent of the context length.

Final Stage   In the final stage of the algorithm we refine the outputs of each cluster of queries by attending from the residual queries to the C_k key and value centroids with bias given by $\overline{\mu}$. We further apply a dipole correction by contracting each residual query with the value-key covariance matrix specialized to the relevant query cluster.

Algorithm: Dipole Aggregation

Input: $\overline{Q} \in \mathbb{R}^{C_q \times D}, \overline{K} \in \mathbb{R}^{C_k \times D}$,
$\qquad \overline{C} \in \mathbb{R}^{C_k \times D \times D}, \overline{\mu} \in \mathbb{R}^{C_k}$
Output: $\overline{C'} \in \mathbb{R}^{C_q \times D \times D}$

$S \leftarrow \mathrm{matmul}(\overline{Q}, \overline{K}^T)$;
$S_{ij} \leftarrow S_{ij} + \overline{\mu}_j$;
$P \leftarrow \mathrm{softmax}(S, \mathrm{axis} = -1)$;
$\overline{C'} \leftarrow \mathrm{matmul}(P, \overline{C})$;
Algorithm 5: Note that the final matmul contracts P with only the first index of $\overline{C}$.
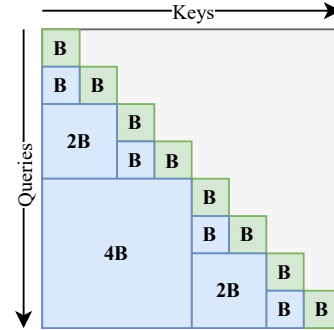

Figure 2: Hierarchical block decomposition for causal attention. Diagonal blocks (green) use Flash attention for exact computation, while below-diagonal blocks (blue) use MuSeapproximation.

6   From Acausal to Causal

We extend our acausal approximation to causal attention by decomposing the lower-triangular attention matrix into blocks as shown in figure 2. The block-diagonal uses Flash attention for exact local computation, while below-diagonal blocks use our MuSe approximation for efficient long-range dependencies.

The block structure forms a binary tree: at each level, blocks double in size, creating $\mathcal{O}(\log N)$ levels total. All blocks at each level are processed in parallel through vectorized operations. For a given query position, outputs from multiple blocks (one from Flash attention on the diagonal, up to $\log N$ from MuSeapproximations) are merged by weighting with their respective logsumexp values:

$$\mathrm{out} = \frac{\sum_i \exp(\mu_i) Y_i}{\sum_i \exp(\mu_i)} \tag{8}$$

where μ_i is the logsumexp value and Y_i the output of the ith block to be merged for this output position. Each level of MuSeprocesses $N/2$ tokens in $\mathcal{O}(NCD)$ time while the flash attention portion runs in $\mathcal{O}(NBD)$ time where B is the block size. This achieves $\mathcal{O}(NCD \log N)$ complexity for causal attention while preserving exact computation for local dependencies where attention is typically strongest.

7   Empirical Evaluation

In this section we empirically evaluate MuSe, focusing on establishing the viability of multipole approximations for transformer pretraining. We address four key questions:

Table 1: Runtime for a combined forward and backward pass, and error of MuSe as clustering quality and number of clusters varies. Here we consider a batch size of 16, a context length of 8192, a head count of 8 and a head size of 64. For this problem size the runtime of our flash attention implementation in Pallas is 67.91ms, and that of Nvidia's cudnn flash attention is 51.74ms. The data used consists of queries, keys and values recorded from the long context attention layer at the end of pretraining for our small gpt model.

Cap	Iters	C	Runtime (ms)	Rel. Sq. Error
1.5	1	32	12.37	0.2475
2.0	3	32	19.37	0.2148
4.0	5	32	29.35	0.2060
1.5	1	64	16.49	0.1946
2.0	3	64	23.05	0.1642
4.0	5	64	34.79	0.1571
1.5	1	128	36.12	0.1435
2.0	3	128	45.82	0.1179
4.0	5	128	72.88	0.1123

Table 2: Runtime in milliseconds for a combined forward and backward pass of MuSe, CUDNN Flash Attention, and our Pallas implementation of Flash Attention as context length N varies holding total batch size (inclusive of head count) fixed at 1M tokens. We emphasize that, since total batch size is constant, linear time algorithms will be constant in this table, and quadratic time algorithms will be linear.

N	MuSe	CUDNN	Pallas
1024	30.03	**8.313**	10.43
2048	24.46	**14.15**	18.65
4096	**18.73**	26.63	35.09
8192	**16.53**	51.77	67.96
16384	**16.29**	102.72	134.62
32768	**16.41**	204.40	267.22
65536	**19.28**	405.72	534.28

1. How the runtime of MuSe compares with that of flash attention.

2. How the quality of clustering, e.g., number of clusters, affects runtime and approximation error.

3. To what extent the various components of our algorithm, such as having two stages and including dipole corrections, contribute to error reduction.

4. Whether the resulting approximation errors in both forward and backward passes are sufficiently small for us to obtain significant wall-clock speedups in long-context pretraining with acceptable loss of accuracy.

We conduct our microbenchmarks (sections 7.1 and 7.2) on isolated attention layers using queries, keys, and values recorded from a trained model, allowing us to thoroughly analyze the approximation quality and runtime characteristics with realistic attention patterns. For end-to-end validation (section 7.3), we demonstrate full pretraining of a 30M parameter model, where we can complete many training runs to 2B to facilitate rapid iteration and comparisons, in order to verify that approximation errors remain acceptable throughout training. Note that all runtimes reported include all overheads, e.g., clustering and merging of acausal blocks, not just the attention computation itself.

7.1   Microbenchmarks

We now benchmark MuSe on a single batch of queries, keys and values recorded after training on 2B tokens. In the main body we consider a context length of 8k tokens, though results for 4k tokens may be found in table 6. We consider a batch size of 16 and a head count of 8, for a combined effective batch size of 128. All benchmarks were performed on a single A100 GPU.

Our key metric of output error here is relative squared error, given by the ratio between the squared 2-norm of the difference between the reference output and the approximate output and the squared 2-norm of the reference output itself. This roughly corresponds to the fraction of variance in the output which is error as opposed to signal.

Referring to table 1, it can be seen that, as expected, both runtime and accuracy increase with cluster count. Runtime costs are roughly linear in cluster count, though we emphasise that our implementation is designed for a cluster count of 64, and performance with either 32 or 128 clusters could likely be improved by changing the kernel design.[6] For this range of cluster counts on this data, the error may be approximately halved by quadrupling the number of clusters. The relative squared error ranges between 0.1123 and 0.2475 for the configurations shown.

---

[6]We tuned block sizes for each cluster count separately, but did not consider changing the iteration pattern or fusing or splitting out kernels.

We opt for the fastest cluster count 64 configuration in our later experiments, with an error of 0.1946. We emphasize that we are using such a low iteration count (1) in order to keep clustering costs similar to attention costs - improved clustering implementations would allow us to use more iterations and significantly reduce error.

It can further be seen that clustering quality has a significant effect on error. One obvious contributor to the clustering quality is number of K-means iterations, which we vary between 1 and 5 (we have found error reduction past 5 iterations to be small, likely because K-means converges quickly in these high dimensional problems). As noted in section 4, we enforce a static cap on cluster size of between 1.5 and 4 times the mean cluster size. A lower cap reduces the compute and storage cost of working with the resulting ragged tensors but reduces clustering quality. Here we vary these hyperparameters jointly, though they can be seen varied individually in tables 5 and 6. The difference between our highest (cap ratio of 4, 5 iterations) and lowest (cap ratio of 1.5, 1 iteration) quality configurations corresponds to a roughly 20% improvement in error, though it more than doubles runtime.

Most notably, however, the runtime of our chosen configuration (cap ratio of 1.5, 1 iteration, 64 clusters) is shown in comparison with the CUDNN implementation of flash attention in table 2 at various context lengths. Also shown is the runtime of our implementation of flash attention in Pallas, which we will use later on. The significance of the Pallas implementation is that it is implemented in the same framework as our MuSe kernels, and is an optimized version of the official example implementation of flash attention in Pallas from Google. This makes it a natural comparison point for similar levels of engineering effort and abstraction.

It can be seen that MuSe begins to be faster than CUDNN's flash attention at a context length of 4k, where it is almost 50% faster than CUDNN, and twice as fast as our Pallas flash attention. Since we hold the total token count constant and scale the batch size inversely with the context length, it can be seen that MuSe has roughly constant runtime as context length increases, while that of flash attention increases approximately linearly. This results in a speedup of 3x over CUDNN and 4x over Pallas at 8k, and reaching speedups of 20-30x at context length 64k.

7.2 Ablation

We now demonstrate the importance of the various components of MuSe for maintaining sufficiently low error in approximating softmax attention. Here we will consider taken both early in training (after 200M tokens) and late (after 2B tokens). It can be seen in table 3 that removing the dipole corrections results in an 80% increase in error early in training and 15% late. We speculate that the greater contribution of the dipole corrections early in training corresponds to higher entropy attention patterns and lower magnitude queries, where the linear correction of the dipole is more significant than the higher order terms we neglect. We also see that removing the two-stage aspect of our algorithm (i.e., having a single query cluster) results in dramatic increases in error, with errors roughly doubling both early and late in training. Finally, as a sanity check we remove the monopole term of the algorithm to verify that we have not, e.g., simply linearized attention with the dipole term, and find that, indeed, the magnitude of error increases by 4-40 times corresponding to the vast majority of the output being error.

Table 3: We show the consequences of removing either the monopole or dipole components of our method or the two-stage component on relative squared error both early in training (after 200M tokens) and late (after 2B tokens).

Component Removed	Early (200M)		Late (2B)	
	Error	Change	Error	Change
None (full MuSe method)	0.025	–	0.195	–
Without Dipole	0.045	+80%	0.224	+15%
Without Two-stage	0.057	+128%	0.380	+95%
Without Monopole	0.960	+3741%	0.922	+373%

7.3 Pretraining

We now demonstrate the use of MuSe to obtain a wall-clock speedup of 12.2% in total runtime over our flash attention implementation at 16k context with minimal loss degradation of 0.36%, as seen in figure 3 and table 4. This is important for two reasons. Firstly, we thereby demonstrate that the extension from acausal to causal attention retains significant speed advantages in practice. Secondly, we demonstrate that the errors measured earlier in the forward pass, and whatever backward pass errors these come with, are acceptable in practice.

The model used here is a 7-layer 29M parameter GPT-like model using RoPE position embeddings where we have limited the context length of all layers except the central layer to 256 with a sliding window. The full hyperparameters
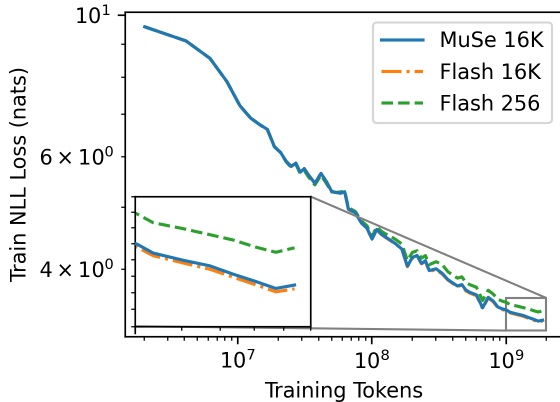
Figure 3: Training negative log-likelihood loss in nats versus number of training tokens for flash attention with context window 256 (CUDNN) and 16k (Pallas), and MuSe with context window 16k. The inset shows convergence behaviour in the range of 1–2 billion tokens, where it can be seen that the long context window significantly improves performance (reduction of 3.6% from 3.436 to 3.310 with flash attention) while the difference between MuSe and flash attention is small (0.36%).

Implementation	Window	Step (ms)	Loss
CUDNN Flash	256	69.7	3.436
Pallas Flash	16k	103.7	3.310
MuSe	16k	90.5	3.322

Table 4: Step time and final loss for the three training runs of figure 3 as a function of sliding window size (16k corresponds to full causal attention). Note that this window size refers to the single, central, global attention layer. All other layers use a constant window size of 256 and CUDNN Flash Attention.

used for these experiments may be found in appendix A. This allows us to isolate the effects of global attention by varying the context length of the central layer and swap out attention implementations for this layer to compare performance. In particular, we compare CUDNN Flash Attention with a sliding window of 256, Pallas Flash Attention with a global window (i.e., 16384), and MuSe with an overall window of 16k and a block size of 8k. This corresponds to the smallest configuration of the causal adaption of MuSe described in section 6 where half of the causally masked attention matrix uses a single large MuSe block and the rest is Pallas Flash Attention. We use the fastest MuSe clustering hyperparameters of table 1 for 64 clusters, as we consider this to be a reasonable speed/accuracy tradeoff.

It can be seen in table 4 that the reduction in training step time (i.e., a full update on one batch) using MuSe is 12.7% relative to the baseline of our Pallas flash attention. It can be seen that using long context attention is costing us 33.8 ms with flash attention, which we reduce by 39% to 20.6 ms with MuSe. This is impressive, because only half of the causally masked attention score matrix is using MuSe, so this corresponds to a roughly 78% reduction in time spent on that portion, lining up with the 76% reduction expected from our earlier acausal microbenchmarks at 8k context.

8   Conclusion

We have seen that MuSe achieves a complexity linear in context length for acausal attention and log-linear in context length for causal attention. We have shown that this translates to practical wall-clock speedups in practice with minimal loss degradation.

We believe that it should be possible in future work to push the cross-over point relative to flash attention lower through greater optimization of the implementation, especially for head dimensions less than 64, which we have not explored in this work. We are also excited to test MuSe in larger scale settings, since larger models should be able to exploit longer contexts than we explored here, potentially producing even greater speedups.

Acknowledgements

References

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *NeurIPS*, 2022.

Coleman Hooper, Sebastian Zhao, Luca Manolache, Sehoon Kim, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Multipole attention for efficient long context reasoning. *CoRR*, abs/2506.13059, 2025a.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*. OpenReview.net, 2020.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Trans. Assoc. Comput. Linguistics*, 9:53–68, 2021.

Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length LLM inference. In *ACL (1)*, pages 32631–32652. Association for Computational Linguistics, 2025b.

Junhui He, Junna Xing, Nan Wang, Rui Xu, Shangyu Wu, Peng Zhou, Qiang Liu, Chun Jason Xue, and Qingan Li. A^2ats: Retrieval-based KV cache reduction via windowed rotary position embedding and query-aware vector quantization. In *ACL (Findings)*, pages 12451–12463. Association for Computational Linguistics, 2025.

Kan Zhu, Tian Tang, Qinyu Xu, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context llms. *CoRR*, abs/2502.12216, 2025.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *CoRR*, abs/2006.04768, 2020.

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 *AAAI*, pages 14138–14148. AAAI Press, 2021.

Yanming Kang, Giang Tran, and Hans De Sterck. Fast multipole attention: A divide-and-conquer attention mechanism for long sequences. *arXiv preprint arXiv:2310.11960*, 2023.

Zhenhai Zhu and Radu Soricut. H-transformer-1d: Fast one-dimensional hierarchical attention for sequences. In *ACL/IJCNLP (1)*, pages 3801–3815. Association for Computational Linguistics, 2021.

Hut Barnes. A hierarchical o(nlogn) force-calculation algorithm. *Nature*, 1986.

V Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60(2): 187–207, 1985.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and João Carreira. Perceiver: General perception with iterative attention. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 4651–4664. PMLR, 2021.

Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. Luna: Linear unified nested attention. In *NeurIPS*, pages 2441–2453, 2021.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *ICLR*. OpenReview.net, 2021.

Rae, Jack W., Potapenko, Anna, Jayakumar, Siddhant M., Lillicrap, and Timothy P. Compressive transformers for long-range sequence modelling. In *ICLR*, 2020.

A   Hyperparameters

All experiments were performed on a single A100 GPU with 40GB of VRAM. All experiments were performed using the PG-19 dataset [Rae et al., 2020], consisting of roughly 28k English language books from Project Gutenberg published before 1919. To avoid wasting too many parameters on token embeddings with tokenizers optimized for internet text, we trained a specialized tokenizer on PG-19 with a vocabulary size of 16384.

Our architecture is a GPT-style architecture, modified to use RoPE position embeddings with a base inverse frequency of 100k in order to permit long context training without excessive parameter overhead. Our model has 29.4M parameters, 7 layers, hidden size 512, 8 heads of dimension 64 per layer, and an intermediate size in the feedforward layers of 2048. All layers bar the central layer use a sliding window of size 256 in their attention component, and use the CUDNN

Flash Attention implementation thereof. We vary the window size of the central layer, either having it be also local at 256, or equal to the full context length (in our experiments 16384). When comparing full context runtimes we use either our Pallas Flash Attention implementation in this central layer, or MuSe.

We train in mixed precision using bfloat16. We use a batch size of 4 and sequence length of 16384. We construct sequences by concatenating PG-19 together with end of sequence separators and cutting it into equal 16384 token chunks, which we then shuffle and batch. We train on a total of 2B tokens of text.

We use AdamW with $\beta_1 = 0.9$ and $\beta_2 = 0.95$ and a weight decay of 0.01. We use a cosine decay learning rate schedule with linear warmup over 1000 steps (batches) followed by decaying from a peak of 1.5×10^{-3} to 1.5×10^{-4} over the course of training. We clip gradients by a global norm of 1.0 in order to maintain stability at the very beginning of training.

B   Detailed Tables

This appendix contains extended tables comparing the runtime and accuracy of MuSe with different cluster counts and K-means hyperparameters. Specifically, Cap refers to the maximum cluster size as a fraction of the (necessarily predetermined) average cluster size, while Iters refers to the number of K-means iterations which are performed (i.e., assignment followed by recomputing centroids), before the final assignment of points to centroids. Note that, since MuSe itself computes weighted centroids, we are effectively getting an extra K-means iteration for free there. Table 5 shows results for an 8k acausal context, as in the main text, whereas results for a shorter 4k context may be found in table 6.

The main difference from the main text is that we vary Iters and Cap independently here for cluster count fixed at 64, and that we also include 4k context, but there are also a few extra datapoints. Specifically, we include Cap 1.25 for context 8k as a maximally fast setting (this is not practical at 4k since the required kernel block sizes are inefficiently small), and we include Cap 2.0 Iters 1 in the case of $C = 128$, since Cap 1.5 forces inefficiently small block sizes with $C = 128$ and is therefore pointlessly slow.

Table 5: Runtime for a combined forward and backward pass, and error of MuSe as clustering quality and number of clusters varies. Here we consider a batch size of 16, a context length of 8192, a head count of 8 and a head size of 64. For this problem size the runtime of our flash attention implementation in Pallas is 67.91ms, and that of Nvidia's cudnn flash attention is 51.74ms. The data used consists of queries, keys and values recorded from the long context attention layer at the end of pretraining for our small gpt model.

Cap	Iters	C	Runtime (ms)	Rel. Sq. Error
1.25	1	32	11.80	0.2650
1.5	1	32	12.37	0.2475
2.0	3	32	19.37	0.2148
4.0	5	32	29.35	0.2060
1.5	1	64	16.49	0.1946
2.0	1	64	18.23	0.1854
4.0	1	64	25.98	0.1807
1.5	3	64	21.33	0.1700
2.0	3	64	23.05	0.1642
4.0	3	64	31.05	0.1622
1.5	5	64	25.18	0.1633
2.0	5	64	27.02	0.1582
4.0	5	64	34.79	0.1571
1.5	1	128	36.12	0.1435
2.0	3	128	45.82	0.1179
4.0	5	128	72.88	0.1123

Table 6: Runtime for a combined forward and backward pass, and error of MuSe as clustering quality and number of clusters varies. Here we consider a batch size of 32, a context length of 4096, a head count of 8 and a head size of 64. For this problem size the runtime of our flash attention implementation in Pallas is 35.10ms, and that of Nvidia's cudnn flash attention is 26.65ms. The data used consists of queries, keys and values recorded from the long context attention layer at the end of pretraining for our small gpt model.

Cap	Iters	C	Runtime (ms)	Rel. Sq. Error
1.5	1	32	12.72	0.2568
2.0	3	32	18.80	0.2192
4.0	5	32	27.71	0.2095
1.5	1	64	18.75	0.1944
2.0	1	64	20.49	0.1835
4.0	1	64	28.22	0.1774
1.5	3	64	23.21	0.1673
2.0	3	64	24.91	0.1599
4.0	3	64	32.72	0.1576
1.5	5	64	26.85	0.1596
2.0	5	64	28.44	0.1527
4.0	5	64	36.45	0.1508
1.5	1	128	49.59	0.1373
2.0	1	128	49.09	0.1284
2.0	3	128	53.11	0.1123
4.0	5	128	79.15	0.1064