Hardware-aware Calibrated Clustered Attention for Efficient Visual Geometric Transformers

Weitian Wang[1,2], Rai Shubham[1], Cecilia De La Parra[1], Akash Kumar[2]
[1]Robert Bosch GmbH, Germany, [2]Ruhr University Bochum, Germany

***Abstract*—The Visual Geometry Grounded Transformer (VGGT) marks a significant leap forward in 3D scene reconstruction, as it is the first model that directly infers all key 3D attributes (camera poses, depths, and dense geometry) jointly in one pass. However, this joint inference mechanism requires global attention layers with extremely long sequences that causes a significant latency bottleneck. In this paper, we propose blockwise clustered attention (BC attention) to accelerate the global attention layers in VGGT. By limiting the clustering within HW-friendly neighborhood blocks, BC attention reduces the computation overhead of query clustering as well as the costly data movement between on- and off-chip memory. This enables BC attention to scale to long sequences and deliver practical latency improvements on GPUs. Moreover, we introduce a hashing hyperplane calibration method and a threshold-based error compensation method to reduce clustering errors efficiently, which is a bottleneck in the current clustered attention mechanism. Overall, our experiments on GPU demonstrate that calibrated BC attention accelerates the global attention layers by 2.10-2.63× and the whole backbone by 1.77-2.35× with negligible loss (1%) for large scenes. With a small performance loss (<5%), calibrated BC attention further achieves a 2.26-2.87× latency improvement on the global attention layers and a 1.90-2.55× improvement on the backbone.**

I. Introduction

The Visual Geometry Grounded Transformer (VGGT) [1] is a recently proposed feed-forward transformer model that directly infers all key 3D attributes of a scene from a variable number of views. By this direct inference, VGGT can outperform state-of-the-art methods while avoiding costly visual-geometry post-processing methods, marking an important breakthrough in 3D computer vision.

As shown in Fig 1, one of the key designs of VGGT is its alternating frame-wise and global attention. In the global attention layers, all tokens from different views participate in the multi-head attention computation. In practice, this approach results in extremely long token sequences (more than 20k tokens) even for small scenes. Thus, the global attention layers become the main latency bottleneck of VGGT, limiting its efficiency on medium and large scene reconstruction.

Motivated by the development of long-context large language models (LLMs) and vision language models (VLMs), many methods [2], [3] have been proposed to ease the high computational costs of long-sequence attention layers. These methods are mainly sparsity-based methods that exploit the pattern that the attention scores of LLMs and VLMs tend to concentrate on a small part of tokens. However, unlike LLMs and VLMs, the attention distributions of VGGT's global attention layers are non-sparse (rare drastic attention scores as shown in Fig. 2), making these sparsity-based methods not ideal.

Emails: {weitian.wang, shubham.rai, cecilia.delaparra}@bosch.com, akash.kumar@ruhr-uni-bochum.de

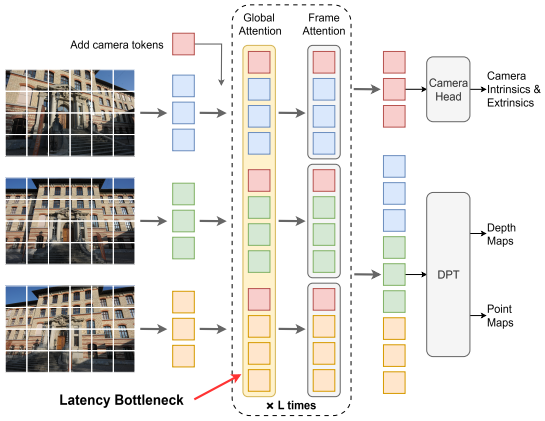

Fig. 1: Architecture overview of VGGT [1]. The global attention layers that perform attention computations across tokens from all views cause a significant latency bottleneck.

On the other hand, this non-drastic attention distribution pattern favors similarity-based methods as the attention weight distributions don't change largely from token to token. Clustered attention [4] was originally proposed to accelerate the attention layers of BERT [5]. Utilizing the similarity between queries, it approximates attention computation by grouping similar queries into clusters and computing attentions only for the cluster centroids; we further refer to this work as *vanilla clustered attention*. Despite its novelty, accelerating modern large vision transformers like VGGT on GPUs with vanilla clustered attention faces several challenges, including large computational and memory access overheads, as well as incompatibility with modern highly efficient attention kernels like FlashAttention [6].

To address these challenges, we propose *blockwise clustered attention* (BC attention) that revisits the idea of clustered attention for VGGT. With hardware-aware algorithmic co-design and optimized GPU implementation, BC attention achieves practical latency improvements on VGGT with negligible performance loss. While our experiments are demonstrated on NVIDIA GPUs, our method is applicable across most types of NVIDIA architectures and can be extended to other accelerator

systems. Our main contributions are summarized as follows:

- We introduce a hardware-aware blockwise clustering design that lowers the computation overhead and costly on- and off-chip data movement of vanilla clustered attention, enabling practical latency improvements on GPUs for long token sequences.
- We propose a novel gradient-based hashing hyperplanes calibration method using differentiable surrogates. Compared with randomly sampled hyperplanes used in vanilla clustered attention, adopting calibrated hyperplanes in the BC attention gives robust results that are more representative of the accurate attention results.
- We replace the unstructured top-k error compensation in vanilla clustered attention with a *structured* threshold-based error compensation method that effectively compensates the clustering error. Unlike unstructured attention, these threshold-based structured attention can be efficiently leveraged by high-performance attention kernels such as FlashAttention [6].

II. Related Work

In this section, we will go through some of the works towards efficiently accelerating attention layers in transformer models. State-of-the-art methods can be broadly classified into:

1) **Sparsity-based Attention Acceleration:** Sparse attention methods have recently gained popularity as an effective means to address the computational challenges of long-sequence attention. Utilizing the inherently sparse attention mechanism in LLMs, methods like Sparse Transformer [7] and BigBird [8] accelerate inference and extend maximum sequence lengths through local or block-based attention but often require retraining. StreamingLLM [9] enables LLMs to handle unlimited texts without fine-tuning by retaining the attention sinks (initial tokens) and the recent tokens. SparseVLM [3] accelerates VLMs through a text-guided training-free token pruning mechanism. For vision transformers (ViTs), SparseViT [10] and MixA-Q [11] discard or compress unimportant tokens with small magnitudes for latency improvements. However, these methods are all dependent on the sparse attention pattern, which VGGT doesn't exhibit distinctly.
2) **Similarity-based Attention Acceleration:**Another group of methods tries to save computation by clustering or merging similar tokens during attention. ToMe [12] merges similar tokens in vision transformers progressively through a bipartite graph matching algorithm, accelerating ViT without fine-tuning. ToMeSD [12] extends ToMe to accelerate diffusion models [13]. By merging tokens before each component of the block and unmerging them afterwards, ToMeSD reduces compute costs without changing the spatial layout of the tokens. One downside of ToMeSD is that it produces repetitive tokens after merging, potentially limiting the model's expressiveness. On the other hand, clustered attention [4] clusters similar query tokens for each head using Locality-Sensitive Hashing (SimHash) [14] and k-means [15]. The attention computation is then approximated by using the centroids to represent clustered query tokens. In this way, clustered attention saves computation costs without hurting the uniqueness of tokens.

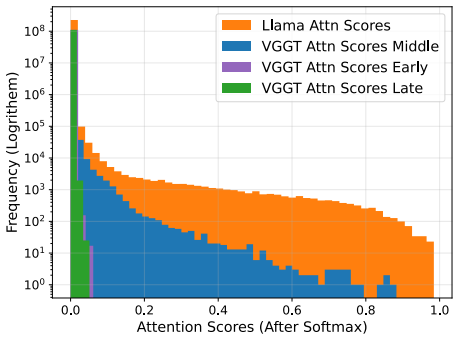

Fig. 2: Attention score distribution comparison between VGGT and Llama 3.1 8B [16]. VGGT has nearly uniform attention distribution in early and late layers. For middle layers, VGGT still has a more flat attention distribution than Llama.

In spite of showing promising results on BERT [5], clustered attention faces challenges when applied to even longer token sequences in VGGT:

- The complexity of clustering n query tokens globally into k clusters with k-means is $O(kn)$. Since the cluster number is proportional to n in practice, the overall clustering overhead grows quadratically with the token length. Additionally, large n and k result in large-sized temporary values like centroids during k-means that can't fit into the SRAM of GPUs, resulting in frequent on- and off-chip data movement.
- Vanilla clustered attention recomputes the top-k attention weights for each query, resulting in an unstructured sparse attention pattern that is incompatible with highly optimized GPU kernels such as FlashAttention [6], incurring high memory overhead and reduced efficiency.

III. Calibrated Blockwise Clustered Attention

A. Observations and Insights

Before going into details of BC attention, we want to first present the observations that led us to this solution. In Fig. 2, we show the attention score distributions of global attention layers in VGGT at different depths (3rd,11th,19th) and compare them with the distribution of an LLM (Llama 3.1 8B [16]). It can be observed that early and late global attention layers of VGGT have nearly uniform attention score distribution, meaning that each token is attending to all other tokens. For middle layers, the attention starts to concentrate more on certain tokens, but still not as drastically as in Llama, where a vast number of attention scores higher than 0.6 can be observed. From these observations, we infer that sparsity-based attention acceleration methods tailored for LLMs are not

ideal for VGGT because there is no small subset of tokens that can accumulate almost all the attention scores, especially for early and late global attention layers. On the other hand, this attention distribution pattern favors similarity-based methods like clustered attention, as the approximation errors are less pronounced for non-dominating tokens.

Then, we evaluate VGGT models using the cosine similarity metric. Fig. 3 illustrates the cosine similarity matrix between all the tokens in the first three frames of a scene. It can be observed that high similarity scores concentrate near the diagonal, meaning that similar tokens mainly reside in neighborhoods. This pattern arises because, unlike other transformer architectures such as BERT [5] and Stable Diffusion [13] which use static positional embeddings added once at the input, VGGT uses Rotary Position Embedding (RoPE) [17] applied at every layer. RoPE amplifies positional distinctions, decreasing the similarity between tokens that are far apart. This positional property motivates us to constrain the clustering within blocks to significantly reduce clustering overhead.

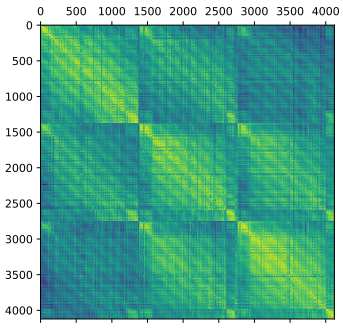

Fig. 3: Cosine similarity matrix between queries from the first three frames after adding the positional embedding (RoPE).

B. Hardware-aware Blockwise Clustered Attention

To understand our proposed approach, it is important to understand vanilla clustered attention. Given $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{L \times E}$ of a head in an attention layer [18], the main purpose of vanilla clustered attention is to reduce the computational complexity of the attention layer by approximating $\boldsymbol{X}$ in

$$\boldsymbol{X} = \text{softmax}(\frac{\boldsymbol{Q} \cdot \boldsymbol{K}^T}{\sqrt{d}}) \cdot \boldsymbol{V} \tag{1}$$

with $\boldsymbol{X'}$ in:

$$\boldsymbol{X'} = \text{scatter}(\text{softmax}(\frac{\boldsymbol{Q_c} \cdot \boldsymbol{K}^T}{\sqrt{d}}) \cdot \boldsymbol{V}, \boldsymbol{c_q})$$

where $\text{scatter}(\boldsymbol{M}, \boldsymbol{c})_{i,:} = \boldsymbol{M}_{c[i],:}$, $\boldsymbol{Q_c} \in \mathbb{R}^{k \times E}, k < L$ is the k centroids of L query tokens and $\boldsymbol{c_q} \in \{0, 1, \ldots, k-1\}^L$ is the cluster assignment of L query tokens. The cluster assignments are decided by performing k-means in Hamming space on the hash codes of queries $\boldsymbol{H} \in \{0, 1\}^L$ obtained using SimHash (More details in Sec. III-C). The centroid of each cluster is computed as the mean of all the queries belonging to that particular cluster. In this way, the computational complexity of (1) is reduced from $O(L^2 E)$ to $O(kLE)$.

Inspired by the observations above and to solve the challenges faced by vanilla clustered attention on long sequences, we propose a hardware-aware blockwise clustered attention (BC Attention) as shown in Alg. 1. Instead of the global clustering method used in vanilla clustered attention, BC attention uses blockwise clustering (line 2) as defined in Alg. 2. By clustering hash codes of queries within blocks of size B, blockwise clustering reduces the computational complexity of clustering L tokens into $\lfloor L/\gamma \rfloor$ clusters through I iterations of k-means from $O(IL^2/\gamma)$ to $O(ILB/\gamma)$. Additionally, if we consider GPU implementation, temporary values computed in k-means, like centroids, sample–centroid distance, etc., can fit inside on-chip registers and shared memory. Only one read and one write to HBM (*High Bandwidth Memory*) are required for the clustering of one block. Finally, given that k-means converges faster on smaller problem sizes [19], blockwise clustering can converge faster than global clustering, reducing the iteration number I in practice.

Algorithm 1 BC Attention: Approximated Attention Layer with Hardware-aware Block-wise Clustering

Require: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N \times H \times L \times E}$, block size B, number of iterations I, compress factor γ
Hyperparams: Calibrated hyperplanes $\boldsymbol{P_h} \in \mathbb{R}^{H \times 32 \times E}$, layerwise threshold τ_ℓ
Ensure: $\boldsymbol{X} \in \mathbb{R}^{N \times H \times L \times E}$

1: $\boldsymbol{H} = \text{SimHash}(\boldsymbol{Q}, \boldsymbol{P_h})$
2: $\boldsymbol{C_q} = \text{BlockCluster}(\boldsymbol{H}, B, I, \gamma)$
3: $k = \lfloor B/\gamma \rfloor$ ▷ Number of clusters in each block
4: $\boldsymbol{Q_c}, \boldsymbol{M} = \text{BlockAggregateThresh}(\boldsymbol{Q}, \boldsymbol{C_q}, B, k, \tau_\ell)$
5: $\boldsymbol{X_c} = \text{Attention}(\boldsymbol{Q_c}, \boldsymbol{K}, \boldsymbol{V})$
6: $\boldsymbol{X} = \text{BlockScatter}(\boldsymbol{X_c}, \boldsymbol{C_q}, B, k)$
7: $\boldsymbol{X}[\boldsymbol{M}] = \text{Attention}(\boldsymbol{Q}[\boldsymbol{M}], \boldsymbol{K}, \boldsymbol{V})$ ▷ Error compensation

Algorithm 2 BlockCluster: Hardware-aware Block-wise Clustering

Require: Hash codes $\boldsymbol{H} \in \mathbb{Z}^{N \times H \times L}$, block size B, number of iterations I, compression factor γ
Ensure: Cluster assignments $\boldsymbol{C_q} \in \mathbb{Z}^{N \times H \times L}$

Partition $\boldsymbol{H}$ to $N \times H \times \lceil L/B \rceil$ blocks of size at most B
for each block of hash codes $\boldsymbol{h_i}$ **do**
  $k = \lfloor B/\gamma \rfloor$ ▷ Number of clusters in each block
  Load $\boldsymbol{H^{(i)}}$ from HBM to on-chip SRAM
  Initialize $\boldsymbol{h_k}$, $\boldsymbol{d}$, $\boldsymbol{C_q^{(i)}}$ in SRAM
  for $iter = 1 \to I$ **do** ▷ Hamming distance k-means
    Update sample-centroids Hamming distances $\boldsymbol{d}$
    Update cluster assignments $\boldsymbol{C_q^{(i)}}$ according to $\boldsymbol{d}$
    Update centroids $\boldsymbol{h_k}$ by bit-wise majority voting
    Reinitialize centroids of empty clusters if needed
  end for
  Write final assignments $\boldsymbol{C_q^{(i)}}$ back to $\boldsymbol{C_q}$ on HBM
end for

To improve the approximation quality of BC attention (Alg. 1), we adopt a headwise calibrated hyperplane during the SimHash (line 1) and recompute the outputs of outliers based on a static layerwise threshold (line 4, 7). Details on these two mechanisms will be demonstrated in the following sections.

C. Hyperplane Calibration via Differentiable Surrogate

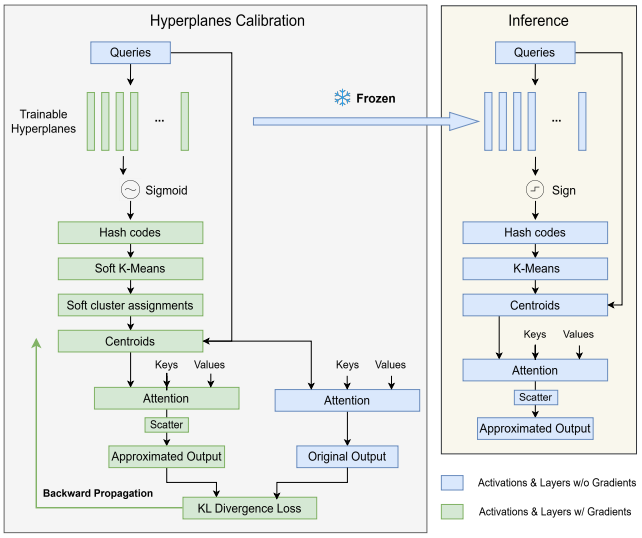

Fig. 4: Pipeline of hyperplanes calibration through differentiable surrogate

Similar to vanilla clustered attention, BC attention uses SimHash to generate hash codes that represent the directions of tokens. By representing the E dimension (64 in VGGT) query token with one `int32` hash code, SimHash largely reduces the overhead of computing cosine similarities between tokens.

In SimHash, for query tokens $\boldsymbol{Q} \in \mathbb{R}^{L\times E}$, A matrix $\boldsymbol{P_h} \in \mathbb{R}^{m\times E}$ that consists of m randomly sampled hyperplanes $\boldsymbol{p_i} \in \mathbb{R}^{E}$ is used to project $\boldsymbol{Q}$ to generate binary hash codes. For a query token $\boldsymbol{q} \in \mathbb{R}^{E}$, its binary hash vector $\boldsymbol{h} \in \{0,1\}^{m}$ is computed by taking the sign of the dot product between $\boldsymbol{q}$ and m hyperplanes, i.e.,

$$\boldsymbol{h} = [\text{sign}(\boldsymbol{p_0^T q}), \text{sign}(\boldsymbol{p_1^T q}), ..., \text{sign}(\boldsymbol{p_{m-1}^T q})] \tag{2}$$

In practice, we use $m = 32$ so that the binary hash vector $\boldsymbol{h}$ can be compacted into a single `int32` hash code, and hash codes of all queries are gathered in $\boldsymbol{H} \in \mathbb{Z}^{L}$. As has been proved in SimHash [14], the Hamming distance d_{ij} between m-bit hash codes h_i,h_j of queries $\boldsymbol{q_i}$ and $\boldsymbol{q_j}$ has the feature:

$$\text{cos\_sim}(\boldsymbol{q_i}, \boldsymbol{q_j}) = \cos(\pi \cdot \frac{\mathbb{E}[d_{ij}]}{m}) \tag{3}$$

This enables clustering query tokens based on similarities efficiently by clustering their hash codes in Hamming distance.

A known issue of SimHash [14] is that the randomly sampled hyperplanes lead to results with high variance [20]. Our aim is to solve this problem by learning sets of model-dependent hyperplanes on a calibration set so that the hyperplanes can better separate the inputs according to downstream losses without introducing extra overhead during inference. However, neither SimHash nor k-means used in clustered attention are differentiable, and therefore we propose a hyperplane calibration pipeline as described in Fig. 4.

The green pipeline mimics the centroid computation in BC attention with differentiable operations and continues the computation of attention output, which is essentially a gradient estimator of BC attention that *surrogates* it during hyperplane fine-tuning. Hence, we refer to this pipeline as a *differentiable surrogate* . As shown on the left side of Fig. 4, by replacing the non-differentiable BC attention layer with the differentiable surrogate, we can optimize the learnable hyperplanes by minimizing the loss between accurate and approximated outputs over a calibration set. After the fine-tuning, we fall back to the original, non-differentiable but efficient BC attention layers as shown on the right side, which now use the calibrated hyperplanes.

Considering the varying activation patterns in different heads and layers, instead of using a single set of 32 randomly sampled hyperplanes, we use a headwise learnable parameter $\boldsymbol{P_h} \in \mathbb{R}^{H\times 32\times E}$ for each layer. We replace the random hyperplanes in Eq. 2 with our learnable planes in $\boldsymbol{P_h}$ and replace the non-differentiable `sign` function with `signmoid` (inspired by HashNet [21]). Following these modifications, hash codes $\boldsymbol{H^{\mathcal{D}}} \in [0,1]^{N\times H\times L\times 32}$ dependent on $\boldsymbol{P_h}$ is computed with differentiable operations. Note that $\boldsymbol{H^{\mathcal{D}}}$ consists of hash code vectors in 32 dimensions that approximate the 32 bits in hash codes with floating point between 0 and 1.

Subsequently, we adopt k-means with soft assignment [22] to perform differentiable clustering of $\boldsymbol{H^{\mathcal{D}}}$. In experiments, we follow the soft assignment implementation from ClusterNet [23]. Soft assignments match samples with clusters via assignment probabilities obtained through a softmax function, which requires a similarity score rather than distances. Following Eq. 3, we define the similarity score of hash code vector $\boldsymbol{h_i^{\mathcal{D}}}$,$\boldsymbol{h_j^{\mathcal{D}}}$ from $\boldsymbol{H^{\mathcal{D}}}$ as:

$$\text{Sim}_{i,j} = \cos(\pi \cdot \frac{\|\boldsymbol{h_i^{\mathcal{D}}} - \boldsymbol{h_j^{\mathcal{D}}}\|_1}{m})$$

For each query block $\boldsymbol{Q^{(i)}}$ of size B, given a cluster number k, the soft k-means calculates a soft assignment $\boldsymbol{S} \in [0,1]^{B,k}$ such that $\sum_{j=0}^{k-1} s_{ij} = 1$ and s_{ij} represents the assignment probability of token $\boldsymbol{q_i}$ belonging to cluster j. Hence, we can compute all the centroids of the block $\boldsymbol{Q_c^{\mathcal{D}}} \in \mathbb{R}^{k,E}$ differentiably with respect to $\boldsymbol{P_h}$ as the weighted average of $\boldsymbol{Q^{(i)}}$ according to the assignment probabilities:

$$\boldsymbol{Q_c^{\mathcal{D}}} = \boldsymbol{S}^T \cdot \boldsymbol{Q^{(i)}}$$

Afterwards, any loss function $\mathcal{L} = f(\boldsymbol{Q_c^{\mathcal{D}}})$ can be used to train the learnable hyperplanes $\boldsymbol{P_h}$ through backpropagation. In experiments, we use the KL-divergence between the approximated output of the differentiable surrogate and the real output as the loss to optimize the hyperplanes. Input images from one scene in the test set are used as the calibration inputs.

D. Threshold-based Clustering Error Compensation

As shown in Eq. 3, the Hamming distance of hash codes may fail to accurately reflect similarities due to variance, resulting in poor clustering results for some queries. Hence, error compensation is required for clustered attention methods. Vanilla clustered attention [4] compensates for the error by recalculating the top-k attention scores for each cluster. However, this recomputation involves top-k key index calculation and top-k sparse attention computation for clusters of varying lengths. These unstructured operations are not compatible with attention kernels like FlashAttention [6], which has substantially optimized the structured attention computation on high-performance GPUs. As a result, these operations lead to high latency overheads for long sequences with a large number of clusters.

To efficiently compensate for the outliers during clustering, we propose a threshold-based method that identifies outliers by comparing the Euclidean distance between queries and the centroids of their clusters with a layer-wise threshold. For efficiency, we merge this error compensation with the blockwise centroid aggregation process.

Algorithm 3 BLOCKAGGREGATETHRESH: Blockwise Centroid Aggregation with Outlier Thresholding

Require: $\boldsymbol{Q} \in \mathbb{R}^{N\times H\times L\times E}$, cluster assignments $\boldsymbol{C_q} \in \mathbb{Z}^{N\times H\times L}$, block size B, cluster number k, threshold τ_ℓ
Ensure: Centroids $\boldsymbol{Q_c} \in \mathbb{R}^{N\times H\times K\times E}$, outlier mask $\boldsymbol{M} \in \{True, False\}^{N\times H\times L}$

Initialize $\boldsymbol{Q_c}$ of shape $N, H, \lceil L/B \rceil \cdot k, E$ on HBM
Partition $\boldsymbol{Q}$ and $\boldsymbol{C_q}$ into $N \times H \times \lceil L/B \rceil$ blocks
for each block i **do**
    Load query block $\boldsymbol{Q}^{(i)}$ and its cluster assignments $\boldsymbol{C_q}^{(i)}$ from HBM to on-chip SRAM
    Initialize centroids $\boldsymbol{Q_c}^{(i)}$ of block i in SRAM
    Aggregate centroids $\boldsymbol{Q_c}^{(i)}$ from $\boldsymbol{Q}^{(i)}$ using $\boldsymbol{C_q}^{(i)}$
    for each query j in $\boldsymbol{Q}^{(i)}$ **do**
        $c = \boldsymbol{C_q}^{(i)}[j]$
        $e = \|\boldsymbol{Q}^{(i)}[j] - \boldsymbol{Q_c}^{(i)}[c]\|_2$ ▷ Error computation
        if $e > \tau_\ell$ **then** ▷ Error thresholding
            $\boldsymbol{C_q}^{(i)}[j] = -1$
            Mark $\boldsymbol{Q_c}^{(i)}[c]$ as affected
        end if
    end for
    Recompute affected centroids in $\boldsymbol{Q_c}^{(i)}$
    Write final centroids $\boldsymbol{Q_c}^{(i)}$ of block i to $\boldsymbol{Q_c}$ on HBM
end for
$\boldsymbol{M} = \mathbf{1}[\boldsymbol{C_q} = -1]$ ▷ Outlier selection mask of -1

As shown in Alg. 3, we first load query blocks and their cluster assignments from HBM to SRAM to avoid frequent data movements. For each block, cluster centroids are aggregated as the mean of queries in each cluster. The error term is computed in place after the first aggregation and compared with a layerwise threshold. When marked as an outlier, the query is discarded from the centroid computation, and the centroids of clusters with outliers are recomputed. In this way, we merge the error computation and the outlier-free centroids aggregation into computation blocks that can be performed in parallel, and the data movements are avoided as much as possible. This blockwise centroid aggregation returns an outlier mask $\boldsymbol{M}$ which is used in Alg. 1, line 7, to select outliers for attention computation and fill the corresponding rows in the output. Note that $\boldsymbol{M}$ has shape $N \times H \times L$, so different heads can have different numbers of outliers. For computing this multi-head attention with various head lengths efficiently, we use the `flash_attn_varlen_func` kernel from FlashAttention.

To determine the layerwise thresholds, we use a calibration dataset as in Sec. III-C. For each layer, the error terms of all queries across all heads are sorted, and the threshold is determined as the 90th percentile. In this way, around 10% of the queries with the largest error are compensated. Experiments show that the threshold is model-dependent, exhibiting large variation across layers yet consistency across different inputs as shown in Fig. 5.

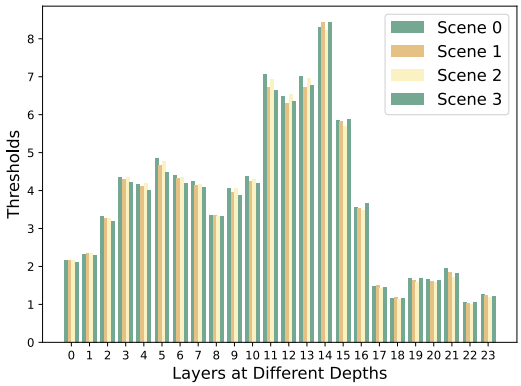

Fig. 5: Top 10% outlier thresholds of 4 random scenes across layers at different depths

IV. EXPERIMENTS

A. Approximation Evaluation

We replace the 24 global attention layers in the pretrained VGGT-1B model with our BC attention and compare its task performance on ETH3D [25] for point map estimation and DTU [26] for dense Multi-View Stereo (MVS) estimation with the standard VGGT. For point map estimation on ETH3D, we use the depth and camera heads (i.e., unprojecting the predicted depth maps to 3D using the predicted camera parameters) because it yields higher accuracy, as stated in the VGGT [1] paper. We report the Accuracy, Completeness, and Overall (Chamfer distance) for both tasks. Note that these metrics are relative distances that depend on the alignment and thresholding of point clouds, so the numbers we report here are not exactly the same as in the original VGGT [1] paper. However, we employ the same point cloud processing method for all the experiments for fairness.

As shown in Table I, on ETH3D, VGGT with calibrated BC attention shows negligible overall performance loss (1%)

TABLE I: Task Performance on VGGT [1]

Dataset	ETH3D			DTU		
Metrics	**Acc.↓**	**Comp.↓**	**Overall↓**	**Acc.↓**	**Comp.↓**	**Overall↓**
Standard Attention						
Baseline	0.871	0.529	0.700	0.448	0.437	0.443
Calibrated BC Attention						
$\gamma = 3$	**0.891**	**0.523**	**0.707**	0.528	0.437	**0.481**
$\gamma = 4$	**0.930**	**0.534**	**0.732**	0.566	0.440	0.503
Random BC Attention						
$\gamma = 3$	1.080	0.563	0.821	0.687	0.450	0.568
$\gamma = 4$	1.215	0.600	0.908	0.740	0.453	0.597
ToMeSD with 10% compensation						
$\gamma = 3$	1.490	0.803	1.147	1.335	0.479	0.907

TABLE II: Latency Comparison

#Frames	**75**	**100**	**150**	**200**
Backbone with Standard Attention				
Baseline	4.10s	6.91s	14.56s	25.11s
Backbone with BC Attention				
$\gamma = 3$	2.31s	3.50s	6.63s	**10.67s (2.35×)**
$\gamma = 4$	2.16s	3.28s	6.13s	**9.84s (2.55×)**
Standard Global Attention Layers				
Baseline	3.40s	5.97s	13.18s	23.27s
BC Global Attention Layers				
$\gamma = 3$	1.62s	2.59s	5.26s	**8.84s (2.63×)**
$\gamma = 4$	1.50s	2.39s	4.82s	**8.11s (2.87×)**

(a) Standard Attention (Baseline) (b) Calibrated Blockwise Clustered Attention (c) Random Blockwise Clustered Attention

Fig. 6: Comparison of reconstruction results using standard attention, blockwise clustered attention with calibrated and random hyperplanes. The reconstruction result using random hyperplanes exhibits noticeable noise compared to the other two results.

TABLE III: Task Performance on MapAnything [24]

Dataset	ETH3D		
Metrics	**Acc.↓**	**Comp.↓**	**Overall↓**
Standard Attention			
Baseline	0.114	0.125	0.120
Calibrated BC Attention			
$\gamma = 4$	**0.115**	**0.127**	**0.121**

with a compression factor $\gamma = 3$ (compress the query length by $3\times$) and small overall performance loss ($<5\%$) with a compression factor $\gamma = 4$. On the other hand, when using random hyperplanes, BC attention introduces rather large loss ($>17\%$). We show one example in Fig. 6 for qualitative evaluation. It can be observed that the reconstruction result using random hyperplanes exhibits noticeable noise compared to the baseline and calibrated results.

The DTU dataset contains ground-truth geometry of small objects obtained with a structured-light scanner at *millimeter-level* accuracy under controlled lab conditions. We evaluate BC attention on DTU to show that the overall performance loss introduced by calibrated BC attention is negligible (<0.05mm) under this high-fidelity setup as well.

In the last row of Table I, we also compared BC attention with ToMeSD [27]. We use ToMeSD to merge similar tokens in each frame before attention computation. For fairness, we also recompute the top 10% outliers during merging. It can be observed that ToMeSD gives worse results with compression factor $\gamma = 3$ than random BC attention with $\gamma = 4$.

To show BC attention's versatility in accelerating different models with similar attention design, we evaluate it on the MapAnything [24] model, a follow-up work of VGGT using a similar *alternating frame/global attention design*. Notably, while MapAnything does not apply RoPE in the attention layers, its tokens similarity patterns are still similar to those of VGGT. As shown in Table III, MapAnything with BC attention maintains similar results to the standard one.

B. Latency Performance

In this section, we evaluate BC attention's latency improvement on hardware. We use the NVIDIA H200 GPU as the evaluation hardware. All experiments are run with PyTorch [28] in Bfloat16 precision, and FlashAttention2 [6] is used as the attention kernel for both standard and BC attention.

As shown in Table II, we evaluate the latency improvement of BC attention compared to standard VGGT with 75 to 200 input frames, which is a usual range in 3D reconstruction tasks to guarantee adequate viewpoint coverage and geometric consistency of large scenes. Due to the clustering overhead and the latency of other parts of the model, the acceleration brought by BC attention becomes more evident for larger scenes.

With a compression factor $\gamma = 3$, BC attention achieves $2.10\times$ to $2.63\times$ latency improvement on all 24 global attention layers during inference, and $1.77\times$ to $2.35\times$ improvement on the whole backbone. With $\gamma = 4$, BC attention further brings $2.26\times$ to $2.87\times$ improvement on the global attention layers and $1.90\times$ to $2.55\times$ on the backbone.

V. Ablation Studies

A. Error compensation

To validate the importance of error compensation, we reduce the error compensation rate from 10% to 5%. This increases

the accuracy distance on ETH3D from 0.891 to 1.320, highlighting the necessity of sufficient error compensation for maintaining the approximation quality of BC attention.

B. Block Sizes Choice

In experiments in Sec. IV-A and Sec. IV-B, we use block size $B = 128$. In this section, we investigate the impact of different block sizes on BC attention's approximation quality.

TABLE IV: Task Performance using Different Block Size

Block Size	64	128	256
ETH3D Acc. ↓	0.906	0.891	0.129

In the above table, we list the reconstruction accuracy on ETH3D using different block sizes with the same number of k-means iterations. It can be observed from the table above that using block sizes of 64 and 128 has similar task performance. However, a bigger block size of 256 results in worse performance. We speculate that a bigger block size needs more iterations to converge during k-means.

VI. Conclusion

In this work, we proposed hardware-aware blockwise clustered attention (BC attention). By constraining clustering within local HW-friendly blocks, calibrating hash-based hyperplanes through a differentiable surrogate, and introducing threshold-based error compensation, our method achieves both accuracy robustness and practical GPU acceleration. Experiments show that calibrated BC attention delivers 2.1–2.6× speedup on global attention layers and 1.8–2.4× end-to-end backbone acceleration with negligible accuracy loss (1%). As our proposed approach focuses solely on local token similarities, future work could further explore exploiting framewise similarities, especially in use cases with input frames having temporal relations.

References

[1] J. Wang, M. Chen, N. Karaev, A. Vedaldi, C. Rupprecht, and D. Novotny, "Vggt: Visual geometry grounded transformer," in *Proceedings of the Computer Vision and Pattern Recognition Conference*, 2025, pp. 5294–5306.

[2] R. Xu, G. Xiao, H. Huang, J. Guo, and S. Han, "Xattention: Block sparse attention with antidiagonal scoring," 2025. [Online]. Available: https://arxiv.org/abs/2503.16428

[3] Y. Zhang, C.-K. Fan, J. Ma, W. Zheng, T. Huang, K. Cheng, D. Gudovskiy, T. Okuno, Y. Nakata, K. Keutzer *et al.*, "Sparsevlm: Visual token sparsification for efficient vision-language model inference," *arXiv preprint arXiv:2410.04417*, 2024.

[4] A. Vyas, A. Katharopoulos, and F. Fleuret, "Fast transformers with clustered attention," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 665–21 674, 2020.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[6] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," *arXiv preprint arXiv:2307.08691*, 2023.

[7] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[8] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[9] H. Liu, M. Zaharia, and P. Abbeel, "Ring attention with blockwise transformers for near-infinite context," *arXiv preprint arXiv:2310.01889*, 2023.

[10] X. Chen, Z. Liu, H. Tang, L. Yi, H. Zhao, and S. Han, "Sparsevit: Revisiting activation sparsity for efficient high-resolution vision transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2061–2070.

[11] W. Wang, R. Shubham, C. De La Parra, and A. Kumar, "Mixa-q: Revisiting activation sparsity for vision transformers from a mixed-precision quantization perspective," *arXiv preprint arXiv:2507.19131*, 2025.

[12] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman, "Token merging: Your vit but faster," *arXiv preprint arXiv:2210.09461*, 2022.

[13] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[14] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 2002, pp. 380–388.

[15] J. MacQueen, "Multivariate observations," in *Proceedings ofthe 5th Berkeley Symposium on Mathematical Statisticsand Probability*, vol. 1, 1967, pp. 281–297.

[16] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, and et. al., "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[17] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *Neurocomputing*, vol. 568, p. 127063, 2024.

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

[19] I. Davidson and A. Satyanarayana, "Speeding up k-means clustering by bootstrap averaging," in *IEEE data mining workshop on clustering large data sets*, vol. 25, no. 2, 2003, pp. 179–183.

[20] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian, "Super-bit locality-sensitive hashing," *Advances in neural information processing systems*, vol. 25, 2012.

[21] Z. Cao, M. Long, J. Wang, and P. S. Yu, "Hashnet: Deep learning to hash by continuation," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5608–5617.

[22] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*. PMLR, 2016, pp. 478–487.

[23] C. Chen, G. Li, R. Xu, T. Chen, M. Wang, and L. Lin, "Clusternet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4994–5002.

[24] N. Keetha, N. Müller, J. Schönberger, L. Porzi, Y. Zhang, T. Fischer, A. Knapitsch, D. Zauss, E. Weber, N. Antunes *et al.*, "Mapanything: Universal feed-forward metric 3d reconstruction," *arXiv preprint arXiv:2509.13414*, 2025.

[25] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger, "A multi-view stereo benchmark with high-resolution images and multi-camera videos," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[26] R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanæs, "Large scale multi-view stereopsis evaluation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2014, pp. 406–413.

[27] D. Bolya and J. Hoffman, "Token merging for fast stable diffusion," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 4599–4603.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.