Journal of Systems Architecture 175 (2026) 103770

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

HAGC: A Hardware-Aware Gradient Compression framework for distributed deep learning☆

Aiqiang Yang [a,*], Jie Liu [a,b,c], Bo Yang [a,b,c], Xiang Zhang [a], Qinglin Wang [a,b,c], Zeyao Mo [d], Keqin Li [e]

[a] National University of Defense Technology, Deya Road, Changsha, 410000, China
[b] National Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Deya Road, Changsha, 410073, China
[c] Laboratory of Digitizing Software for Frontier Equipment, National University of Defense Technology, Deya Road, Changsha, 410073, China
[d] China Academy of Engineering Physics, Mianyang, China
[e] SUNY New Paltz, 1 Hawk Drive, New Paltz, 12561, NY, USA

ARTICLE INFO

Dataset link: https://github.com/airayangbovo1/HAGC-MAIN

Keywords:
Dynamic quantization
Tensor Cores
Bilateral Hadamard Transform
Hardware-aware model

ABSTRACT

Distributed deep learning faces a persistent communication bottleneck as models scale exponentially. Standard software-based compression efforts often fail to exploit specialized GPU units, resulting in a computational lag that offsets intended bandwidth gains. This paper introduces Hardware-Aware Gradient Compression (HAGC), a framework that offloads the entire compression workload to hardware-native operations. Restructuring the Bilateral Hadamard Transform from memory-bound recursion into dense matrix operations allows HAGC to execute directly on Tensor Core mixed-precision units. This structural synchronization masks compression latency, while an integrated error feedback mechanism ensures convergence stability. The framework supports a systematic co-design strategy that calibrates training parameters to align processor capacity with training stability. Empirical evaluations on NVIDIA A100 GPUs demonstrate substantial performance gains, achieving up to a 3.15× end-to-end speedup for data-intensive models such as VGG-19. The resulting framework reduces energy consumption by 2.9× and memory footprint by nearly 4× , while maintaining model fidelity comparable to full-precision baselines. Such findings validate that bridging algorithmic design with physical hardware capabilities is essential for building sustainable, high-performance distributed systems.

1. Introduction

Deep learning models have achieved significant breakthroughs in natural language processing and computer vision. As model parameters grow exponentially, distributed learning across multiple nodes becomes essential to efficiently leverage computing resources. A primary bottleneck in this paradigm is the significant communication overhead caused by transmitting massive gradients and parameters [1]. While various Gradient Compression (GC) techniques have been proposed to mitigate this—ranging from sparsification (e.g., Deep Gradient Compression [2]) to low-rank approximation (e.g., PowerSGD [3]) and quantization (e.g., QSGD [4])—they often introduce a new challenge: the computational overhead of the compression and decompression steps themselves [5].

Existing fixed quantization methods effectively reduce communication volume but frequently suffer from gradient overflow and precision loss, leading to degraded model accuracy [6]. More importantly, many state-of-the-art (SOTA) compression algorithms are implemented purely in software, failing to exploit the specialized acceleration units available in modern GPUs, such as NVIDIA's Tensor Cores. This creates a hardware–software mismatch: while the network transmission is compressed, the GPU utilization during the compression phase remains suboptimal, limiting the total training throughput.

To address these challenges, this paper proposes Hardware-Aware Gradient Compression (HAGC), a unified framework that synergizes dynamic quantization with Tensor Core Unit (TCU) acceleration. Unlike traditional approaches, HAGC integrates a dynamic quantization scheme based on tensor statistics (standard deviation and distribution) to adaptively adjust intervals, effectively eliminating overflow while maintaining high precision. The pipeline enhances quantization robustness against disruptive outliers by integrating a Bilateral

☆ This work was supported by the National Key Research and Development Program of China [2023YFA1011704].
* Corresponding author: Aiqiang Yang, National University of Defense Technology, Changsha 410000, China. E-mail: yangaiqiang0644@nudt.edu.cn.
E-mail addresses: yangaiqiang0644@nudt.edu.cn (A. Yang), liujie@nudt.edu.cn (J. Liu), yangbo78@nudt.edu.cn (B. Yang), zhangxiang08@nudt.edu.cn (X. Zhang), wangqinglin@nudt.edu.cn (Q. Wang), zeyao_mo@iapcm.ac.cn (Z. Mo), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.sysarc.2026.103770
Received 18 December 2025; Received in revised form 11 February 2026; Accepted 4 March 2026
Available online 9 March 2026

Hadamard Transform, which we have specifically adapted to meet hardware execution constraints.

The main innovation of this work lies in the optimization of these algorithmic components, accounting for hardware constraints. The computationally intensive matrix operations underlying Hadamard's contraction/expansion and transformation processes are directly mapped onto the tensor core. By leveraging TCU's mixed-precision computing, the HAGC compression pipeline's computational latency is significantly reduced, transforming a potentially costly operation into one with negligible cost.

The key contributions of this paper are summarized as follows:

- *The HAGC Framework:* A TCU-accelerated communication framework is presented that integrates dynamic quantization, error response, and a Bilateral Hadamard transform. This algorithm has been specially optimized for the Tensor Core architecture to minimize compression latency and optimize throughput.
- *Hardware–Software Co-Design:* A conscious hardware-adjustment strategy has been proposed to tune hyperparameters based on processor capabilities and tensor properties, thereby ensuring stable learning in large-batch scenarios.
- *Systematic Evaluation:* Experience has shown that HAGC outperforms the standard baseline. HAGC not only accelerates training productivity from start to finish but also maintains model accuracy at a level comparable to high-precision training, thereby amply demonstrating its effectiveness in distributed deep learning.

Organization: The structure of this paper is as follows: Section 2 covers background and related work; Section 3 describes the proposed HAGC methodology and hardware implementation; Section 4 presents experimental results and ablation studies; and Section 6 provides concluding remarks.

2. Background and motivation

2.1. Hadamard transform in gradient compression

The Hadamard transform uses orthogonal matrices to reorganize data using elementary addition and subtraction operations. Defined as $y = H_n \times x$, the matrix relies on the Hadamard matrix $H_n \in \{-1,+1\}^{2^n \times 2^n}$, constructed recursively:

$$H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix} \tag{1}$$

In distributed machine learning, the Random Hadamard Transformation (RHT) is crucial for mitigating the impact of gradient compression discrepancies. By "pivoting" the gradient vector, the RHT distributes heavy loads across all dimensions to approximate a Gaussian distribution. Recent works have successfully applied this property to distributed settings; for instance, the Tensor Homomorphic Compression (THC) framework by Li et al. [7] utilizes RHT to enable efficient in-network aggregation on programmable switches.

However, a technical distinction must be made regarding hardware utilization. Existing approaches like THC primarily focus on the mathematical tensor structure for aggregation logic, typically implementing the transform using standard software-based kernels (e.g., Fast Walsh–Hadamard Transform). As noted in prior studies [8], these recursive algorithms ($O(N \log N)$) involve non-sequential memory access patterns that are highly inefficient and memory-bound on GPUs. Crucially, these software-centric methods do not leverage NVIDIA Tensor Cores – specialized hardware units designed for high-throughput mixed-precision matrix operations – leaving these powerful resources idle during the compression phase.

To bridge this gap, our HAGC framework proposes a bilateral block strategy (Section 3). Unlike prior implementations, we reformulate the transformation into a series of dense matrix multiplications. This architectural shift allows us to offload the compression workload directly to Tensor Cores, effectively transforming the bottleneck from memory-bound recursion to compute-bound execution.

2.2. Dynamic quantization

Quantization reduces communication volume by mapping floating-point values to low-bit integers. Dynamic quantization adapts the bounds $[x_{\min}, x_{\max}]$ based on the real-time statistics of the tensor X [9–11]:

$$Q(X) = \text{round}\left(\frac{X - x_{\min}}{x_{\max} - x_{\min}} \cdot (2^b - 1)\right) \tag{2}$$

Compared to static quantization, dynamic quantization is essential for training as data distributions shift rapidly [12,13].

Although dynamic approaches like DAdaQuant [12] improve accuracy, they introduce a significant penalty: the need to scan large tensors to compute statistics at every iteration [14]. In high-bandwidth environments (e.g., NVLink), this computational overhead can paradoxically become the new bottleneck, slowing down the training process despite reduced communication volume. HAGC addresses this by integrating statistical computation directly into the Tensor Core pipeline, hiding the cost of dynamic adaptation.

2.3. Error feedback mechanisms

Error feedback (EF) is a critical technique to maintain the convergence of deep neural networks by accumulating quantization residuals [15]. It has been extensively studied for handling massive datasets [16,17] and adaptive stochastic gradient methods [18,19]. The utility of EF extends to diverse scenarios, including large-scale training (e.g., SignSGD [20]) and privacy-preserving frameworks [21,22].

While EF guarantees convergence, it requires element-wise read–modify–write operations ($G_{t+1} \leftarrow G_{t+1} + E_t$). Current implementations typically execute these on standard CUDA cores, consuming significant memory bandwidth [23]. Unlike global compensation methods, our proposed HAGC framework employs a fused kernel design, where error compensation is performed in-register before Tensor Core operations, minimizing global memory traffic.

2.4. Tensor cores and hardware acceleration

Efficient tensor computations are the backbone of modern AI, supporting applications ranging from neuroscience to NLP [24] and signal processing [25,26]. NVIDIA's Tensor Cores provide specialized hardware for mixed-precision matrix multiply-accumulate (MMA) operations [27,28], accelerating workloads like FFT [29] and deep learning compilers [30].

While originally designed for deep learning, the massive throughput of Tensor Cores has inspired research into general-purpose algorithm acceleration. Recent works have successfully repurposed these mixed-precision units for fundamental parallel primitives. For instance, Navarro et al. [31] demonstrated that arithmetic reductions can be encoded as a set of chained MMA operations, achieving significantly higher bandwidth than standard CUDA cores. Similarly, specialized algorithms for parallel scan (prefix sum) have been reformulated into matrix multiplication forms [32], enabling Tensor Cores to process data-dependent sequences efficiently. In the domain of discrete mathematics, Quezada and Navarro [33] utilized Tensor Cores to accelerate fractal generation (e.g., Sierpinski triangles) by mapping thread-space coordinates to compact data-space representations via hardware-accelerated matrix transformations.

These advancements highlight the versatility of Tensor Cores for non-standard computational tasks. However, within the specific domain of gradient compression for distributed learning, this potential remains

largely untapped. Despite their power, Tensor Cores are primarily utilized for model forward/backward propagation [34]. Existing compression libraries (e.g., for QSGD or PowerSGD) rarely utilize Tensor Cores for the compression step itself, leaving these powerful units idle during communication phases.

This work bridges this gap. HAGC reformulates the compression pipeline – specifically the Hadamard transform and quantization – as Tensor Core-compatible GEMM operations, unlocking significant throughput gains that purely software-based methods cannot achieve.

Recent works leverage Tensor Cores for low-precision training, such as FP8 acceleration [35] and LLM optimization [36]. However, HAGC differs in two key aspects:

1. Objective: Unlike computation-centric FP8 methods that reduce calculation latency, HAGC targets communication bottlenecks. By compressing gradients before transmission, it yields benefits even in bandwidth-constrained environments where computation is not the limiting factor.

2. Stability: While native formats often rely on simple truncation, HAGC integrates Randomized Block Hadamard Transform with error feedback. This guarantees unbiased estimation [37] and superior convergence stability compared to direct low-precision casting.

3. Methodology: The HAGC framework

With the increasing complexity of deep learning applications [24,38], the computational demands for dense tensor operations have escalated [39]. To address the communication bottlenecks in distributed training while leveraging modern hardware accelerators [25,26], this work presents the Hardware-Aware Gradient Compression (HAGC) framework.

3.1. HAGC system overview

The HAGC framework shifts the training bottleneck from network bandwidth to readily available on-chip compute capability. Unlike software-based compression which competes for CPU resources, HAGC offloads the compression pipeline to NVIDIA Tensor Cores (TCUs). The workflow consists of three synchronized stages: (1) Error Compensation: Correcting local gradients using accumulated residuals. (2) TCU-Accelerated Transformation: A reformulated Bilateral Hadamard Transform (BHT) executed via Warp Matrix Multiply-Accumulate (WMMA) instructions. (3) Dual-Adaptive Quantization: Compressing gradients based on real-time statistical feedback.

The discretized state vector $\mathbf{q}_t$ is calculated as follows:

$$\mathbf{q}_t = \text{Round}\left(\text{Clamp}\left(\frac{\mathbf{v}_t}{\Delta_t}, -2^{b-1}, 2^{b-1}-1\right)\right). \tag{3}$$

Finally, the error accumulation is updated: $\mathbf{e}_t = \mathbf{v}_t - \mathcal{Q}^{-1}(\mathbf{q}_t)$.

3.2. Stage 1: Dynamic quantization with error feedback

Traditional static measurements often fail to capture the high dynamic range of gradients. HAGC uses a dual compensation mechanism that adjusts measurement parameters in both the temporal and spatial domains.

Let $\mathbf{g}_t \in \mathbb{R}^N$ be the gradient vector at iteration t. First, the error function is applied to obtain the corrected gradient $\mathbf{v}_t$:

$$\mathbf{v}_t = \mathbf{g}_t + \mu\mathbf{e}_{t-1}, \tag{4}$$

where $\mathbf{e}_{t-1}$ is the cumulative error and μ is the step size.

The discretization scale factor Δ_t adjusts in response to the transmission probability p_t. Temporally, the update rule employs an exponential moving average:

$$\Delta_t = \begin{cases} \alpha \cdot \Delta_{t-1}, & \text{if } p_t > \gamma \text{ (Expand)} \\ \beta \cdot \Delta_{t-1}, & \text{otherwise (Shrink)} \end{cases} \tag{5}$$

where $\alpha > 1$ and $\beta < 1$ control the adaptation rate.

Thanks to the dynamic calibration of Δ_t, HAGC imposes a strict upper limit on counting noise. Let $\mathcal{Q}(\cdot)$ denote the quantization-reconstruction process, the quantization noise is bounded by:

$$\mathbb{E}[\|\mathbf{g}_t - \mathcal{Q}(\mathbf{g}_t)\|^2] \le \frac{N\Delta_t^2}{4}. \tag{6}$$

Crucially, this bounded variance property, combined with the error feedback mechanism in Eq. (4), satisfies the theoretical prerequisites for convergence in non-convex optimization, as established in [40]. This ensures that neither the bias from quantization nor the dynamic scaling compromises the training trajectory.

The discretized state vector $\mathbf{q}_t$ (integer representation) is calculated as follows:

$$\mathbf{q}_t = \text{Round}\left(\text{Clamp}\left(\frac{\mathbf{v}_t}{\Delta_t}, -2^{b-1}, 2^{b-1}-1\right)\right). \tag{7}$$

Finally, the error accumulation is updated by calculating the residual between the corrected gradient and its reconstructed value:

$$\mathbf{e}_t = \mathbf{v}_t - \mathcal{Q}^{-1}(\mathbf{q}_t), \tag{8}$$

where $\mathcal{Q}^{-1}(\cdot)$ denotes the de-quantization operator, defined as $\mathcal{Q}^{-1}(\mathbf{q}_t) = \mathbf{q}_t \cdot \Delta_t$.

3.3. Stage 2: Bilateral Hadamard transform on tensor cores

The standard fast Walsh–Hadamard transform (FWHT) relies on recursive operations (O(N log N)) that produce out-of-phase memory access patterns, leading to cache misses on the GPU. HAGC recodes this into a computationally bounded blockwise operation.

The HAGC-BHT transformation as:

$$\mathbf{Y} = \mathbf{D}_1\mathbf{H}_{block}\mathbf{D}_2\mathbf{V}, \tag{9}$$

where $\mathbf{V} \in \mathbb{R}^{m\times B_{TC}}$ is the reshaped input matrix. The components of this transformation frame are illustrated in Fig. 1. $\mathbf{D}_1$ and $\mathbf{D}_2$ are diagonal matrices by default. Fig. 1(b) is constructed in such a way as to guarantee the inversion of the sign of the determinant, while $\mathbf{H}_{block}$ is constructed using the dense 16 × 16 Hadamard kernel shown in Fig. 1a.

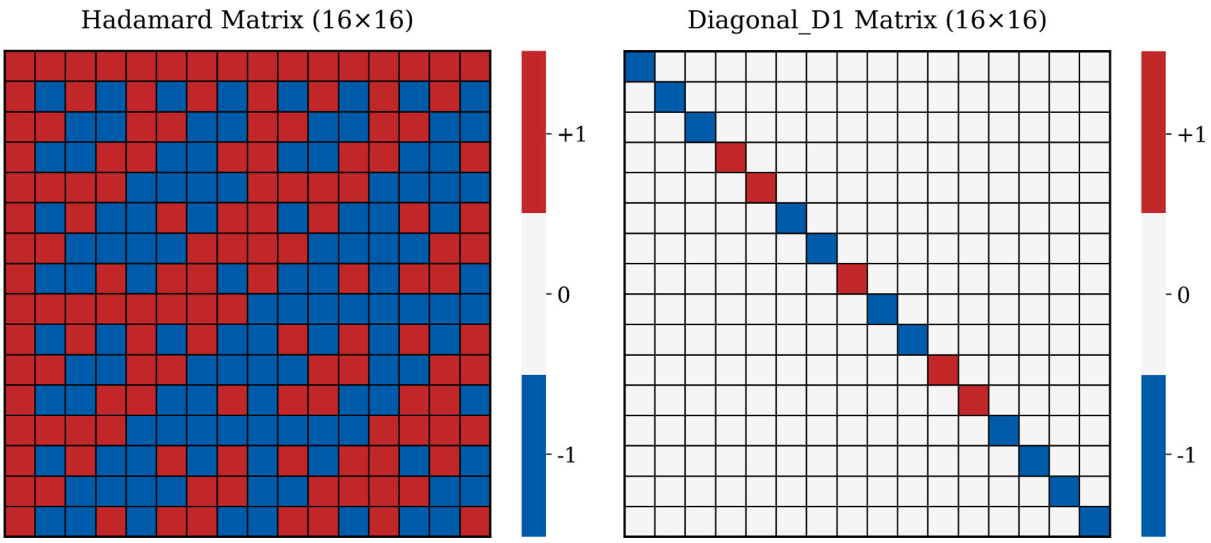
To leverage the specialized processing power of NVIDIA's Ampere Tensor cores, H_{block} is organized in a tile-based format with 16 × 16 Hadamard kernels; this organization is essential to maintain compatibility with the hardware primitive `wmma::mma_sync`. By reorganizing the required 16 × 16-cell granularity, the framework effectively circumvents the overhead associated with memory retrieval or irregular data layouts. As illustrated in Fig. 2, this architecture allows direct mapping to the hardware root, where localized blocks are treated as FP16 floating-point numbers and aggregated into FP32, thereby bypassing the scalar operation unit.

- *Warp-Level Parallelism:* Each of the 32 threads processes a 16 × 16 grid independently, thereby eliminating the additional overhead of thread synchronization.
- *Shared Memory Hierarchy:* Loading a small 16 × 16 Hadamard kernel into shared memory (L1 cache) achieves near-perfect cache hit rates, thereby bypassing the bottleneck of global memory latency in the standard FWHT.

3.4. Stage 3: Hardware-aware optimization

To bridge the gap between theoretical algorithmic efficiency and actual execution efficiency on NVIDIA A100 GPUs, HAGC uses a strict hardware–software co-design strategy.

The flow illustrated in Fig. 3 shows a strategy that deliberately avoids the pitfalls of heuristic guesswork; instead, it follows a systematic, top-down dictionary of causes and effects:



(a) Hadamard kernel ($\mathbf{H}_{16}$). (b) Randomized diagonal ($\mathbf{D}_1$).

Fig. 1. Kernel visualization for Eq. (9). (a) The 16 × 16 Hadamard kernel serves as the dense mixing block. (b) The sparse diagonal matrix $\mathbf{D}_1$ provides randomized sign flipping, ensuring unbiased compression. Both are aligned to the 16 × 16 Tensor Core granularity.

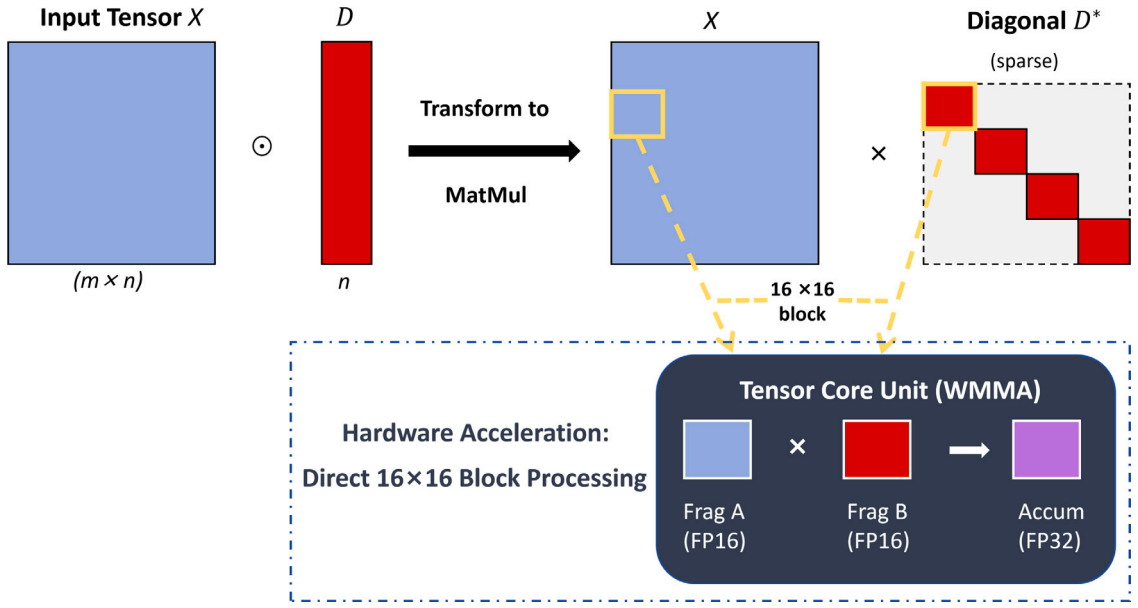


Fig. 2. Mapping BHT to Tensor Cores. The framework breaks down the calculation $\mathbf{Y} = \mathbf{D}_1\mathbf{H}\mathbf{D}_2\mathbf{V}$ into 16 × 16 kernels in order to exploit the internal parallelism of modern GPUs. This architectural choice is complemented by a mixed precision mode, in which FP16 inputs are passed to FP32 accumulators, allowing the entire processing pipeline to fully utilize Ampere's transposition capabilities.

1. Physical Layer (Top): Asymmetric channels of the A100 architecture (HBM bandwidth and Tensor Core square area requirements).
2. Logic Layer (Middle): Mathematical formulas that convert these physical quantities into algorithmic parameters.
3. Parameter Layer (Bottom): Latest implementation parameters (group size, learning rate) derived from the logic layer.

To guide future implementations, we detail two critical hardware-specific configurations. Tensor Cores require memory access to be aligned to 16-byte boundaries. For unaligned embedding dimensions (e.g., $N \pmod{16} \neq 0$), our kernel dynamically applies zero-padding to the nearest multiple of 16. As analyzed, this overhead is negligible (<1.5%) for typical model architectures. When loading 16 × 16 blocks into shared memory, standard addressing can cause bank conflicts, serializing access. To mitigate this, we implemented a padded memory layout by allocating shared memory tiles with a stride of `[TILE_SIZE + 1]`. This skewing technique ensures that columns of the tile map to distinct memory banks, allowing fully parallel access and maximizing the throughput of the `wmma` instructions.

3.4.1. *Memory alignment (The granularity constraint)*

As shown on the right side of the diagram in Fig. 3, Tensor Core kernels require strict memory ordering. The padding operation $\mathcal{P}(\cdot)$ is

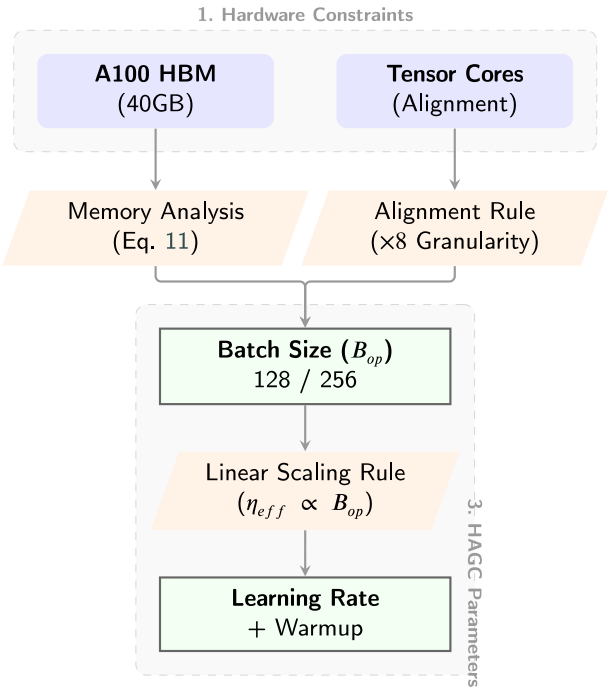


Fig. 3. Hardware–software dependency graph.

defined such that, for an input dimension N:

$$\tilde{N} = \left\lceil \frac{N}{B_{TC}} \right\rceil \times B_{TC}, \tag{10}$$

where $B_{TC} = 16$. A concrete illustration of this mechanism is shown in Fig. 4.

A potential concern with this fixed blocking is the overhead for models with unaligned embedding dimensions. We analyze two scenarios:

- Aligned Architectures (Standard): Most modern architectures, particularly in NLP, use hidden dimensions that are multiples of 16. In these cases, HAGC incurs zero padding overhead, fully leveraging Tensor Core acceleration.
- Unaligned Architectures (Edge Case): For dimensions not divisible by 16, HAGC applies zero-padding to the nearest alignment boundary. This overhead is negligible because the padding quantity is strictly upper-bounded by the block size ($<B_{TC}$), representing a diminishing fraction of the total data volume as model dimensionality grows. Consequently, this minor memory cost is effectively amortized by the substantial throughput boost provided by Tensor Core acceleration.

Ultimately, this alignment mechanism ensures that the restructuring operation creates a memory layout compatible with the unified memory architecture, thereby maximizing DRAM bandwidth utilization.

3.4.2. Automated batch sizing (The capacity constraint)

The left branch of Fig. 3 deals with memory efficiency. In order to take advantage of the massive transfer from the Tensor Core, the optimal local batch size B_{op} is derived as follows:

$$B_{op} = \left\lfloor \frac{M_{\mathrm{G}} \cdot \gamma_{util}}{P_{\mathrm{model}} \cdot S_{\mathrm{state}}} \right\rfloor_{\times 8}, \tag{11}$$

where M_{G} is the memory capacity of the GPU (40 GB) and $\lfloor \cdot \rfloor_{\times 8}$ imposes a scheduling constraint.

Table 1
Theoretical complexity comparison (per iteration).

Method	Time complexity	Memory access	Hardware suitability
Standard FWHT	$O(N \log N)$	Strided (Poor)	CPU/Scalar GPU
Random rotation	$O(N^2)$	Continuous	Tensor Cores
HAGC (Ours)	$\mathbf{O(N \cdot B_{TC})}$	**Coalesced (Optimal)**	**Tensor Cores**

Algorithm 1 HAGC Training Procedure

Require: Gradient $\mathbf{g}_t$, Error $\mathbf{e}_{t-1}$, Scale Δ_{t-1}, Block size B_{TC}.
Ensure: Compressed $\mathbf{q}_t$, Updated $\mathbf{e}_t$.
Phase 1: Compensation & Alignment
$\mathbf{v}_t \leftarrow \mathbf{g}_t + \mathbf{e}_{t-1}$
if $N \mod B_{TC} \neq 0$ **then** $\tilde{\mathbf{v}}_t \leftarrow \mathrm{Pad}(\mathbf{v}_t)$
end if
Reshape $\tilde{\mathbf{v}}_t$ to $\mathbf{V} \in \mathbb{R}^{m \times B_{TC}}$
Phase 2: TCU Transformation
Load $\mathbf{H}_{block}$ to Shared Memory
$\mathbf{Y} \leftarrow \mathbf{D}_1 \cdot (\mathbf{H}_{block} \cdot \mathbf{V}) \cdot \mathbf{D}_2$ ▷ WMMA instruction
Phase 3: Adaptive Quantization
Update Δ_t based on overflow statistics of $\mathbf{Y}$
$\mathbf{q}_t \leftarrow \mathrm{Round}(\mathbf{Y}/\Delta_t \cdot (2^{b-1} - 1))$
Phase 4: Error Update
$\mathbf{e}_t \leftarrow \mathbf{v}_t - \mathrm{Unpad}(\mathrm{InverseBHT}(\mathrm{Dequantize}(\mathbf{q}_t)))$
return $\mathbf{q}_t, \mathbf{e}_t$

3.4.3. Distributed stability (The scaling consequence)

Finally, as shown in the bottom of Fig. 3, the determination of B_{op} necessitates a corresponding adjustment in optimization dynamics. To this end, the Linear Scaling Rule is applied:

$$\eta_{eff} = \eta_{base} \times \frac{B_{op} \times N_{GPU}}{B_{base}}. \tag{12}$$

This ordering procedure ensures that the HAGC makes the most of the equipment transfer while maintaining joint stability.

3.4.4. Complexity and overhead analysis

To rigorously demonstrate the effectiveness of HAGC, compare with Table 1 for theoretical complexity using standard methods.

While the full rotation matrix has a complexity of $O(N^2)$, our diagonal block formulation reduces it to $O(N \cdot B_{TC})$, which is linear in N with a constant block size of $B_{TC} = 16$. By offloading the transformation to Tensor Cores (which offer 8× the throughput of standard CUDA Cores), the framework ensures the computational overhead remains virtually imperceptible.

4. Experiments

4.1. Experimental settings

All experiments were performed on the N32EA14P partition of the Beijing High-Performance Computing Center. The characteristics of the compute nodes are presented in Table 2. Hyperthreading was not enabled on the N32EA14P partition. For a given number of nodes, the performance test was run by placing an MPI process on each core.

Setup. The evaluation was conducted using the described test setup. Distributed training was performed with BytePS, configured with environment variables to optimize communication efficiency. All experiments were carried out on a single server equipped with eight NVIDIA A100-PCIe GPUs (each with 40 GB of memory). The system features two Intel Xeon Platinum 6248R processors, each with 24 cores running at 3.0 GHz, and 384 GB of RAM. The GPUs communicate primarily via the PCIe 3.0 × 16 interface. The software environment

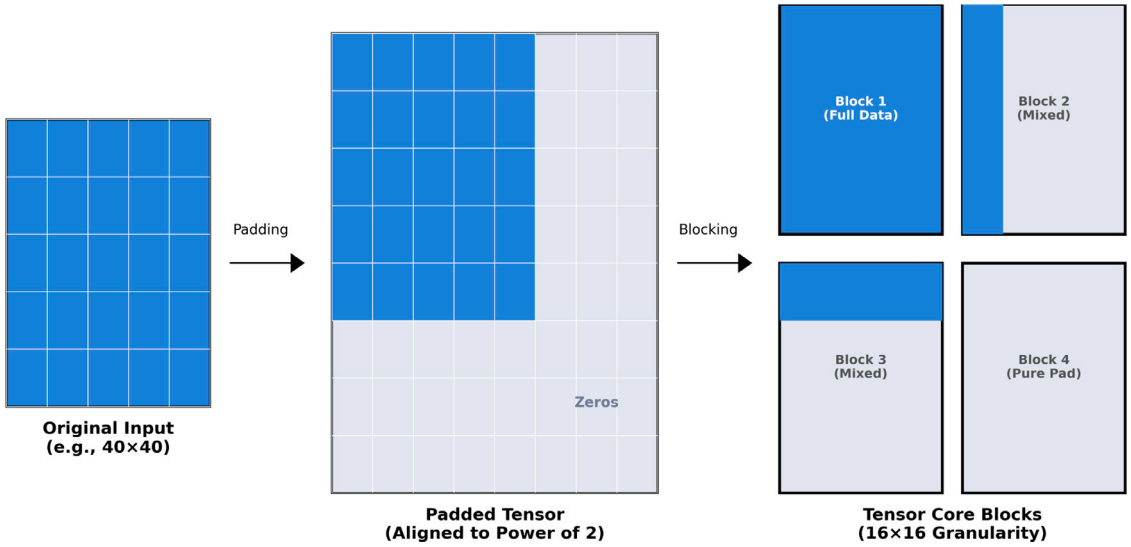


Fig. 4. Padding and blocking strategy. To satisfy the strict requirement for alignment of 16 × 16 Tensor cores (Section 3.4), HAGC pads the original gradient tensor (e.g., 40 × 40) with zeros to the nearest multiple of 16 (e.g., 64 × 64) before partitioning it into independent blocks for parallel execution.

Table 2
Single compute node.

Parameter	Specification
CPU model	Intel Xeon Platinum 6248R, 3.0 GHz
µarch	Cascade Lake
Populated sockets	2
Physical core count	48
Hyper-Threading	Disabled
RAM capacity	384 GB
GPU model	8× NVIDIA Tesla A100 (PCIe)
GPU memory	40 GB per GPU (Total: 320 GB)
Nominal FP32 peak	19.5 TFLOP/s per GPU (156 TFLOP/s total)
STREAM/copy	1555 GB/s (GPU memory bandwidth)

runs on CentOS, with a stack comprising PyTorch 1.10.1, CUDA 11.3, cuDNN 8.6.0, and NCCL 2.11.4.

Workloads. The impact of three optimization techniques – dynamic quantization (DynQuant), Hadamard transform (Hadamard), and Tensor Cores acceleration – is evaluated across multiple vision models, including ViT-Small [41], EfficientNet-B0 (Eff-B0) [42], ResNet-18 and ResNet-50 [43], as well as VGG19 and VGG11 [44]. Experiments are conducted using CIFAR-10 [45] (for ViT-Small, ResNet-18, and VGG19), MNIST [46], and ImageNet-Tiny (a curated 100-class subset of ImageNet-1 K [47], for Eff-B0). These models and datasets cover diverse architectural paradigms and complexity levels, enabling a comprehensive evaluation of optimization trade-offs in resource-constrained environments.

System Integration. The framework is implemented as a hierarchical extension to PyTorch. During the backward pass, gradient tensors are intercepted via communication hooks. These hooks invoke our custom CUDA kernels (via `pybind11`) to perform HAGC compression directly on the GPU. The compressed streams are then handed over to BytePS for efficient distributed synchronization. To facilitate reproducibility and adoption, the source code is publicly available at: https://github.com/airayangbovo1/HAGC-MAIN.

Experimental Configuration. Four configurations of optimization techniques are evaluated:

- **A**: Baseline (no optimizations)
- **B**: Dynamic quantization (8-bit adaptive precision)
- **C**: Dynamic quantization + Hadamard transform
- **D**: Full scheme (dynamic quantization + Hadamard transform + TensorCore acceleration)

using Top-1 accuracy and average training iteration latency (ms) as metrics, with hardware execution on an NVIDIA A100 GPU.

4.2. Performance evaluation

4.2.1. Convergence analysis with tensor core acceleration

To validate the stability of the proposed hardware-aware optimization, we analyze the training convergence behavior against a full-precision (FP32) benchmark. Fig. 5 show the training loss curves for ResNet-18 and VGG-19.

The HAGC framework demonstrates a convergence trajectory that closely mirrors the FP32 baseline. Specifically, for VGG-19, HAGC achieves a final loss of 0.1051, exhibiting a negligible deviation of less than 0.2% from the baseline's 0.1048. As indicated by the error bars (representing standard deviation across multiple runs), this performance is statistically stable. This robustness is primarily attributed to the integrated error feedback mechanism, which actively compensates for accumulated quantization noise, ensuring high model fidelity without the accuracy degradation typical of aggressive compression schemes.

The performance in terms of time-to-accuracy, illustrated in Fig. 6, highlights a significant improvement in training efficiency. This acceleration is mainly attributed to the strategic transfer of the Hadamard transform and quantization workloads to specialized Tensor Cores, which offloads compression tasks from general-purpose units. Additionally, the included error bars indicate the variance across independent experimental runs, confirming that the observed reductions in training time are statistically stable.

A comparative analysis reveals that VGG-19 benefits significantly more from our framework than ResNet 18. Empirical results show that VGG-19 achieves a substantial 3.15× speedup, whereas ResNet-18 demonstrates a 1.82× improvement. This trend is explained by the fundamental difference in their Communication-to-Computation Ratio (CCR). Since the HAGC framework is designed to alleviate communication bottlenecks, the communication-intensive VGG-19 experiences the most substantial performance gain. In contrast, ResNet-18 is more computation-bound. While it still achieves a notable speedup by reducing communication-induced idle time, the relative improvement is less pronounced than that of VGG-19.

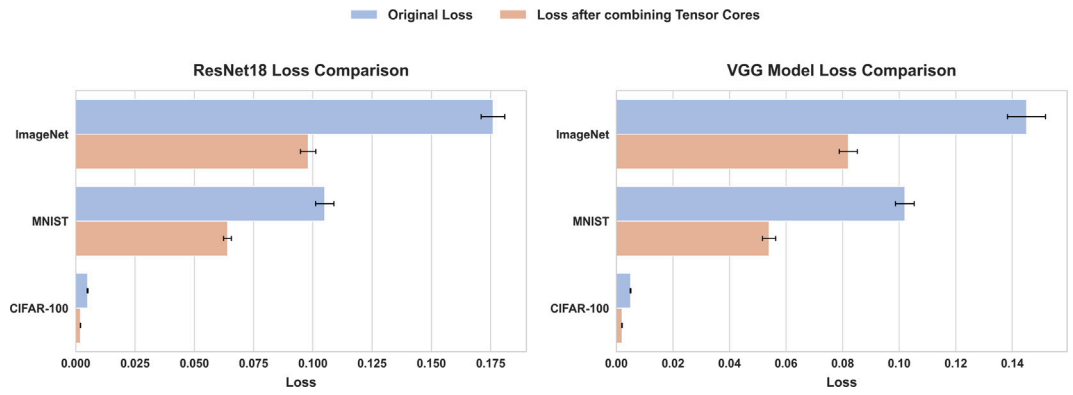


Fig. 5. Training loss convergence analysis.

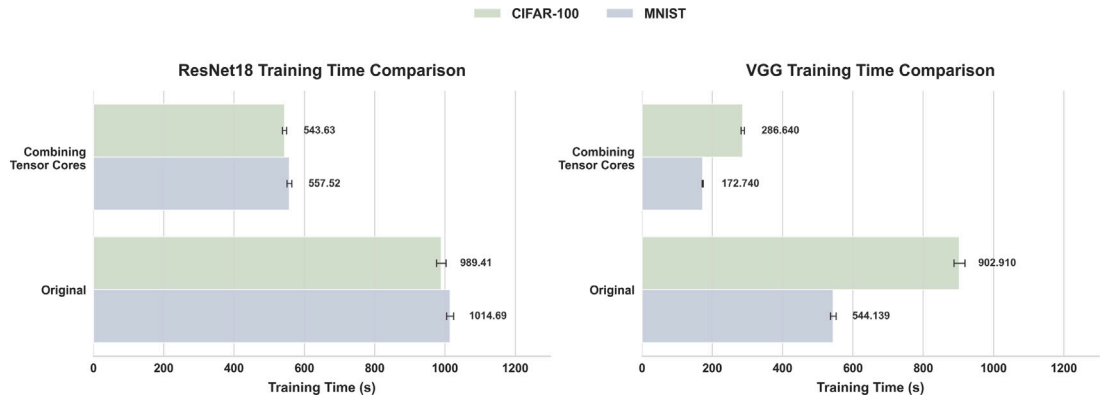


Fig. 6. Time-to-Accuracy performance comparison.

4.2.2. Hyperparameter sensitivity: Batch size analysis

To determine the optimal hardware utilization of the NVIDIA A100 GPU, a sensitivity analysis was performed across batch sizes 32–256. As presented in Fig. 7, the results reveal a clear trade-off between system throughput and model convergence. Based on these empirical observations, Batch Size 128 is identified as the robust operating point that reconciles hardware efficiency with convergence stability across diverse datasets.

Specific constraints dictate this dynamic, as smaller batch sizes (<64) fail to fully exploit the A100 processor's enormous computing power, resulting in suboptimal Tensor Core saturation. Conversely, while robust datasets like CIFAR-100 tolerate larger batches, the analysis reveals dataset-dependent generalization gaps on tasks like MNIST and ImageNet-Tiny, where accuracy degrades at batch sizes > 256. Consequently, Batch Size 128 emerges as the unified setting for all subsequent comparative experiments, ensuring high hardware occupancy while mitigating the risk of overfitting on sensitive benchmarks.

4.2.3. Comparative efficiency and architecture analysis

In addition to initial validation against standard FP32 implementations, the study includes a detailed comparative analysis of our method's performance against current industry-leading benchmarks. The following sections describe this comparative evaluation process with PowerSGD and QSGD and examine different neural network architectures.

1. Comparison with FP32 Baseline:

As shown in Fig. 8, the proposed method achieves substantial compression of the effective gradient size. With VGG-19, the data transfer rate is reduced by 67.9% (from 140 Mbps to 45 Mbps). With ViT-Small, a 50% reduction is achieved. This is directly visible in the latency increase illustrated in Fig. 14(b), where the training iteration latency of ViT-Small decreases from 10.5 ms to 3.5 ms (a threefold increase in speed) (see Fig. 9).

2. Comparison with State-of-the-Art (SOTA):

To comprehensively evaluate the performance of the proposed framework, a comparative analysis is conducted against three common benchmarks: FP32 (baseline), QSGD (4-bit quantization), and PowerSGD (low-level rank-4 approximation).

End-to-End Training Speedup. As shown in Fig. 10, HAGC consistently outperforms PowerSGD in training performance, achieving a 3.15× speedup on the data-intensive VGG-19 architecture. This performance is significantly higher than PowerSGD's 2.20×, whose reliance on SVD computation on high-dimensional gradients introduces a latency bottleneck that HAGC's hardware-optimized approach avoids. Even on the computationally demanding ResNet-50 architecture, HAGC maintains a 1.82× gain – well above QSGD's 1.15× improvement – demonstrating that the framework overcomes the "negative optimization" trap, where local compression costs negate the benefits of hardware optimization.

Overhead Analysis and Convergence. The latency distribution of VGG-19 illustrated in Fig. 11 highlights the source of HAGC's efficiency: while QSGD and PowerSGD are penalized by CPU-intensive compression or the need for intensive factorization, HAGC reduces this overhead to a negligible 5 ms. By offloading Hadamard transformations and quantization to the Tensor Cores, the framework "masks" these costs within the gradient calculation itself. This speed gain corresponds to the model's exact accuracy. As shown in Fig. 12, HAGC follows

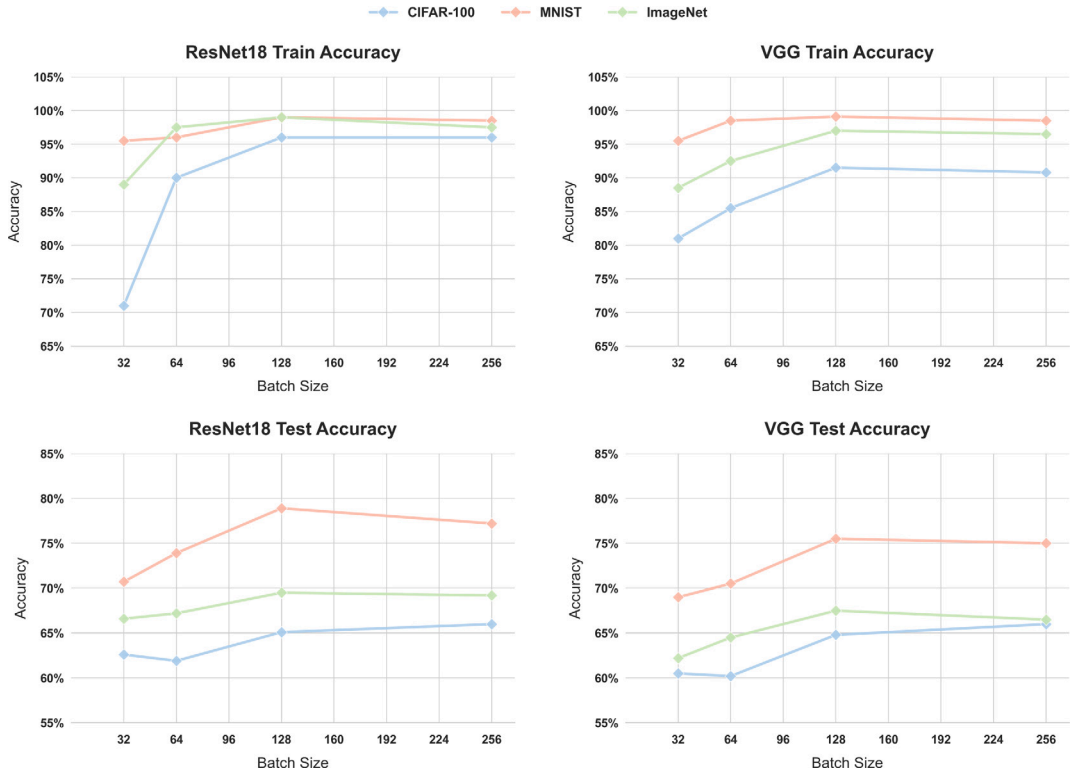


Fig. 7. Accuracy evaluation curves. Comparison of training and testing accuracy on ResNet-18 (Top row) and VGG (Bottom row).

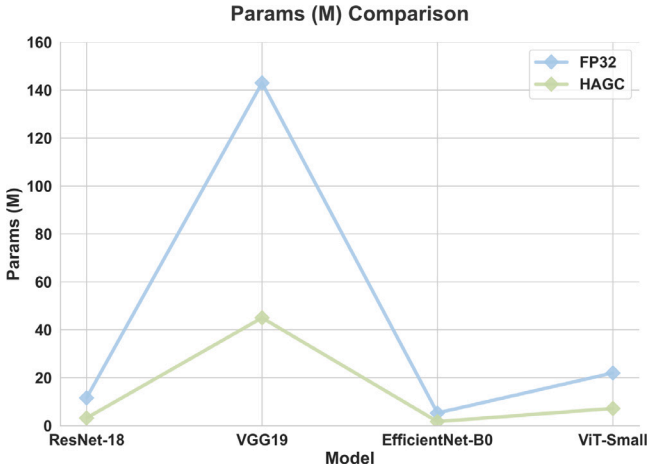


Fig. 8. Effective gradient size reduction.

the same convergence trajectory as PowerSGD, but reaches the target accuracy thresholds much faster. The final accuracy of 76.1% (top-1), almost identical to that of the FP32 benchmark and slightly higher than that of PowerSGD, confirms that the hardware-aware optimization enables faster training without sacrificing accuracy.

Bandwidth Sensitivity Analysis. To evaluate the framework's robustness under varying network conditions, we conducted a sensitivity analysis by enforcing software-level communication throttles on the 8-GPU node. We emulated restricted bandwidth environments ranging from 1 Gbps to 100 Gbps. This controlled setup isolates the impact of gradient compression on transmission time, effectively simulating the bandwidth constraints of distributed clusters – such as cloud services without RDMA support – without the confounding variables of network jitter or physical packet loss.

The results shown in Fig. 13 highlight HAGC's resilience to communication bottlenecks, particularly in the most constrained environments. At 1 Gbps, HAGC achieves 8.5× acceleration, significantly surpassing QSGD's 4.5× improvement. This confirms that as bandwidth decreases, the marginal benefit of our hardware-aware compression grows substantially.

3. Architecture-Dependent Performance Analysis:

An important observation from the Fig. 14 is that the performance gains vary from one model to another. This variation is due to the communication-to-computing ratio (CCR):

- High CCR: The only constraint on training with fully connected layers is network bandwidth, so an aggressive compression approach offers the greatest marginal advantage (a 3× speedup).
- Low CCR: Traditional software compression often results in "negative optimization" in this case. Because HAGC uses underutilized Tensor cores for compression, a net speed gain is still achieved, even in these computationally demanding models.

4. Resource Consumption Breakdown:

Figs. 15, 16, and 17 provide a comprehensive overview of resource efficiency.

- Energy: Fig. 16 shows a reduction in energy consumption of 2.97 to 3.61 times. This is crucial for green AI, especially for computationally intensive models like VGG.
- Memory: Fig. 17 shows a reduction in memory usage from 3.46 to 3.85 times. These memory savings enable larger batch sizes, making better use of the A100's capacity.

HAGC demonstrates its ability to simplify even the most complex, parameterized models by combining rigorous accuracy requirements with an efficient architecture. The system keeps accuracy loss

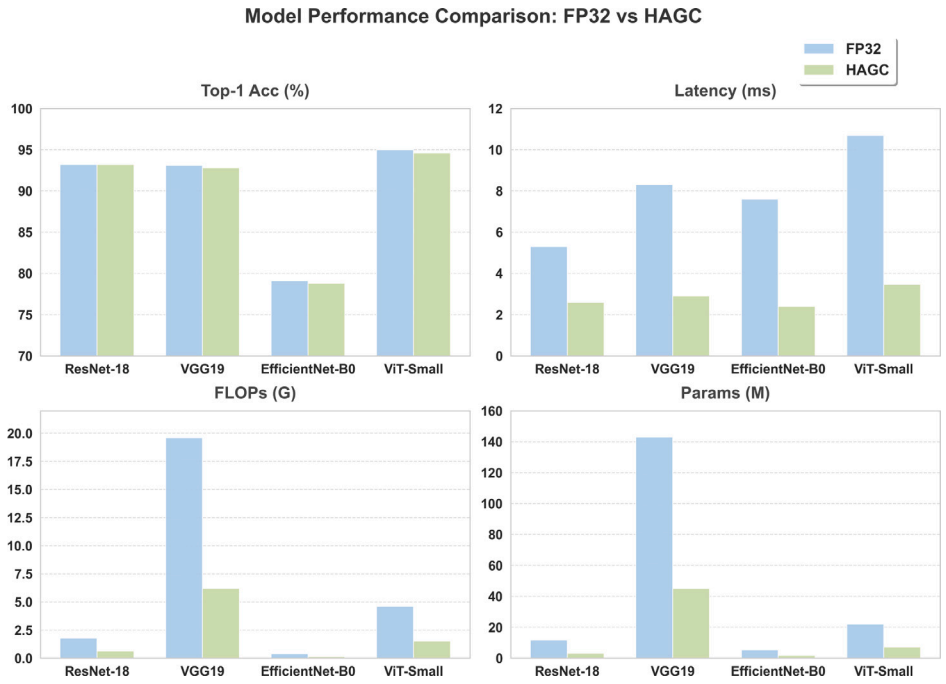


Fig. 9. Performance comparison: accuracy, latency, FLOPs, and parameters comparison between FP32 and HAGC.

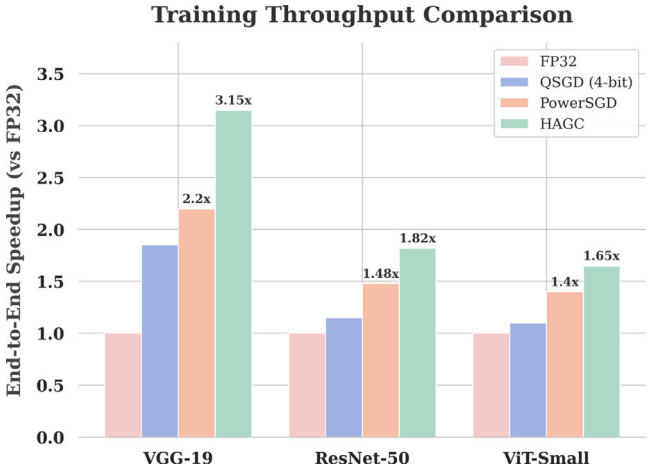


Fig. 10. End-to-End training speedup comparison.

below 0.5%, doubles training speed, and reduces power consumption. Furthermore, it effectively combines seemingly disparate domains through high-level compression algorithms and low-level hardware performance.

5. Related work

Quantization reduces communication bottlenecks by decreasing gradient accuracy, a concept typical of the ternary application TernGrad ($x \in \{-1, 0, 1\}$). While static quantization effectively reduces data size, it often introduces errors that degrade accuracy. Dynamic strategies address this issue by adapting bit widths to specific gradient distributions. The efficiency of these methods largely depends on their compatibility with hardware primitives. Specialized accelerators, such as NVIDIA Tensor Cores, facilitate high-performance, low-precision operations, and frameworks like QNNPACK and TVM demonstrate that hardware-specific quantization significantly accelerates computational throughput and execution efficiency.

Traditional techniques, such as sparsity and network aggregation, reduce communication overhead but often result in a loss of accuracy or a local computational overhead. HAGC addresses these systemic challenges by optimizing dynamic quantization for efficient devices like Tensor Cores. By balancing the compaction efficiency model with the accuracy framework, hardware-adapted compaction guarantees a net gain. This integration ultimately resolves the friction between algorithmic mathematics and physical performance, thus optimizing distributed training in various deep learning environments.

6. Conclusion

This paper presents HAGC, a hardware-aware gradient compression framework that redefines gradient quantization by reformulating the recursive Hadamard transform into a series of dense matrix operations. By co-designing the algorithm with the underlying microarchitecture, we effectively bypass the memory-bound latencies inherent in traditional software-based methods and shift the workload to high-throughput NVIDIA Tensor Cores.

Empirical evaluations demonstrate the framework's effectiveness and versatility. HAGC achieves a significant training speedup of 3.15× for communication-intensive architectures (e.g., VGG-19) and 1.82× for computation-intensive models (e.g., ResNet-18). This variance highlights that models with lower arithmetic intensity (i.e., higher communication-to-computation ratio) benefit more substantially from our pipeline, as the gradient compression effectively mitigates the dominant communication bottlenecks. Furthermore, the framework fosters "Green AI" by reducing overall energy consumption, while maintaining convergence stability comparable to full-precision baselines (with loss deviation $< 0.2\%$).

However, we acknowledge certain boundary conditions in our current study. Performance Boundaries: HAGC is optimized for high-throughput scenarios. Efficiency is maximized with sufficient Tensor Core occupancy, whereas gains may be modest under extremely small batch sizes (low saturation); Experimental Scope: Our evaluation used

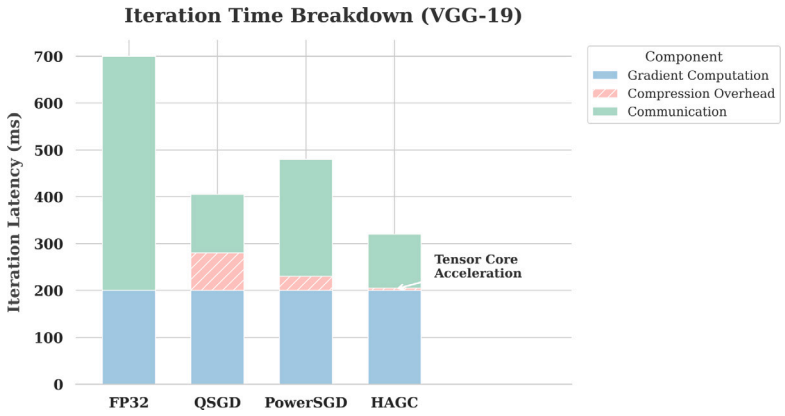


Fig. 11. Iteration time breakdown on VGG-19.

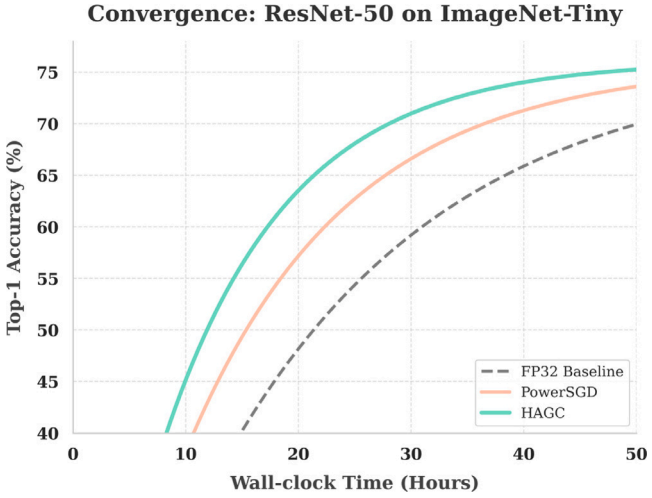


Fig. 12. Top-1 accuracy vs. wall-clock time.

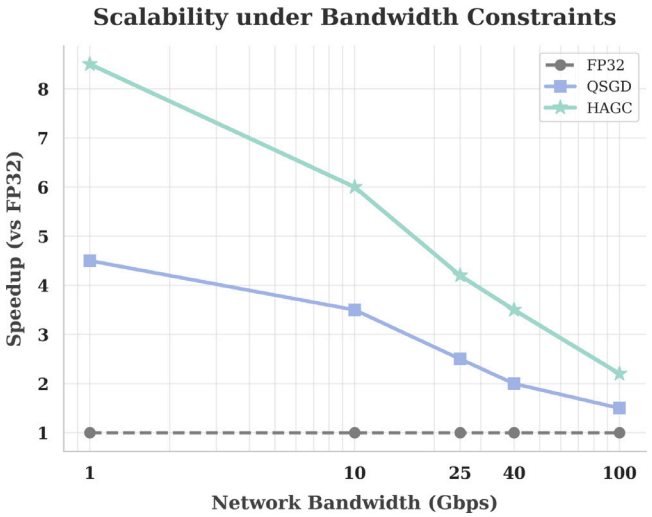


Fig. 13. Training speedup vs. emulated network bandwidth.

bandwidth emulation on homogeneous clusters to rigorously isolate algorithmic efficiency. While effective for controlled validation, future work will extend this to physical distributed environments to assess performance under complex network dynamics (e.g., jitter) or hardware heterogeneity; Hardware Specificity: The framework currently leverages NVIDIA Tensor Cores to prioritize maximum acceleration, limiting immediate deployability on non-Tensor Core hardware.

Looking forward, we plan to address these limitations by exploring adaptive compression strategies driven by reinforcement learning. This integration will enable the system to dynamically manage the trade-offs between network bandwidth and training accuracy, specifically optimizing for heterogeneous cluster environments and varying hardware capabilities.

CRediT authorship contribution statement

Aiqiang Yang: Software, Methodology, Conceptualization. **Jie Liu:** Data curation. **Bo Yang:** Formal analysis. **Xiang Zhang:** Supervision. **Qinglin Wang:** Validation. **Zeyao Mo:** Writing – review & editing. **Keqin Li:** Writing – review & editing.

Data and code availability

The source code of the HAGC framework is openly available at: https://github.com/airayangbovo1/HAGC-MAIN.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used Gemini 3 Pro in order to improve language and readability. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the published article.

Funding

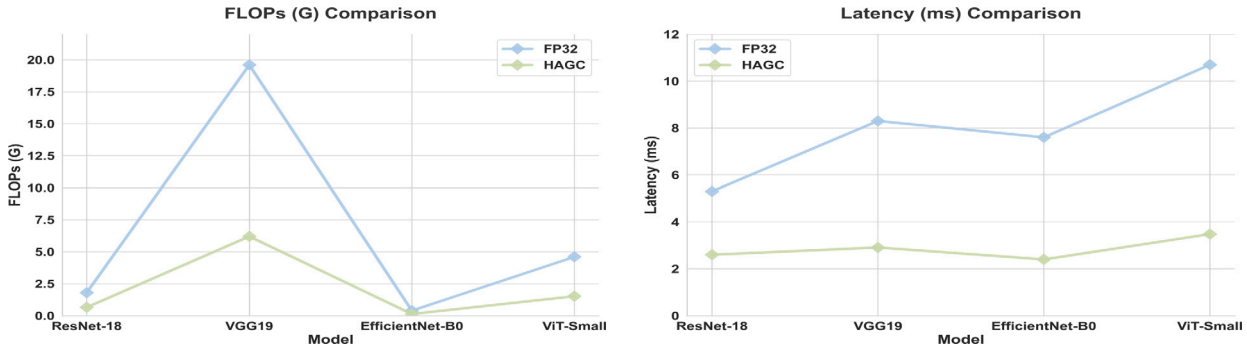
The research was funded by the National Key Research and Development Program of China (CO_2-EOR and Storage Simulation) [2023YFA1011704].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

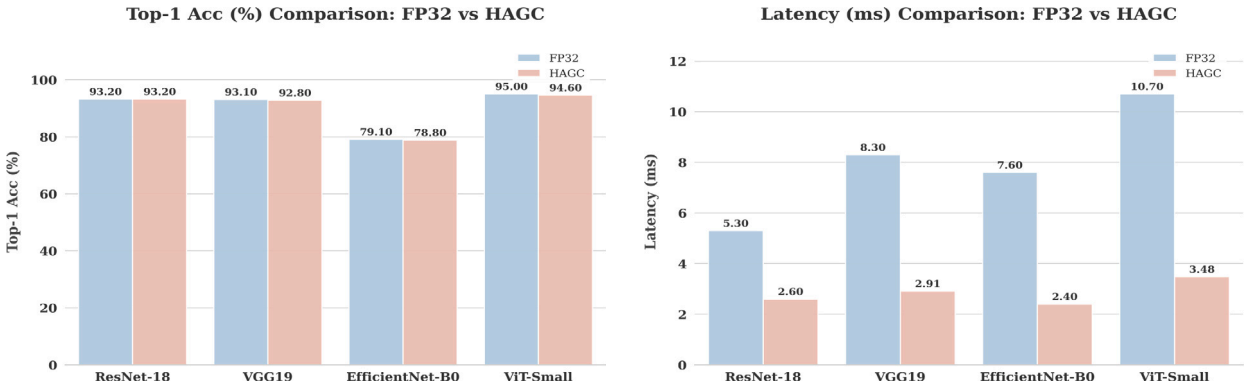
Data availability

The source code is publicly available at: https://github.com/airayangbovo1/HAGC-MAIN.



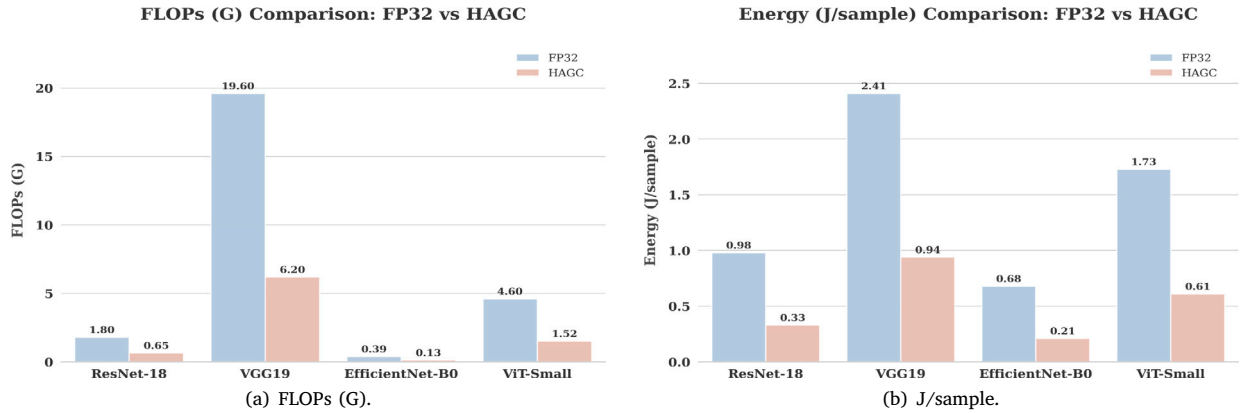
(a) Computational efficiency (FLOPs). (b) Latency speedup comparison.

Fig. 14. Efficiency analysis. Comparison of (a) Computational cost (FLOPs) and (b) Training Iteration Latency across different models.



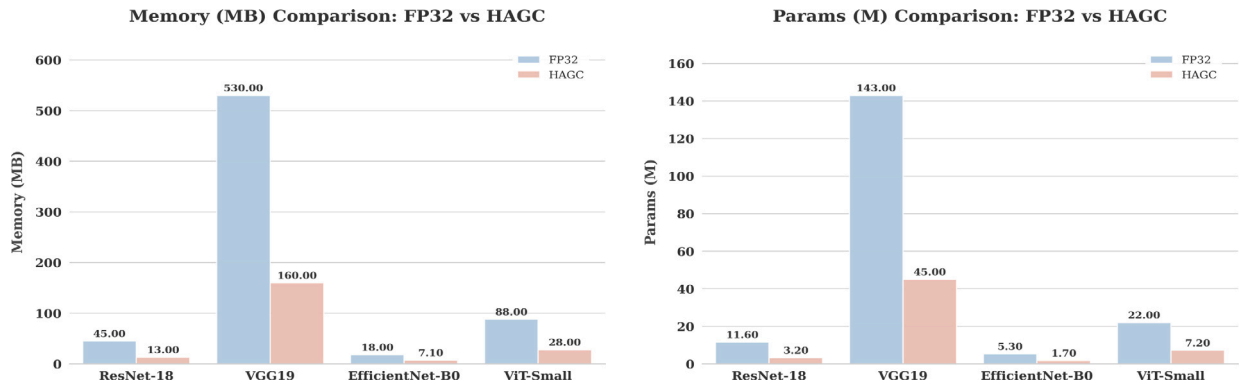
(a) Top-1 accuracy. (b) Latency (ms).

Fig. 15. Trade-off Analysis. Comparison between (a) Accuracy retention and (b) Latency performance.



(a) FLOPs (G). (b) J/sample.

Fig. 16. Resource consumption analysis. Comparison of computational cost (FLOPs) and energy consumption per sample.



(a) Memory footprint. (b) Parameter count.

Fig. 17. Resource overhead analysis. Comparison of model memory footprint and parameter count.

References

[1] A. M Abdelmoniem, A. Elzanaty, M.-S. Alouini, M. Canini, An efficient statistical-based gradient compression technique for distributed training systems, Proc. Mach. Learn. Syst. 3 (2021) 297–322.

[2] Y. Lin, S. Han, H. Mao, Y. Wang, W.J. Dally, Deep gradient compression: Reducing the communication bandwidth for distributed training, 2017, arXiv preprint arXiv:1712.01887.

[3] T. Vogels, S.P. Karimireddy, M. Jaggi, PowerSGD: Practical low-rank gradient compression for distributed optimization, Adv. Neural Inf. Process. Syst. 32 (2019).

[4] D. Alistarh, D. Grubic, J. Li, R. Tomioka, M. Vojnovic, QSGD: Communication-efficient SGD via gradient quantization and encoding, Adv. Neural Inf. Process. Syst. 30 (2017).

[5] P. Talli, F. Pase, F. Chiariotti, A. Zanella, M. Zorzi, Effective communication with dynamic feature compression, IEEE Trans. Commun. (2024).

[6] A. Beznosikov, S. Horváth, P. Richtárik, M. Safaryan, On biased compression for distributed learning, J. Mach. Learn. Res. 24 (276) (2023) 1–50.

[7] M. Li, R.B. Basat, S. Vargaftik, C. Lao, K. Xu, M. Mitzenmacher, M. Yu, {THC}: Accelerating distributed deep learning using tensor homomorphic compression, in: 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), 2024, pp. 1191–1211.

[8] K. Huang, B. Ni, X. Yang, Efficient quantization for neural networks with binary weights and low bitwidth activations, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 3854–3861.

[9] R. Krishnamoorthi, Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018, arXiv preprint arXiv:1806.08342.

[10] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, C. Guo, A generic communication scheduler for distributed DNN training acceleration, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 16–29.

[11] H. Kal, H. Choi, I. Jeong, J.-S. Yang, W.W. Ro, A convertible neural processor supporting adaptive quantization for real-time neural networks, J. Syst. Archit. 145 (2023) 103025.

[12] R. Hönig, Y. Zhao, R. Mullins, DAdaQuant: Doubly-adaptive quantization for communication-efficient federated learning, in: International Conference on Machine Learning, PMLR, 2022, pp. 8852–8866.

[13] D. Liu, A. Lamb, X. Ji, P. Notsawo, M. Mozer, Y. Bengio, K. Kawaguchi, Adaptive discrete communication bottlenecks with dynamic vector quantization, 2022, arXiv preprint arXiv:2202.01334.

[14] Z. Wang, M. Wen, Y. Xu, Y. Zhou, J.H. Wang, L. Zhang, Communication compression techniques in distributed deep learning: A survey, J. Syst. Archit. 142 (2023) 102927.

[15] S. Ahmadunnisa, S.E. Mathe, A comprehensive review on hardware implementations of lattice-based cryptographic schemes, J. Syst. Archit. (2025) 103486.

[16] Y. Ren, Y. Cao, C. Ye, X. Cheng, Two-layer accumulated quantized compression for communication-efficient federated learning: TLAQC, Sci. Rep. 13 (1) (2023) 11658.

[17] Z. Ren, L. Wang, D. Yuan, H. Su, G. Shi, Spatio-temporal communication compression in distributed prime-dual flows, 2024, arXiv preprint arXiv:2408.02332.

[18] X. Li, B. Karimi, P. Li, On distributed adaptive optimization with gradient compression, 2022, arXiv preprint arXiv:2205.05632.

[19] A. Xu, Z. Huo, H. Huang, Step-ahead error feedback for distributed training with compressed gradient, in: Proceedings of the AAAI Conference on Artificial Intelligence, 35, (12) 2021, pp. 10478–10486.

[20] S.P. Karimireddy, Q. Rebjock, S. Stich, M. Jaggi, Error feedback fixes signsgd and other gradient compression schemes, in: International Conference on Machine Learning, PMLR, 2019, pp. 3252–3261.

[21] S.K. Ankireddy, S.A. Hebbar, Y. Jiang, P. Viswanath, H. Kim, Compressed error harq: Feedback communication on noise-asymmetric channels, in: 2023 IEEE International Symposium on Information Theory, ISIT, IEEE, 2023, pp. 1160–1165.

[22] C.W. Ling, C.T. Li, Vector quantization with error uniformly distributed over an arbitrary set, IEEE Trans. Inform. Theory (2024).

[23] T.T. Phuong, et al., Differentially private stochastic gradient descent via compression and memorization, J. Syst. Archit. 135 (2023) 102819.

[24] A. Gondimalla, M. Thottethodi, T. Vijaykumar, Eureka: Efficient tensor cores for one-sided unstructured sparsity in DNN inference, in: Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, 2023, pp. 324–337.

[25] H. Feng, B. Zhang, F. Ye, M. Si, C.-H. Chu, J. Tian, C. Yin, S. Deng, Y. Hao, P. Balaji, et al., Accelerating communication in deep learning recommendation model training with dual-level adaptive lossy compression, in: SC24: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2024, pp. 1–16.

[26] A. Li, S. Su, Accelerating binarized neural networks via bit-tensor-cores in turing gpus, IEEE Trans. Parallel Distrib. Syst. 32 (7) (2020) 1878–1891.

[27] W. Sun, S. Sioutas, S. Stuijk, A. Nelson, H. Corporaal, Efficient tensor cores support in tvm for low-latency deep learning, in: 2021 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2021, pp. 120–123.

[28] S. Markidis, S.W. Der Chien, E. Laure, I.B. Peng, J.S. Vetter, Nvidia tensor core programmability, performance & precision, in: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, IEEE, 2018, pp. 522–531.

[29] B. Li, S. Cheng, J. Lin, Tcfft: Accelerating half-precision FFT through tensor cores, 2021, arXiv preprint arXiv:2104.11471.

[30] Y. Liu, Y. Xue, Y. Cheng, L. Ma, Z. Miao, J. Xue, J. Huang, Scaling deep learning computation over the inter-core connected intelligence processor with T10, in: Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, 2024, pp. 505–521.

[31] C.A. Navarro, R. Carrasco, R.J. Barrientos, J.A. Riquelme, R. Vega, GPU tensor cores for fast arithmetic reductions, IEEE Trans. Parallel Distrib. Syst. 32 (1) (2020) 72–84.

[32] A. Dakkak, C. Li, J. Xiong, I. Gelado, W.-m. Hwu, Accelerating reduction and scan using tensor core units, in: Proceedings of the ACM International Conference on Supercomputing, 2019, pp. 46–57.

[33] F.A. Quezada, C.A. Navarro, N. Hitschfeld, B. Bustos, Squeeze: Efficient compact fractals for tensor core gpus, Future Gener. Comput. Syst. 135 (2022) 10–19.

[34] Z. Zhang, Z. Fan, W. Li, Y. Qiu, Z. Wang, X. Ye, D. Fan, X. An, Accelerating tensor multiplication by exploring hybrid product with hardware and software co-design, J. Syst. Archit. 159 (2025) 103333.

[35] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, et al., Fp8 formats for deep learning, 2022, arXiv preprint arXiv:2209.05433.

[36] H. Peng, K. Wu, Y. Wei, G. Zhao, Y. Yang, Z. Liu, Y. Xiong, Z. Yang, B. Ni, J. Hu, et al., Fp8-lm: Training fp8 large language models, 2023, arXiv preprint arXiv:2310.18313.

[37] L. Zhang, L. Zhang, S. Shi, X. Chu, B. Li, Evaluation and optimization of gradient compression for distributed deep learning, in: 2023 IEEE 43rd International Conference on Distributed Computing Systems, ICDCS, IEEE, 2023, pp. 361–371.

[38] X.-Y. Liu, T. Zhang, H. Hong, H. Huang, H. Lu, High-performance computing primitives for tensor networks learning operations on GPUs, in: Proc. Int. Conf. Neural Inf. Process. Syst. Workshop Quantum Tensor Netw. Mach. Learn, 2020, pp. 1–9.
[39] J. Feldmann, N. Youngblood, M. Karpov, H. Gehring, X. Li, M. Stappers, M. Le Gallo, X. Fu, A. Lukashchuk, A.S. Raja, et al., Parallel convolutional processing using an integrated photonic tensor core, Nature 589 (7840) (2021) 52–58.
[40] S.U. Stich, J.-B. Cordonnier, M. Jaggi, Sparsified SGD with memory, Adv. Neural Inf. Process. Syst. 31 (2018).
[41] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., An image is worth 16x16 words: Transformers for image recognition at scale, 2020, arXiv preprint arXiv:2010.11929.
[42] M. Tan, Q. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6105–6114.
[43] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[44] K. Simonyan, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[45] A. Krizhevsky, G. Hinton, et al., Learning Multiple Layers of Features from Tiny Images, Technical Report,, University of Toronto, 2009.
[46] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.

Aiqiang Yang born in 2001, is currently pursuing the Ph.D. degree with the National University of Defense Technology. Her research interests include communication compression, distributed training, and parallel optimization.

Jie Liu received the Ph.D. degree s in computer science from the National University of Defense Technology (NUDT), Changsha, China. He is a professor with the College of Computer Science and Technology, National University of Defense Technology. His research interests include high performance computing and machine learning.

Bo Yang born in 1987, Ph.D., is currently a Researcher with the National University of Defense Technology. His research interests include communication optimization and high-performance computing.

Xiang Zhang Ph.D., is currently a Researcher with the National University of Defense Technology. His research interests include communication optimization, high-performance computing, and deep learning.

Qinglin Wang born in 1987. He received the B.S. degree in mechanical engineering from Tsinghua University, China, in 2009, and the M.S. and Ph.D. degrees in electrical science and Technology from the National University of Defense Technology, China, in 2016. He has been a Research Assistant with the Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, since 2016. His research interests include high-performance computing, VLSI signal processing, and machine learning.

Zeyao Mo received his B.S. and Ph.D. from the National University of Defense Technology. He currently serves as the Party Committee Secretary of the China Academy of Engineering Physics. His research focuses on parallel computing and high-performance scientific computing, with contributions in large-scale parallel algorithms, parallel programming frameworks, and high-performance CAE software. He has authored several books on parallel computing.

Keqin Li (Fellow, IEEE, AAAS, AAIA, ACIS, AIIA, Academia Europaea) is a SUNY Distinguished Professor of Computer Science with the State University of New York and a National Distinguished Professor with Hunan University, China. He received the B.S. degree from Tsinghua University in 1985 and the Ph.D. degree from the University of Houston in 1990.

He has authored/co-authored 1130+ refereed publications and holds 80 patents. Ranked #2 worldwide (2020) in single-year impact and #4 in career-long impact in parallel/distributed computing (Scopus). Top 0.02% (Scilit) and top 0.002% (ScholarGPS) scholars globally. Awards include IEEE TCCLD Research Impact Award (2022), IEEE TCSVC Research Innovation Award (2023), etc.

Current research interests:

- Cloud/fog/mobile edge computing
- Energy-efficient computing
- Embedded & cyber–physical systems
- Heterogeneous computing
- Big Data & high-performance computing
- CPU–GPU hybrid computing
- Machine learning & AI