Delta-MoE: Rethinking Transfer Granularity for CPU–GPU MoE Serving

Jiangnan Yu*
jyuc@connect.ust.hk
The Hong Kong University of Science and Technology
Hong Kong, China

Kunming Shao
kshaoaa@connect.ust.hk
The Hong Kong University of Science and Technology
Hong Kong, China

Shiwei Liu†
shiweiliu@tongji.edu.cn
Fudan University
Shanghai, China

Chixiao Chen
cxchen@fudan.edu.cn
Fudan University
Shanghai, China

Yuan Xie†
yuanxie@ust.hk
The Hong Kong University of Science and Technology
Hong Kong, China

ABSTRACT

Serving large Mixture-of-Experts (MoE) models over a CPU–GPU system is often bottlenecked by PCIe expert transfers. Existing approaches reduce this bottleneck by fetching earlier, placing experts better, or compressing expert weights, but after routing they still transfer an activated expert, or a compressed version of one. Our key insight is that the real design lever is the post-routing transfer unit itself: experts share structures that the system can keep the shared portion in GPU HBM and fetch only the expert-specific difference. This motivates our central question: can offloaded MoE serving replace full-expert transfers with expert-specific deltas without sacrificing accuracy or GPU efficiency?

We introduce Delta-MoE, a CPU–GPU MoE serving design that factors each expert around a shared E_{base} and an expert-specific ΔE_i. After routing, the runtime launches the E_{base} GEMM immediately while ΔE_i is transferred over PCIe, so only the transfer time not hidden by that overlap, plus the later ΔE_i GEMM, remains on the critical path. To make this new transfer unit practical, we introduce SAME, a Sign-Aware, Memory-Aligned encoding that suppresses outlier amplification and packs ΔE_i into a dense 4-bit GPU-friendly stream, and S-MMA, the corresponding decode-and-accumulate datapath implemented as a fused CUDA kernel on commercial H20 GPUs. Across Mixtral-8×7B, Qwen1.5-MoE-A2.7B, and Phi-3.5-MoE, Delta-MoE preserves near-BF16 accuracy, improves end-to-end throughput by up to 3.2×, and reduces PCIe transfer latency by 2.3× on the H20 GPU. These results show that changing the post-routing transfer unit is an effective design point for commodity CPU–GPU MoE serving.

1 INTRODUCTION

Large Mixture-of-Experts (MoE) models often have to be served in a CPU–GPU offloading regime. Sparse activation reduces computation, but it does not remove the need to store and fetch the experts that routing may select [1, 2]. As shown in Fig. 1(a), a gating network routes each token to the top-k experts in a transformer block, and the resulting expert pool is often too large to keep entirely in GPU HBM.

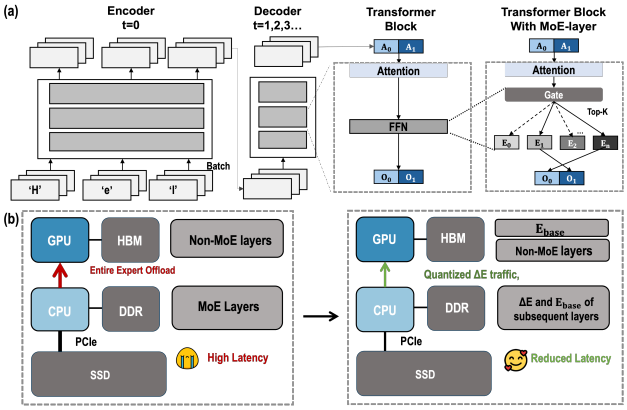

Figure 1: MoE structure and CPU–GPU expert offloading.

When the expert pool exceeds HBM capacity, practical deployments offload experts to host memory. In Mixtral-8×7B, a single expert within one MoE layer occupies about 0.35 GB in BF16, while the full expert pool across all MoE layers is about 90 GB, which cannot reside in GPU HBM together with non-expert weights and KV cache. As shown in Fig. 1(b) (left), dense non-expert parameters remain in GPU HBM while the full expert pool is stored in CPU memory, and the routed experts of the current MoE layer are fetched over PCIe on demand [3]. This makes PCIe transfer part of the critical path, not just a capacity workaround: execution proceeds layer by layer, and the GPU cannot continue until the selected experts for the current layer arrive across the interconnect. This bottleneck is substantial in practice. Under a 2K-token Mixtral-8×7B profile on an Intel Xeon Gold 6548Y CPU and an NVIDIA H20 GPU, PCIe transfer takes 11 ms for each routed expert of a layer while computation takes only 4 ms, so transfer accounts for 73.3% of total latency.

Prior work has reduced this bottleneck along four main axes: earlier fetch, better expert placement, fewer bytes per transfer, and different execution substrates. ExFlow [4] and related prediction-based methods such as Pre-gated MoE [5] issue expert fetches earlier, which lowers exposed latency when prediction is accurate, but the miss path still transfers full experts after routing. MoE-Infinity [6]

*Jiangnan Yu is the first author.
†Shiwei Liu and Yuan Xie are the corresponding authors.

and HybriMoE [7] improve cache placement and hybrid CPU–GPU scheduling, which is effective when expert reuse is high, but they do not reduce the size of each post-routing miss. Compression-oriented approaches simply shrink expert weights: GPTQ-style PTQ [8] and MoE-specific quantization such as EAQuant [9] reduce the bytes of each expert, but the post-routing path still moves a compressed expert rather than changing the transfer unit itself. MoNDE [10] goes further by moving cold-expert computation near memory, but that requires a different hardware substrate rather than a drop-in change to today's CPU–GPU offloading stack. Across these axes, the shared limitation is that, after routing, PCIe still carries an activated expert, or a compressed version of one.

We propose **Delta-MoE**, which changes the post-routing transfer unit in offloaded MoE serving. Instead of fetching an activated expert, the system keeps E_{base} in GPU HBM and fetches only an expert-specific delta ΔE_i from host memory, as shown in Fig. 1(b) (right). After routing, the runtime launches the E_{base} GEMM immediately while the host transfers ΔE_i, so only the transfer time not hidden by that overlap, plus the later ΔE_i GEMM, remains on the critical path. This differs from earlier fetch, better placement, or expert compression because Delta-MoE changes the post-routing object itself. Yet factorization alone is not enough: a dense BF16 ΔE_i is still a large transfer object, and naive subtraction can amplify outliers. We therefore introduce SAME, short for Sign-Aware, Memory-Aligned Encoding, which first constructs ΔE_i in a sign-aware exact reparameterization and then packs it into one dense fixed-width 4-bit layout. To avoid making SAME decode the next bottleneck, we further define S-MMA as the corresponding decode-and-accumulate datapath; the software implementation uses a fused CUDA kernel on an unmodified H20, while a hardware-support study estimates the benefit of embedding that datapath into tensor cores.

We make the following contributions.

- **Rethink post-routing transfer granularity.** We show that offloaded MoE serving remains bottlenecked by expert-granular CPU–GPU movement, and elevate post-routing transfer granularity to a first-class systems/codesign knob by replacing full-expert transfer with a streamed per-expert delta ΔE_i while keeping E_{base} resident in HBM.
- **Design SAME for a practical ΔE_i path.** SAME first suppresses outlier amplification through sign-aware ΔE_i construction and then packs normal values and outliers into one dense fixed-width 4-bit layout, avoiding fragmented side channels and irregular accesses.
- **Implement the SAME path and study hardware support.** We implement the SAME path with a fused CUDA kernel on an unmodified H20, and define S-MMA as the corresponding tensor-core extension for native decode-and-accumulate support; a hardware-support study shows that embedding it into tensor cores would add only 0.26% die area while further improving energy efficiency.
- **Demonstrate end-to-end gains in offloaded serving.** Across Mixtral-8×7B, Phi-3.5-MoE, and Qwen1.5-MoE-A2.7B, Delta-MoE preserves near-BF16 accuracy, achieves up to 3.2× end-to-end speedup, and reduces PCIe latency by 2.3× on an H20 GPU.

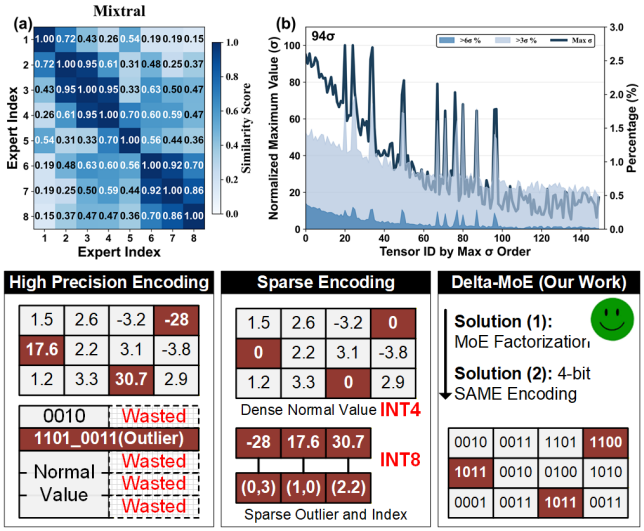

Figure 2: Mixtral-8×7B expert statistics. (a) Intra-layer cross-expert similarity, and (b) Magnitude statistics of ΔE_i.

2 BACKGROUND AND MOTIVATION

2.1 Expert-Granular Transfer on CPU–GPU Systems

Because conventional offloading must fetch routed experts after gating, Delta-MoE targets the post-routing transfer unit without changing the model or routing policy.

2.2 Expert Redundancy Suggests a Better Transfer Unit

The next question is whether a full expert is actually the right object to move. We observe substantial cross-expert redundancy in MoE models, which suggests that expert-sized movement is unnecessarily coarse. We adopt Centered Kernel Alignment (CKA) [11] to quantify similarity directly on the expert weight tensors of each routed MoE layer. As shown in Fig. 2(a), cross-expert similarity typically falls within 0.5–0.8 for Mixtral's 8 experts. Importantly, this redundancy is not limited to small expert pools: Qwen1.5-MoE-A2.7B, which employs 60 experts per layer, still exhibits CKA similarities of 0.3–0.5 across experts, showing that this shared structure persists even in finer-grained MoE architectures. Across the three MoE families evaluated later, the routed-layer average pairwise CKA is 0.64 for Mixtral-8×7B, 0.41 for Qwen1.5-MoE-A2.7B, and 0.53 for Phi-3.5-MoE; the corresponding ΔE_i norm ratios $\|\Delta E_i\|_F / \|E_i\|_F$ are 0.47, 0.58, and 0.51. These values indicate substantial but model-dependent shared structure: Qwen1.5-MoE-A2.7B is the weakest-sharing case among the three, while Mixtral exhibits the strongest cross-expert similarity. These results suggest a different transfer abstraction: instead of moving an entire expert, we can keep E_{base} on the GPU and move only the expert-specific delta,

$$E_i = E_{\text{base}} + \Delta E_i, \tag{1}$$

Here E_{base} is the HBM-resident tensor shared across experts in that layer, and ΔE_i captures what is specific to expert i. Under this decomposition, the post-routing object becomes ΔE_i instead

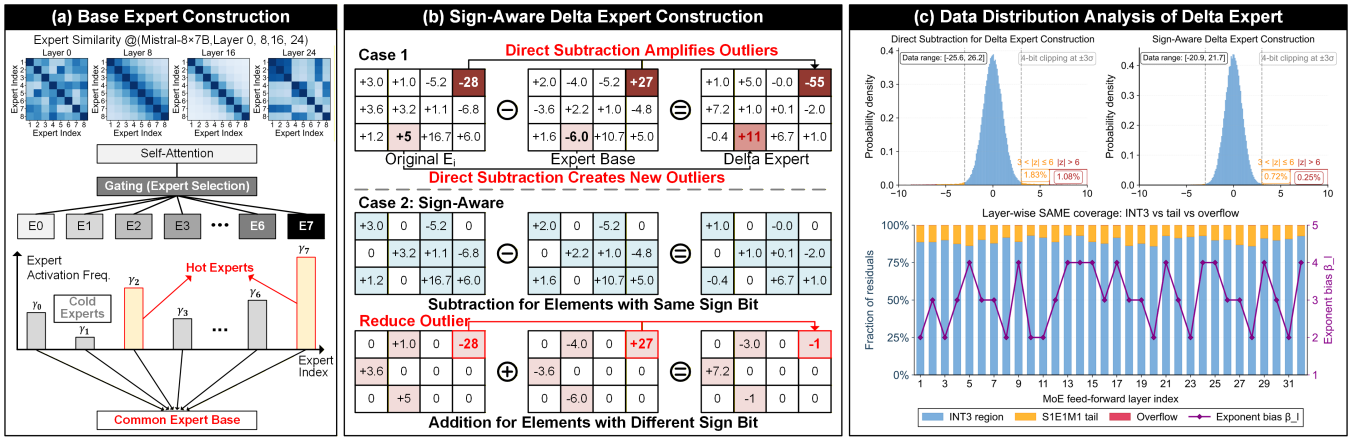

Figure 3: Delta-MoE design: (a) expert factorization, (b) sign-aware ΔE_i construction with one block-wise sign $S_B \in \{+1, -1\}$ per 16×16 block, and (c) memory-aligned ΔE_i encoding with one S1E4M3 per-block scale a_B for the INT3 normal path; outliers use the S1E1M1 path with layer-wise bias β_ℓ.

of E_i. This change is conceptual as much as numerical: Delta-MoE promotes base-plus-delta to the runtime decomposition, but only ΔE_i moves over PCIe while E_{base} remains resident in HBM. The factorization alone does not reduce serialized bytes for a dense BF16 ΔE_i, but it does let the runtime launch the E_{base} GEMM immediately and transfer ΔE_i concurrently; only the transfer time not overlapped by the E_{base} GEMM remains exposed on the critical path before the later delta GEMM. That runtime role also makes ΔE_i construction more constrained, because naive $\Delta E_i = E_i - E_{\text{base}}$ can amplify outliers and produce a representation that is compact on paper but awkward to execute efficiently on GPUs.

Fig. 2(b) profiles the dense naive $\Delta E_i = E_i - E_{\text{base}}$ distributions before SAME, normalized and sorted by maximum σ. Most entries in these naive ΔE_i tensors lie in a compact near-Gaussian bulk: outliers beyond 3σ account for less than 0.5% of entries, and values beyond 6σ are extremely rare. The design challenge is therefore not whether ΔE_i can be compressed, but how to preserve the dominant small-value mass while still accommodating the rare tail in one low-precision format that remains dense in memory and aligned with GPU execution.

3 DELTA-MOE DESIGN

Delta-MoE has three steps: build E_{base} matched to runtime access skew, construct ΔE_i tensors that do not amplify outliers, and encode them as one dense GPU-friendly stream.

3.1 Constructing E_{base}

Delta-MoE begins by constructing the layer-local shared tensor E_{base}. At deployment, each expert is represented relative to this shared E_{base} using expert-specific block-wise sign metadata and an encoded ΔE_i, rather than by transferring the full dense expert tensor. The goal is not minimum reconstruction error alone. If a small subset of experts dominates offloaded execution, E_{base} should absorb their common structure first because those experts define the critical path most often. Fig. 3(a) shows the construction. We first run the pretrained MoE on a small calibration set and record each expert's activation frequency. As shown in Fig. 3(a), only a small set of hot experts is selected frequently, while many others are rarely used [12]. We then use these activation frequencies, f_i, to construct E_{base} so that E_{base} primarily captures the hot experts,

$$\gamma_i = \min\big(f_i,\ \tau \cdot \text{median}(f)\big), \qquad E_{\text{base}} = \frac{\sum_i \gamma_i E_i}{\sum_i \gamma_i}, \tag{2}$$

E_{base} is therefore a clipped frequency-weighted centroid of the experts in that layer. The clipping parameter $\tau > 0$ prevents a few very hot experts from dominating E_{base} and pushing their distinctive structure into every ΔE_i. Experts with activation frequencies above $\tau \cdot \text{median}(f)$ are clipped, with their specific features preserved in the corresponding ΔE_i tensors. τ is tuned offline on calibration data, and Section 4.2.2 evaluates this choice.

3.2 SAME: Sign-Aware, Memory-Aligned Encoding

The streamed ΔE_i path is useful only if it is both numerically stable under low-bit encoding and regular enough for dense GPU execution. We refer to this combined ΔE_i encoding as SAME, short for Sign-Aware, Memory-Aligned Encoding, and use SAME below for brevity. Naively computing $\Delta E_i = E_i - E_{\text{base}}$ violates the first condition: as shown in Fig. 3(b), opposite signs in E_i (+5.0) and E_{base} (-6.0) produce an outlier of 11.0 in ΔE_i. SAME resolves this in two coupled steps: sign-aware ΔE_i construction first makes ΔE_i compressible by suppressing outlier amplification, and memory-aligned encoding then turns that compressibility into a dense executable 4-bit stream.

3.2.1 ***Sign-Aware*** ΔE_i ***Construction:*** SAME begins with an offline sign-aware construction step. For exposition, we first write the ideal element-wise rule as

$$\Delta E_i = E_i - S_i \odot E_{\text{base}}, \tag{3}$$

where S_i is a sign mask (+1 for same-sign pairs, -1 for opposite-sign pairs). When the signs match, SAME uses the usual difference. When the signs differ, SAME flips the contribution of E_{base} and computes $E_i + E_{\text{base}}$, avoiding the amplification that would arise from subtracting two values with opposite signs. This narrows the ΔE_i range and converts most potential outliers into normal values

(case 2 in Fig. 3(b)). Before low-bit encoding, Eq. (3) is an exact reparameterization:

$$E_i = S_i \odot E_{\text{base}} + \Delta E_i. \tag{4}$$

For one element j, let e_j and b_j denote the entries of E_i and E_{base}. If the signs match, $S_{i,j} = +1$ and $\Delta E_{i,j} = e_j - b_j$, so $S_{i,j} b_j + \Delta E_{i,j} = b_j + (e_j - b_j) = e_j$. If the signs differ, $S_{i,j} = -1$ and $\Delta E_{i,j} = e_j + b_j$, so $S_{i,j} b_j + \Delta E_{i,j} = (-1) b_j + (e_j + b_j) = e_j$. The deployed format therefore keeps one $S_i \odot E_{\text{base}}$ term plus one ΔE_i term; SAME quantization only approximates ΔE_i after this reparameterization is formed. In deployment, SAME retains one block-wise sign value $S_B \in \{+1, -1\}$, stored as one sign bit per 16×16 block, for the online execution path. Delta-MoE uses a tensor-core-friendly block-wise approximation to generate this one block-wise sign value during the offline transform:

$$\begin{aligned} S_B &= \operatorname{sign}\Big(\sum_{j \in B} E_i[j]\, E_{\text{base}}[j]\Big), \\ \Delta E_i[B] &= E_i[B] - S_B\, E_{\text{base}}[B]. \end{aligned} \tag{5}$$

Algorithm 1 and the GPU implementation use this block-wise rule and retain the corresponding S_B, encoded as one sign bit per block, for the online execution path. It replaces ideal element-wise decisions with a single offline sign choice per block, reducing sign-bit storage cost while preserving most of the outlier suppression effect (Section 4.2.2). The deployed block therefore still satisfies $E_i[B] = S_B E_{\text{base}}[B] + \Delta E_i[B]$. Runtime consumes E_{base}, the encoded ΔE_i, and the block-wise sign metadata S_B jointly, without materializing a separate dense expert tensor in memory. Unless otherwise noted, ΔE_i below refers to this deployed block-wise sign-aware delta, rather than the naive dense difference $E_i - E_{\text{base}}$.

3.2.2 ***Memory-Aligned*** ΔE_i ***Encoding:*** After sign-aware construction narrows the ΔE_i range, the second part of SAME maps every ΔE_i value into one dense fixed-width stream. Unlike prior mixed-precision schemes that place outliers in side channels [13], SAME keeps normal values and outliers in the same 4-bit layout. ❶ Following the ΔE_i statistics in Section 2.2, we apply a 3σ threshold to each ΔE_i tensor, rather than per block. ❷ Normal values in each 16×16 block are quantized to INT3 with one S1E4M3 scale a_B per block, similar to Microscaling [14] but with a different block size and bitwidth. ❸ Outliers use a customized S1E1M1 abfloat format, since range matters more than mantissa precision. A shared layer-wise exponent bias, β_ℓ, extends the numerical range; unlike the INT3 normal branch, the outlier branch does not use the per-block scale a_B:

$$\textit{outlier} = (-1)^s (1 + m)\, 2^{e + \beta_\ell}. \tag{6}$$

The shared β_ℓ keeps outliers outside the normal-value range. ❹ Each value is then packed as one SAME symbol in a contiguous 4-bit stream: the most significant bit selects INT3 normal value versus S1E1M1 outlier, and the remaining three bits carry the payload. At deployment, the side metadata are exactly one S1E4M3 scale a_B and one sign bit encoding S_B per 16×16 block.

As shown in Fig. 3(c), most normal values fall within the INT3 range, while sign-aware construction cuts outliers ($|z| > 6$) by 2.5× in the 3–6 range and 4.3× versus naive subtraction [15]. The remaining tail is well captured by S1E1M1 with exponent bias $\beta_\ell \in \{2, 3, 4\}$,

Algorithm 1 SAME Encoding and Online Execution

Offline encoding (per expert E_i)

for each 16×16 block B **do**
  $S_B \leftarrow \operatorname{sign}(\sum_{j \in B} E_i[j]\, E_{\text{base}}[j])$, $\Delta_B \leftarrow E_i[B] - S_B \cdot E_{\text{base}}[B]$
  choose block scale a_B (one S1E4M3 value for block B) and threshold $T = 3\sigma$
  for each element $\delta_j \in \Delta_B$ **do**
    if $|\delta_j| \le T$ **then**
      emit $c_j \leftarrow (1 \| \text{int3}(\delta_j / a_B))$
    else
      emit $c_j \leftarrow (0 \| \text{abfloat}(\delta_j, \beta_\ell))$
  store dense block stream $(\{c_j\}, a_B, S_B)$

Online execution

for each block B and symbol $c_j = (m_j \| q_j)$ **do**
  $\hat{\delta}_j \leftarrow a_B \cdot \text{int3}(q_j)$ if $m_j = 1$, else $\hat{\delta}_j \leftarrow \text{abfloat}(q_j, \beta_\ell)$
assemble the decoded ΔE_i block $\hat{\Delta}_B$ from the $\hat{\delta}_j$
use the stored block-wise sign S_B to realize the deployed block relation $E_i[B] = S_B E_{\text{base}}[B] + \hat{\Delta}_B$
execute the E_{base} path and metadata-guided ΔE_i path directly on the deployed artifacts
do not materialize a separate dense expert tensor in memory

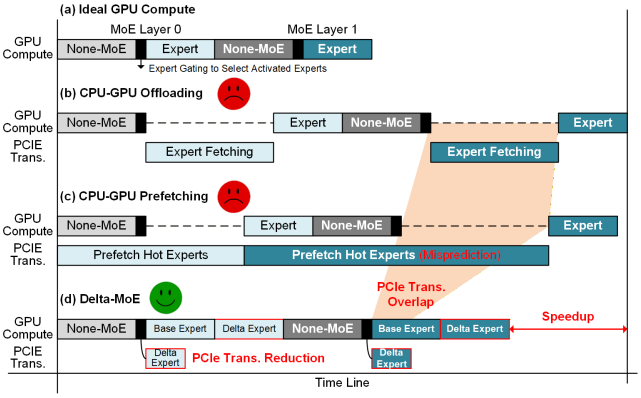

Figure 4: Delta-MoE execution timeline.

leaving only 0.005% overflow. This packing also remains compact in bytes: each 16×16 SAME block stores 256 four-bit symbols (128 B), one S1E4M3 scale a_B (1 B), and one sign bit encoding S_B (0.125 B), for an average payload of about 4.03 bits/element with only tiny metadata overhead. For Mixtral-8×7B, each SAME-encoded ΔE_i occupies ~90 MB versus ~360 MB in BF16 (about 0.252× the bytes, i.e., 4.0× smaller); because E_{base} stays in HBM, only ΔE_i crosses PCIe.

Algorithm 1 summarizes SAME encoding and online execution.

3.3 Delta-MoE Offloading Flow

Fig. 4 isolates the effect of changing the transfer unit. It compares four strategies: (a) oracular GPU-only with all experts in HBM; (b) naive offloading, where the full routed experts of the current layer are fetched after gating and PCIe remains on the critical path; (c) prediction-based prefetching [4, 5, 16], which moves experts earlier but not less; and (d) Delta-MoE, which keeps E_{base} in HBM and streams only compact ΔE_i. Once gating selects experts for layer ℓ, the GPU immediately executes the layer-local $E_{\text{base}}^{(\ell)}$ via

BF16 tensor cores while the host streams $\Delta E_i^{(\ell)}$ through PCIe using asynchronous DMA. Because the E_{base} GEMM and PCIe DMA use separate engines, that E_{base}-compute interval can overlap with part of the ΔE_i transfer time; only any transfer time not hidden by that overlap, followed by the later delta GEMM, remains on the critical path. Unlike prediction-based prefetching, Delta-MoE does not merely move traffic earlier; it changes the object being moved and uses E_{base} computation to overlap with part of that smaller transfer, which is why Fig. 4(d) approaches the GPU-only timeline. Cache-aware and hybrid methods such as MoE-Infinity [6] and HybriMoE [7] remain orthogonal because they can still operate on the now-smaller transfer unit.

3.4 S-MMA Implementation for SAME ΔE_i Decoding

3.4.1 ***Software Implementation of S-MMA on Commodity GPUs.*** We now describe the software realization of SAME on commodity GPUs. After the offline preparation in Sections 3.1 and 3.2, each pretrained MoE layer is deployed as an HBM-resident E_{base}, host-resident SAME-encoded ΔE_i tensors, and compact per-block metadata: one block-wise sign bit encoding S_B and one S1E4M3 scale a_B per 16×16 SAME block.

At runtime, the GPU first launches the BF16 E_{base} GEMM on resident E_{base} while the activated expert's SAME-encoded ΔE_i and metadata are streamed from host memory via PCIe DMA. Once that transfer completes, a fused CUDA kernel decodes the packed 4-bit SAME symbols by separating mode and payload bits, dequantizing normal symbols with the per-block scale a_B, decoding outlier symbols through the S1E1M1 path with layer-wise bias β_ℓ, restoring the stored block-wise sign S_B, and forming the INT8 MMA operand fragment. The software path launches the E_{base} GEMM first and folds the sign-dependent correction into the later metadata-guided ΔE_i path, which is algebraically equivalent to the deployed relation $S_B E_{\text{base}}[B] + \hat{\Delta}_B$ for each block. The runtime therefore executes the deployed sign-aware representation directly, rather than reconstructing a separate dense E_i tensor in HBM before GEMM.

These remaining software overheads are structural rather than implementation accidents. Current tensor cores expose INT8/FP8 operands, not SAME tiles with mode-bit, block-sign, and per-block-scale semantics, so operand unpacking, LUT-based reconstruction, sign recovery, and 4-bit-to-8-bit widening must all happen in software before MMA can begin. These steps add instructions, shared-memory traffic, and register pressure, while the widening step wastes half of the operand bandwidth on the ΔE_i path.

Importantly, these software-side execution overheads do not change the memory-residency assumption behind Delta-MoE. The E_{base} footprint remains well within the CPU–GPU offloading regime we target: across all routed layers, the collection of E_{base} tensors occupies ~11.5 GB for Mixtral-8×7B, ~10.2 GB for Phi-3.5-MoE, and ~1.2 GB for Qwen1.5-MoE-A2.7B. Combined with non-expert weights (5–7 GB), this uses less than 20% of H20's 96 GB HBM, still leaving more than 75 GB for KV cache and activations.

3.4.2 ***Hardware Design of S-MMA for Native SAME Support.*** Figure 5 shows S-MMA, a tensor-core extension that directly executes SAME-encoded ΔE_i. Its scope is intentionally narrow: E_{base} continues to use the existing BF16 FP-MMA path, while only the

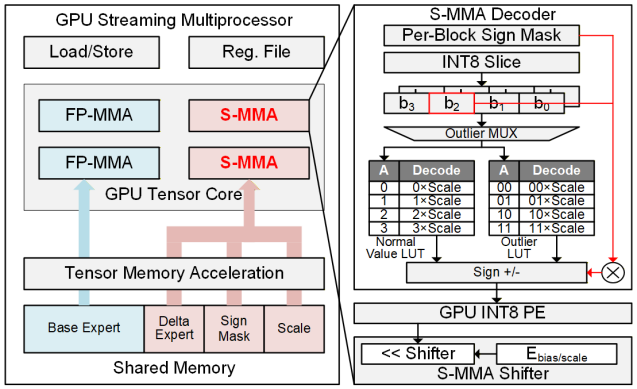

Figure 5: S-MMA hardware datapath for SAME-encoded ΔE_i. Each 16×16 SAME block carries one block-wise sign bit and one S1E4M3 scale; the E_{base} path still uses the standard FP-MMA path, and only the ΔE_i path is extended.

ΔE_i path is augmented. At the SM level, the tensor memory path supplies three objects for the ΔE_i computation: the packed delta block itself, one block-wise sign bit encoding S_B per block, and one S1E4M3 scale a_B per block. These are delivered to the S-MMA decoder rather than first being expanded by a software kernel, and together with the E_{base} path they realize the same deployed block relation defined in Section 3.2.

The right half of Fig. 5 shows the operator-level decode path for one SAME block. First, the decoder slices the packed operand into 4-bit SAME symbols. Within each symbol, the mode bit selects whether the payload should be interpreted by the normal-value LUT or the outlier LUT. The normal branch reconstructs the common small-magnitude values under the per-block scale a_B, while the outlier branch reconstructs the rare large-magnitude values via the S1E1M1/β_ℓ path. The stored block-wise sign S_B then drives the *Sign +/-* stage so that the decoded value matches the sign convention of the ΔE_i path before entering compute.

After decode, the value is forwarded directly into the GPU INT8 processing element, avoiding a separate software-visible decode buffer. Below the INT8 PE, the S-MMA shifter applies the ΔE_i bias/scale term and performs shift-augmented accumulation, $acc \leftarrow acc + (product \ll bias)$, before the result is merged back with the E_{base} path. In other words, S-MMA absorbs operand unpacking, mode selection, LUT-based reconstruction, sign restoration, and shift handling into the tensor-core input path itself.

The commodity-GPU implementation in Section 3.4.1 emulates this same sequence with shared-memory LUTs and register fusion, which is sufficient for all end-to-end results on an unmodified H20. S-MMA is the native hardware realization of that same ΔE_i path. Section 4.3.3 evaluates a tensor-core extension of this design via an `s8_same` operand type.

4 EVALUATION

We evaluate Delta-MoE with a common setup and then proceed from cross-model quality, including robustness to the main offline knobs, to end-to-end throughput from changing the transfer unit, positioning against representative baselines, and the incremental value of native S-MMA support.

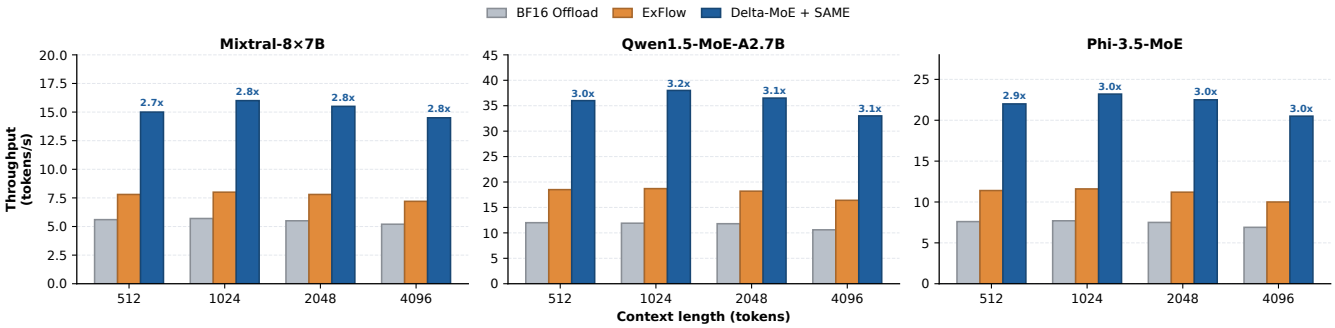

Figure 6: End-to-end MoE inference throughput (tokens/s) under a context sweep with batch size 1 and a fixed 256-token output length.

4.1 Experimental Setup

Unless noted otherwise, all results use the same CPU–GPU offloading testbed: an Intel Xeon Gold 6548Y CPU and an NVIDIA H20 GPU connected through PCIe Gen5 x16.

We evaluate three MoE LLMs spanning 8–60 experts: Mixtral-8×7B [17], Phi-3.5-MoE [18], and Qwen1.5-MoE-A2.7B [19]. Models are factorized and SAME-encoded in PyTorch, and quality is measured on MMLU [20], GSM8K [21], MBPP (pass@1) [22], and C4 perplexity [23]. Unless otherwise stated, throughput uses batch size 1 with 256 prompt tokens and 256 generated tokens; Fig. 6 instead fixes the output length to 256 and sweeps the input context from 512 to 4096 tokens. A single offline calibration uses 128 C4 samples of 2K tokens to set expert activation frequency, τ, and SAME encoding ranges. After this offline step, the deployed E_{base}, ΔE_i, and SAME parameters are fixed and reused across downstream tasks unless otherwise stated.

Tokens/s is measured with `cudaEventRecord`; PCIe transfer time is measured by timing the asynchronous H2D copy path built on `cudaMemcpyAsync`. *Delta-MoE + SAME* denotes the software realization of the SAME execution path, which fuses SAME decode and delta GEMM in CUDA. *Native S-MMA* denotes the tensor-core extension of the same datapath evaluated in Section 4.3.3 using Verilog, Synopsys Design Compiler and IC Compiler, Accel-Sim [24], and GPUWattch [25]. In Section 4.3, *BF16 expert bytes / fetch* denotes the serialized H2D bytes of one activated expert in the standard BF16 offload path, excluding non-expert layers that stay in HBM, while *SAME bytes / fetch* denotes the packed 4-bit payload plus the per-block metadata: one S1E4M3 scale and one sign bit per 16×16 block. Under the current dense SAME layout, this representation averages 4.03 bits/element, or about 0.252× of BF16.

4.2 Cross-Model Quality of Delta-MoE

4.2.1 ***Delta-MoE Quality and Aligned Ablations Study.*** Table 1 compares four aligned variants on all three models: the original *BF16* model, *Delta + BF16*, *Naive Delta + INT4 (GPTQ)*, and *Delta-MoE + SAME*. Delta + BF16 and Naive Delta + INT4 generate the delta expert via direct subtraction as described in Section 3.2, while the latter further quantizes the delta expert to INT4.

The main result is that Delta-MoE tracks BF16 closely across all three MoE families, while naive 4-bit delta quantization does not. *Delta + BF16* mainly confirms that the factorization itself is benign at high precision. The critical ablation is *Naive Delta + INT4 (GPTQ)*,

Table 1: Cross-model quality of Delta-MoE and aligned ablations.

Model	Method	MMLU	GSM8K	MBPP (pass@1)	C4 (PPL)
Mixtral-8×7B (47B, 13B active, 8 experts)	BF16 baseline	70.6	58.4	60.7	6.78
	Delta + BF16	70.6	58.3	60.5	6.80
	Naive Delta + INT4 (GPTQ)	68.5	56.2	58.0	7.40
	Delta-MoE + SAME (our work)	70.3	57.9	60.1	7.00
Qwen1.5-MoE-A2.7B (14.3B, 2.7B active, 60 experts)	BF16 baseline	62.5	61.5	36.6	10.5
	Delta + BF16	62.4	61.4	36.4	10.6
	Naive Delta + INT4 (GPTQ)	60.4	57.9	33.3	12.1
	Delta-MoE + SAME (our work)	62.1	60.8	35.7	10.9
Phi-3.5-MoE (42B, 6.6B active, 16 experts)	BF16 baseline	78.9	88.7	73.0	8.22
	Delta + BF16	78.9	88.6	72.9	8.25
	Naive Delta + INT4 (GPTQ)	76.5	86.1	70.2	9.20
	Delta-MoE + SAME (our work)	78.6	88.0	72.5	8.60

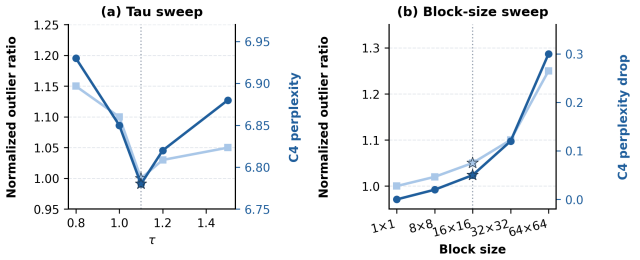

Figure 7: Sensitivity to τ and SAME block size. Star markers indicate the defaults used in the main experiments.

which isolates what happens when ΔE_i is quantized without sign-aware handling. The difference is easiest to read on C4. Naive Delta + INT4 raises perplexity from 6.78 to 7.40 on Mixtral-8×7B, from 10.5 to 12.1 on Qwen1.5-MoE-A2.7B, and from 8.22 to 9.20 on Phi-3.5-MoE. Delta-MoE + SAME recovers most of that loss because it first suppresses outlier amplification in ΔE_i and only then applies fixed-width 4-bit encoding. The ordering across models is also consistent with Section 2.2: Qwen1.5-MoE-A2.7B is the hardest case, yet Delta-MoE still remains close to BF16 and removes most of the gap introduced by naive ΔE_i quantization.

4.2.2 ***Design Space Exploration and Robustness Analysis.*** Fig. 7 sweeps the two main offline knobs: the clipping threshold τ used to construct E_{base}, and the SAME block size used by the sign-aware transform.

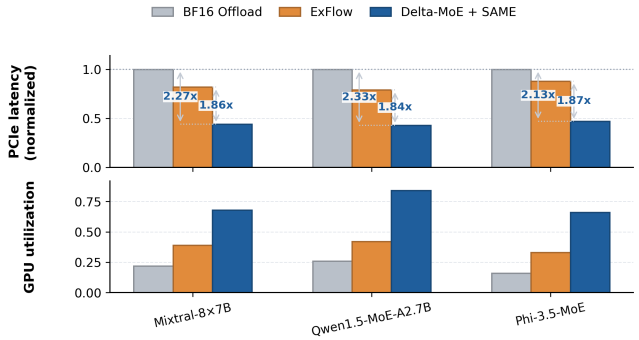

Figure 8: PCIe latency (top, normalized) and GPU utilization (bottom) under three runtime strategies.

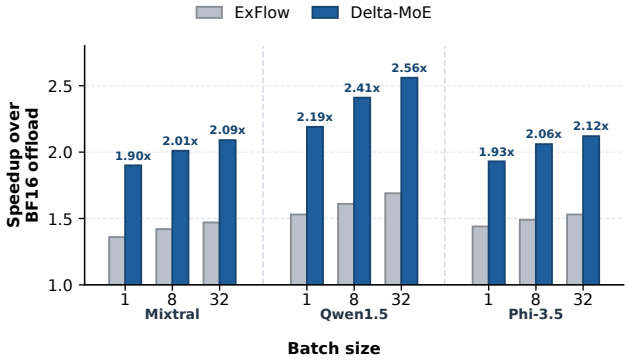

Figure 9: Multi-batch scaling on three models (C4, 1024/512 tokens).

The sweep shows a broad stable region rather than a knife-edge optimum. Fig. 7(a) shows that $\tau = 1.1$ gives the best balance between normalized outlier ratio and perplexity on Mixtral-8×7B under C4 calibration, while Fig. 7(b) shows that 1×1 gives the best quality but the degradation remains small up to 16×16 and becomes more visible only for larger blocks. We therefore use $\tau = 1.1$ and a 16×16 SAME tile as the default configuration because this choice stays close to the best-quality point while matching the tensor-core-friendly tile granularity of the implementation. Importantly, the offline calibration is performed once on C4 to determine the expert activation statistics, the clipping threshold τ, and the SAME quantization parameters; after compression, the resulting E_{base}, ΔE_i, and SAME metadata are fixed and directly reused for inference on MMLU, GSM8K, and MBPP without further calibration. Table 1 shows that this C4-calibrated configuration remains close to BF16 across all three downstream workloads, so $\tau = 1.1$ should be read as a robust transferred default rather than as a workload-universal optimum.

To further probe workload shift, we additionally recalibrated on MMLU, GSM8K, and MBPP. True SAME overflow beyond the representable range remains very small at 0.021%, 0.012%, and 0.006%, respectively, indicating that these downstream workloads stay within a similar quantization regime even when the deployed compression is calibrated once on C4. When workload-specific adaptation is desired, recalibration is lightweight: for Mixtral-8×7B, calibrating with 128 samples of 2K tokens takes only 1.2 minutes [26]. This behavior is also consistent with prior observations that MoE experts exhibit stable specialization patterns across related workloads [27].

4.3 End-to-End Throughput

4.3.1 ***MoE Inference Speedup.*** Fig. 6 gives the headline throughput result across a context sweep and compares them with two baselines, *BF16 Offload* and *ExFlow* [4]. *BF16 Offload* loads the activated experts from host memory on demand, whereas *ExFlow* exploits cross-layer expert affinity to predict and prefetch the experts activated in the subsequent layer. All end-to-end results of Delta-MoE run on a commodity H20 GPU without the S-MMA tensor core extension (described in Section 3.4.2).

Delta-MoE sustains the highest throughput among the reproduced runtime baselines, and combining it with ExFlow adds another 1.1× average speedup. The gain is therefore complementary to scheduling and prefetching rather than a replacement for them. Outside this sweep, at batch size 1 with 256 prompt tokens and 256 generated tokens, Delta-MoE reaches 16.0, 38.0, and 23.2 tokens/s on Mixtral-8×7B, Qwen1.5-MoE-A2.7B, and Phi-3.5-MoE, corresponding to 2.8×–3.2× speedup over BF16 Offload and 1.9×–2.1× over ExFlow. For longer output token sequences, Delta-MoE sustains its speedup over the baselines, as LLM decoding latency is dominated by PCIe transfers.

Fig. 8 explains the source of the gain. Delta-MoE reduces PCIe latency by 2.1×–2.3× versus BF16 Offload and by 1.8×–1.9× versus ExFlow, while GPU utilization increases modestly. The speedup therefore comes primarily from a shorter and more overlapped post-routing transfer path, not from a large increase in raw compute throughput. A separate 2K-token Mixtral-8×7B micro-profile matches this explanation: BF16 Offload spends 11 ms on PCIe transfer and 4 ms on computation per expert, whereas Delta-MoE reduces the transfer payload to the 0.252× SAME representation and overlaps that transfer with the E_{base} GEMM. ExFlow relies on probabilistic inter-layer expert affinity and can reduce communication by up to 67%, with inevitable mispredictions limiting further gains. Therefore, this is the end-to-end payoff of changing *what* moves after routing, not only *when* it moves.

4.3.2 ***Multi-Batch Throughput Analysis.*** To evaluate the robustness of Delta-MoE, we conduct additional experiments under multi-batch settings. In Fig. 9, with 1024-token prompts and 512 generated tokens, ExFlow delivers 1.36×–1.47× speedup over BF16 Offload on Mixtral-8×7B, 1.53×–1.69× on Qwen1.5-MoE-A2.7B, and 1.44×–1.53× on Phi-3.5-MoE across batch sizes 1–32. Delta-MoE remains higher at every batch size, reaching 1.90×–2.09×, 2.19×–2.56×, and 1.93×–2.12× on the same three models. Absolute speedup is lower than in the default batch-size-1 case. That's because non-expert computation accounts for more of end-to-end latency with batch size increasing, but the speedup trend stays flat or improves slightly with batch, which is consistent with ΔE_i H2D time being hidden under the E_{base} path computation.

4.3.3 ***Analysis of S-MMA Tensor-core Extension.*** Fig. 10 reports normalized energy per token for BF16 offload, the software SAME CUDA kernel, and S-MMA tensor-core under the representative Mixtral-8×7B 2K-token workload. S-MMA hardware extends only the SAME-encoded ΔE_i compute path; routing, PCIe transfer, and overlap scheduling remain unchanged, so we treat this as

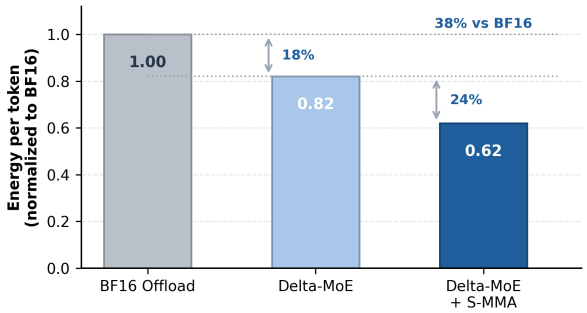

Figure 10: Normalized energy per token for BF16 offload, the software SAME path, and native S-MMA.

an incremental hardware evaluation rather than a multi-workload end-to-end sweep.

Relative to BF16 offload, the software SAME path reduces normalized energy per token from 1.00 to 0.82, and native S-MMA further reduces it to 0.62, as estimated with Accel-Sim and GPUWattch [25]. This corresponds to a 1.6× energy-efficiency gain over BF16 and a 1.3× gain over the software path. The synthesized extension corresponds to an estimated 2.1 mm^2 of H20-equivalent die area, about 0.26% of the die. Native S-MMA is therefore a small optional extension that further improves the SAME datapath already validated in software, not a prerequisite for the main end-to-end speedups in this paper.

4.4 Comparison to SOTAs

Table 2 compares Delta-MoE against representative SOTA baselines for the same CPU–GPU MoE offloading bottleneck. We evaluate the perplexity (PPL) and throughput (tokens/s) of Mixtral-8×7B on the C4 dataset. The batch size is set to 64. Each sample uses 1K input prompt tokens to generate an additional 256 output tokens. The prediction in ExFlow [4], quantization in EAQuant [9] and low-rank approximation in D^2-MoE [28] are applied exclusively to expert layers. D^2-MoE is reproduced from the authors' released codebase to run on the same H20 system. In addition, the proposed Delta-MoE is evaluated using the S-MMA CUDA kernel without tensor core extensions.

Table 2 shows where each baseline saturates under the same Mixtral-8×7B setting. ExFlow improves throughput from 121 to 183 tokens/s while keeping C4 perplexity essentially unchanged (6.80 vs. 6.78), which is consistent with its design: it exploits probabilistic inter-layer expert affinity and, in its own evaluation, reports up to 67% lower routing latency, but it still does not remove post-routing expert transfer because the transferred object remains a full routed.

EAQuant-4bit pushes throughput further to 251 tokens/s by directly compressing the transferred expert, but its C4 perplexity rises to 7.60. While EAQuant reduces the bytes of each routed expert, it does not redesign the post-routing transfer object or co-design the compressed representation with the GPU execution path. As a result, the throughput gain from fewer transferred bytes is limited.

D^2-MoE is the closest structural comparison because it also exploits inter-expert redundancy, but its design target is different. In D^2-MoE, the base-plus-delta form is used primarily as a compression-and-reconstruction format, with a merged base and an SVD-truncated low-rank delta, whereas Delta-MoE uses the factorization to redesign the post-routing transfer unit for GPU serving. Using the authors' released implementation on our H20 testbed, D^2-MoE reaches 262 tok/s and 12.40 PPL; this perplexity is close to the SVD result reported in the original paper on A100, suggesting that the observed quality gap is not merely an artifact of our rerun. One plausible contributor is that its delta construction does not suppress outlier amplification before compression; another is that the SVD-truncated reconstruction is more sensitive to truncation error, consistent with prior work on SVD-based LLM compression [29]. Delta-MoE + SAME instead forms a sign-aware ΔE_i, keeps the shared E_{base} resident, and streams only the SAME-encoded post-routing delta. This gives a much better serving operating point in Table 2, improving throughput from 262 to 341 tok/s while reducing C4 perplexity from 12.40 to 7.00.

Table 2: Comparison on Mixtral-8×7B (H20; bs=64; C4). Parentheses report ΔPPL and throughput speedup relative to BF16 offload.

	Method	PCIe Transfer Granularity	PPL on C4	Throughput (Token/s)
BF16 Offload	Offload	Coarse-Grained Full Expert	6.78	121
ExFlow	Prediction+ Offload	Coarse-Grained Full Expert	6.80 (+0.02)	183 (×1.5)
EAQuant-4b	Quantization +Offload	Coarse-Grained Full Expert	7.60 (+0.82)	251 (×2.1)
D^2-MOE	Delta+SVD	Fine-Grained ΔE	12.40 (+5.62)	262 (×2.2)
Delta-MoE	Delta+SAME	Fine-Grained ΔE	7.00 (+0.22)	341 (×2.8)

Taken together, these results indicate that the measured advantage comes from redesigning the post-routing transfer object to a compact ΔE_i around resident E_{base}, rather than from scheduling alone, direct quantization alone, or delta compression alone. Cache-aware and hybrid serving systems such as MoE-Infinity and HybriMoE [6, 7] remain orthogonal because they optimize placement and reuse rather than transfer granularity.

5 CONCLUSION

We introduced Delta-MoE, a CPU–GPU MoE serving design that changes the post-routing transfer unit from a full expert to a streamed ΔE_i around a resident E_{base}. By constructing E_{base} offline and encoding ΔE_i with SAME, Delta-MoE preserves near-BF16 accuracy while improving end-to-end throughput by up to 3.2× and reducing PCIe transfer latency by up to 2.3× across three MoE LLM families on H20. The same transfer path also maps naturally to native support: our S-MMA study shows that a small tensor-core extension can further improve energy efficiency over the software SAME path at modest area cost.

More broadly, this work shows that post-routing transfer granularity is a first-class systems/codesign knob in MoE serving. Future systems should therefore rethink the runtime transfer object itself, not only when or where data moves, and co-design representation, runtime overlap, and datapath support around transfer units that are both compact and regular. This perspective opens a broader design space for transfer-aware expert representations that balance compression efficiency, reconstruction cost, and accelerator compatibility. It also suggests that expert placement and scheduling policies should account for the amount and structure of transferred state rather than treating every expert invocation uniformly.

REFERENCES

[1] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A Survey on Efficient Inference for Large Language Models. arXiv:2404.14294 [cs.CL]

[2] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. arXiv:2006.16668 [cs.CL]

[3] Dianhai Yu, Liang Shen, Hongxiang Hao, Weibao Gong, Huachao Wu, Jiang Bian, Lirong Dai, and Haoyi Xiong. 2024. MoESys: A Distributed and Efficient Mixture-of-Experts Training and Inference System for Internet Services. arXiv:2205.10034 [cs.DC]

[4] Jinghan Yao, Quentin Anthony, Aamir Shafi, Hari Subramoni, Dhabaleswar K., and Panda. 2024. Exploiting Inter-Layer Expert Affinity for Accelerating Mixture-of-Experts Model Inference. arXiv:2401.08383 [cs.LG]

[5] Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. 2024. Pre-gated MoE: An Algorithm-System Co-Design for Fast and Scalable Mixture-of-Expert Inference. arXiv:2308.12066 [cs.LG]

[6] Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. 2025. MoE-Infinity: Efficient MoE Inference on Personal Machines with Sparsity-Aware Expert Cache. arXiv:2401.14361 [cs.LG]

[7] Shuzhang Zhong, Yanfan Sun, Ling Liang, Runsheng Wang, Ru Huang, and Meng Li. 2025. HybriMoE: Hybrid CPU-GPU Scheduling and Cache Management for Efficient MoE Inference. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, San Francisco, CA, USA, 1–7. https://doi.org/10.1109/DAC63849.2025.11133274

[8] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers. Proceedings of the International Conference on Learning Representations (ICLR).

[9] Zhongqian Fu, Ning Ding, Kai Han, Xianzhi Yu, Xiaosong Li, Xinghao Chen, Yehui Tang, and Yunhe Wang. 2025. EAQuant: Enhancing Post-Training Quantization for MoE Models via Expert-Aware Optimization. arXiv:2506.13329. Available at: https://arxiv.org/abs/2506.13329.

[10] Taehyun Kim, Kwanseok Choi, Youngmock Cho, Jaehoon Cho, Hyuk-Jae Lee, and Jaewoong Sim. 2024. MoNDE: Mixture of Near-Data Experts for Large-Scale Sparse Models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '24)*. Association for Computing Machinery, New York, NY, USA, Article 221, 6 pages. https://doi.org/10.1145/3649329.3655951

[11] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. 2012. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research* 13, 1 (2012), 795–828.

[12] Zihan Qiu, Zeyu Huang, Bo Zheng, Kaiyue Wen, Zekun Wang, Rui Men, Ivan Titov, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models. arXiv:2501.11873 [cs.LG]

[13] Tianshi Xu, Shuzhang Zhong, Wenxuan Zeng, Runsheng Wang, and Meng Li. 2024. PrivQuant: Communication-Efficient Private Inference with Quantized Network/Protocol Co-Optimization. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. Association for Computing Machinery, New York, NY, USA, 1–9.

[14] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. 2023. Microscaling data formats for deep learning. arXiv:2310.10537 [cs.LG]

[15] Mengxia Yu, De Wang, Qi Shan, Colorado J Reed, and Alvin Wan. 2025. The Super Weight in Large Language Models. arXiv:2411.07191 [cs.CL]

[16] Jiaming Yan, Jianchun Liu, Hongli Xu, and Liusheng Huang. 2025. Accelerating Mixture-of-Expert Inference with Adaptive Expert Split Mechanism. arXiv:2509.08342 [cs.LG]

[17] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. arXiv:2401.04088 [cs.CL]

[18] Microsoft. 2024. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. arXiv:2404.14219 [cs.CL]

[19] Qwen Team. 2024. Qwen1.5-MoE: Matching 7B Model Performance with 1/3 Activated Parameters. https://qwenlm.github.io/blog/qwen-moe/. Blog post, accessed 2026-04-12.

[20] Qihao Zhao, Yangyu Huang, Tengchao Lv, Lei Cui, Qinzheng Sun, Shaoguang Mao, Xin Zhang, Ying Xin, Qiufeng Yin, Scarlett Li, et al. 2025. Mmlu-cf: A contamination-free multi-task language understanding benchmark. Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 13371–13391.

[21] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. arXiv:2410.05229 [cs.CL]

[22] Rem Yang, Julian Dai, Nikos Vasilakis, and Martin Rinard. 2025. Evaluating the generalization capabilities of large language models on code reasoning. arXiv:2504.05518 [cs.SE]

[23] Wanrong Zhu, Jack Hessel, Anas Awadalla, Samir Yitzhak Gadre, Jesse Dodge, Alex Fang, Youngjae Yu, Ludwig Schmidt, William Yang Wang, and Yejin Choi. 2023. Multimodal c4: An open, billion-scale corpus of images interleaved with text. *Advances in Neural Information Processing Systems* 36 (2023), 8958–8974.

[24] Mahmoud Khairy, Zhesheng Shen, Tor M Aamodt, and Timothy G Rogers. 2020. Accel-sim: An extensible simulation framework for validated gpu modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, Los Alamitos, CA, USA, 473–486.

[25] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling energy optimizations in GPGPUs. *ACM SIGARCH computer architecture news* 41, 3 (2013), 487–498.

[26] Maohao Shen, Subhro Das, Kristjan Greenewald, Prasanna Sattigeri, Gregory W. Wornell, and Soumya Ghosh. 2024. Thermometer: Towards Universal Calibration for Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, Vienna, Austria, 44687–44711.

[27] Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. 2024. OpenMoE: An Early Effort on Open Mixture-of-Experts Language Models. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 55625–55655.

[28] Hao Gu, Wei Li, Lujun Li, Qiyuan Zhu, Mark G. Lee, Shengjie Sun, Wei Xue, and Yike Guo. 2025. Delta Decompression for MoE-based LLMs Compression. Proceedings of the 42nd International Conference on Machine Learning, Proceedings of Machine Learning Research, Vol. 267. pp. 20497–20514.

[29] Hayuan Wang, Mingzhao Yang, Yifan Wang, Iman Mirzadeh, and Hadi Esmaeilzadeh. 2025. SVD-LLM: Truncation-aware Singular Value Decomposition for Large Language Model Compression. International Conference on Learning Representations.