MoE-PACT: Confidential MoE Inference via Searched Bit Partitioning and Optimal Clustering of Expert Tensors

Anonymous authors
Paper under double-blind review

Abstract

Mixture-of-Experts (MoE) architectures are widely deployed thanks to their low training and inference compute cost, but their large parameter counts impose severe memory and host-to-device (H2D) transfer bottlenecks. These bottlenecks are most severe on cloud server nodes whose H2D bandwidth is restricted by NVIDIA Confidential Computing (CC) mode. Existing solutions for memory and bandwidth overhead come at the price of degraded accuracy, or of expensive retraining or auxiliary memory for mixed-precision weight storage.

We present MoE-PACT, an MoE inference system that reduces memory footprint and H2D traffic under Confidential Computing. We first systematically analyze NVIDIA CC mode to characterize the PCIe bottlenecks introduced by encryption and decryption, achieving a $5.3\times$ bandwidth gain through vector AES and multiprocess transfers. We then propose an exhaustive per-tensor search over base-delta bit partitions with optimal 1D k-means clustering, and a fused GPU kernel that performs bit-unpacking, lookup-table (LUT) base-delta reconstruction, and dequantization on the fly. Against the strongest baselines under CC, MoE-PACT stores 25% fewer expert bytes and cuts H2D traffic by up to half, accelerating decoding by up to $1.65\times$ at batch size one and $2.01\times$ at large batches, with negligible accuracy degradation.

1 Introduction

Mixture-of-Experts (MoE) has become an important architecture for building capable large language models (LLMs), such as Mixtral 8x7B Jiang et al. (2024), DeepSeek-V3 DeepSeek-AI et al. (2025), Qwen2.5 Team et al. (2025), and Grok-1 xAI Team (2024). By activating only a small subset of experts for each token, MoE increases model capacity while limiting the computation required for training and inference, improving energy efficiency and reducing serving costs. Deployments that process sensitive data must obtain these benefits under confidentiality requirements, preventing an untrusted host operating system, hypervisor, or infrastructure administrator from accessing private user inputs and proprietary model weights. Confidential Computing (CC) provides hardware support for this, combining CPU technologies such as Intel TDX Cheng et al. (2023) and AMD SEV-SNP Advanced Micro Devices, Inc. (2020) with NVIDIA GPU CC Apsey et al. (2023); NVIDIA (2025) to extend isolation, attestation, and protected transfers across the CPU-GPU boundary.

Confidential deployments, however, face constraints on both GPU memory capacity and its expansion. Sparse activation reduces the parameters used to process each token, but all expert weights must still be stored, which is often infeasible on a single GPU, and resident weights must share GPU memory with the KV cache and runtime buffers. Expanding GPU resources under CC requires compatible device protection, interconnect security, and software support: Hopper's single-GPU pass-through CC mode assigns at most one GPU to a Confidential Virtual Machine (CVM), and protected multi-GPU operation depends on supported platforms and interconnect configurations with corresponding security assumptions NVIDIA (2025). We therefore focus on confidential deployments with limited protected GPU resources, where GPU memory cannot accommodate the complete model and its runtime state.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Protected CPU memory within the CVM offers a direct way to extend model storage capacity. Expert offloading keeps most weights in CPU memory, caches a subset on the GPU, and fetches the rest during execution Eliseev & Mazur (2023); Xue et al. (2024). This lets large models run within a limited GPU memory budget, but layerwise streaming and expert cache misses make every decoding step refetch weights over PCIe. With 75% of Mixtral-8×7B's expert weights offloaded, decoding accounts for 96% of total inference time in our measurements, motivating a closer look at the decoding path.

CC makes this data movement substantially more expensive. Under the NVIDIA Hopper CC transfer mechanism, host data is encrypted by the CPU and decrypted by the GPU, so effective H2D bandwidth is bounded by these operations Apsey et al. (2023); NVIDIA (2025); Gu et al. (2026). Our per-layer measurements show that, even with optimized host encryption (Section 3), CC leaves computation unchanged but increases H2D transfer time by $6.1\times$, making it an even more severe bottleneck. Therefore, limited protected GPU memory forces offloading, and offloading triggers frequent encrypted transfers: together they bottleneck confidential MoE inference.

How can we deliver efficient MoE inference under CC's memory and bandwidth constraints?

We present *MoE-PACT*, an MoE inference system for GPU-memory-limited deployments that jointly optimizes CC transfers, expert weight representation, and GPU execution. We first characterize CC mode and find that CPU Vector AES (VAES) instructions and multiprocess transfer parallelism raise encrypted H2D bandwidth from 4.5 to 24 GB/s, yet leave it at under half of normal mode, motivating a reduction in the bytes traversing the encrypted path. Prior quantization and adaptive-precision approaches reduce those bytes but trade against quantization error, expensive retraining or per-dataset calibration, and GPU reconstruction overhead Lin et al. (2024); Frantar & Alistarh (2024); Tang et al. (2026). MoE-PACT instead represents each expert weight tensor as a base and a delta components, quantizing both offline with one-dimensional k-means, for which dynamic programming yields the optimal clustering minimizing within-cluster squared error Wang & Song (2011), and searching bit assignments to the two components under a fixed budget. Custom fused GPU kernels integrate bit unpacking, lookup-table (LUT) reconstruction, and dequantization into the matrix computation, so inference consumes the compact representation directly. At batch size one, routing weights guide on-demand delta fetching; at larger batches, the compact representation continues to reduce storage and transfer volume, covering both regimes where prior work targets only one Tang et al. (2026); Liu et al. (2026); Cao et al. (2024).

This paper makes the following contributions:

- **Characterization of confidential offloading bottlenecks.** We systematically measure the H2D path under CC, analyze the interaction between CPU encryption, transfer parallelism, and GPU decryption, demonstrate improvements from VAES and multiprocess transfers, and quantify the bandwidth gap that remains after these optimizations.
- **Compact weight representation for both batch size one and large batch serving.** We combine per-expert base-delta decomposition, one-dimensional clustering, and bit partition search into a compact weight representation that reduces memory and transfer volume with negligible accuracy loss. Unlike prior designs tuned for a single serving regime, it remains effective across batch sizes, which require different expert fetching strategies.
- **Fused GPU kernels for on-the-fly weight reconstruction.** We develop custom kernels that fuse bit unpacking, LUT reconstruction, and dequantization into the matrix computation, keeping reconstruction overhead hidden behind the H2D transfers it replaces.
- **Implementation and evaluation.** We implement MoE-PACT on top of MoE-Lightning Cao et al. (2024), integrating host-side execution management with custom Triton GPU kernels, and evaluate H2D transfer volume, model accuracy, and decode throughput across batch sizes in normal and CC modes.

2 Background and Related Work

2.1 Sparse MoE Execution

A Mixture-of-Experts (MoE) layer replaces the dense feed-forward network in a Transformer block with multiple independent expert networks. For each token, a router scores every expert, dispatches

the token to the top-k experts, and aggregates their outputs weighted by the routing scores Zhang et al. (2025).

Sparse routing limits computation to a subset of experts, but all expert weights must still be stored: Mixtral-8×7B holds roughly 47B parameters while activating 13B per token, so its footprint far exceeds that of a dense model with comparable inference cost Jiang et al. (2024); Lepikhin et al. (2020); Chitty-Venkata et al. (2025). When the full model exceeds GPU memory, expert weights are kept in CPU memory and transferred over PCIe when selected.

2.2 Confidential CPU-GPU Transfers

Intel TDX Cheng et al. (2023) and AMD SEV-SNP Advanced Micro Devices, Inc. (2020) protect Confidential Virtual Machines (CVMs), while NVIDIA GPU CC protects on-GPU computation and data Apsey et al. (2023); NVIDIA (2025). Because the GPU cannot directly access private CVM memory, data must be encrypted, staged in shared host memory, transferred over PCIe, and decrypted in protected GPU memory (Figure 1). Expert offloading repeatedly exercises this path, whose effective bandwidth depends on host-side encryption, transfer parallelism, and GPU-side decryption (Section 3).

2.3 Related Work

Batch-Size-One MoE Offloading. For low-latency single-user workloads, systems try to exploit routing locality and predictability. MoE-Infinity Xue et al. (2024) uses activation tracing to predict and prefetch experts from CPU DRAM. EARTH Liu et al. (2026) and MoE-APEX Tang et al. (2026) cache experts on the GPU and fetch weights at dynamic precision, using base-delta encoding and a second INT2 copy of weights, respectively. Both target batch size one; at large batches, token diversity activates nearly all experts, leaving little room for selective precision.

Large-Batch MoE Offloading. Large-batch serving shifts focus from expert caching to link saturation and I/O overlapping. FlexGen Sheng et al. (2023) introduces block-by-block pipelining optimized via linear programming over the memory hierarchy, and MoE-Lightning Cao et al. (2024) adapts this to MoE models with a CPU-GPU-I/O overlapping pipeline (`CGOPipe`) and a Hierarchical Roofline Model. Neither targets batch-size-one inference or Confidential Computing, and vLLM Kwon et al. (2023), though general across models and configurations, is not designed for CC offloading, and in our attempts it was unable to serve our offloaded configurations under CC.

Confidential LLM Serving. PipeLLM Tan et al. (2025) addresses the NVIDIA CC encryption bottleneck by interposing on CUDA transfer and OpenSSL calls to decouple encryption from memory copying, adding speculative ciphertext preparation, IV prediction, and NOP padding; its own security discussion acknowledges unresolved NOP-related side channels. It also targets dense models, where the weights needed for future computation are easier to predict than for sparse MoE at batch size one. Our system instead retains native NVIDIA CC encryption, IV management, and authenticated transfer, so our reported bandwidth characterizes the native CC path.

3 Accelerating H2D Transfers in CC Mode

Confidential MoE inference repeatedly transfers offloaded expert weights into protected GPU memory, making encrypted H2D bandwidth critical to serving performance. We accelerate the native CUDA CC transfer path through CPU-feature exposure and application-level concurrency, retaining the driver's encryption and IV management. **With CC mode, exposing vector encryption instructions and using four concurrent transfer processes increases H2D bandwidth from approximately 4.5 to 24 GB/s, a $5.3\times$ improvement on our platform.**

Diagnosing Host Encryption. Our H2D microbenchmark transfers pinned host memory to GPU memory across payloads from 1 MB to 16 GB (Appendix C). For large transfers the baseline reaches approximately 4.5 GB/s under CC against 50–55 GB/s in normal mode, an 11–12× gap consistent with prior characterizations of the unoptimized path (5.8 GB/s in Tan et al. (2025); roughly 4 GB/s, attributed to CPU encryption, in Apsey et al. (2023)).

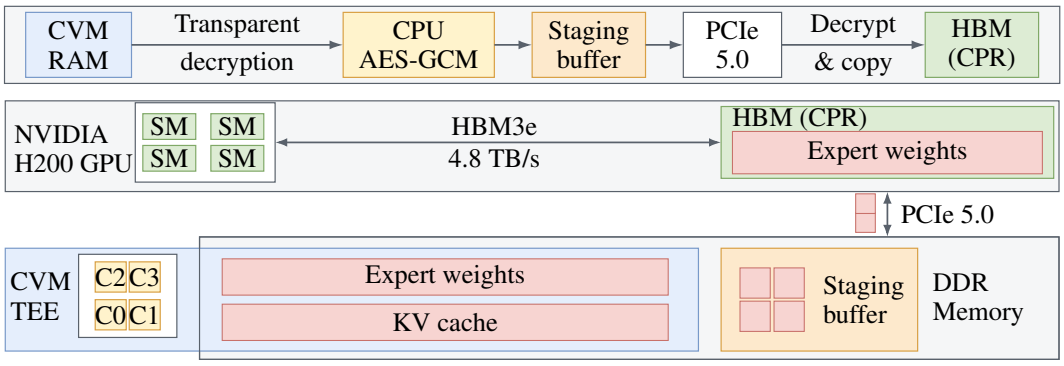

Figure 1: Confidential CPU–GPU data movement. Top: the in-CVM driver encrypts with AES-GCM into a shared staging buffer, and the GPU Copy Engines decrypt and authenticate into protected memory. Bottom: where each structure lives Apsey et al. (2023); Gu et al. (2026).

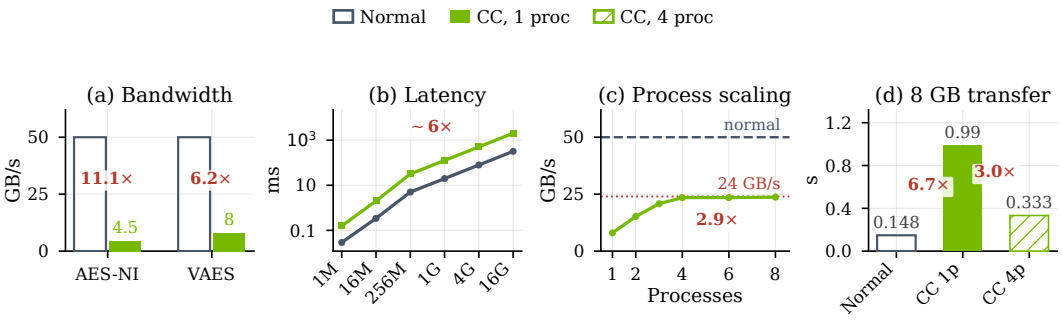

Figure 2: Accelerating H2D transfers under CC. (a) Sustained bandwidth under the two host encryption configurations. (b) Transfer latency from 1 MB to 16 GB. (c) Aggregate CC bandwidth against transfer process count, with the normal-mode reference and the measured saturation point. (d) Time for an 8 GB transfer.

Profiling attributes roughly 90% of execution time to copy routines in `libcuda.so`, which invoke OpenSSL with AES-NI before writing to the staging buffers, pointing to the CPU features exposed to the CVM as the limiting factor. On our VAES-capable host, exposing VAES and AVX-512 enables vectorized encryption and raises single-process CC bandwidth to 8–9 GB/s (Figure 2a,b). This nearly 2× improvement requires no driver changes and identifies CVM CPU-feature configuration as a practical factor in confidential GPU I/O performance.

Scaling Encrypted Transfers with CPU Parallelism. Vectorized encryption improves each transfer process, while concurrent processes let the workload use additional CPU resources. Running multiple H2D transfer processes within the same CVM under VAES and CC mode, aggregate bandwidth grows with process count to approximately 24 GB/s at four processes (Figure 2c), cutting 8 GB transfer latency by roughly 3× relative to single-process VAES (Figure 2d). Together the two optimizations recover 44–48% of the 50–55 GB/s measured with CC disabled, against 8–9% for the baseline CC configuration: a substantial part of the bandwidth loss is recoverable through host configuration and transfer parallelism alone, while keeping confidential computing enabled.

Beyond four processes we observe little additional throughput, even with eight vCPUs assigned to the CVM, so the plateau is not a guest CPU limit. We therefore report 24 GB/s as the saturation point of our complete configuration rather than an isolated hardware decryption limit, since sustained throughput also depends on staging-buffer management, driver scheduling, and GPU-side decryption (Appendix C). These optimizations make substantially more bandwidth available for expert offloading, but the remaining gap to normal-mode bandwidth motivates reducing the bytes that traverse the encrypted path at all.

4 MoE-PACT

4.1 Compression Algorithm

As shown in Sections 2 and 3, the two major bottlenecks for MoE inference are limited GPU memory capacity and H2D transfer bandwidth. To address these bottlenecks, we employ base-delta decomposition of the expert weights and further compress each part. The two-part decomposition lets us use adaptive precision fetching in batch size one serving, where deltas are fetched only for experts whose routing weights indicate significance. Compressing the base and delta components further reduces memory footprint and H2D usage, benefiting both small- and large-batch serving.

This approach resembles the base-delta encoding in EARTH Liu et al. (2026), but instead of a fixed bit assignment between base and delta, we search for the best split according to our error model and select the best compression per weight tensor, sustaining accuracy across batch sizes. The decomposition also yields two precision levels at batch size one without storing additional copies of the weights at different precisions, unlike prior adaptive-precision schemes Tang et al. (2026).

The critical question is which bit partition to use and how to compress each component. At first glance, allocating the higher-order bits to the base seems preferable, since they contribute more numerically. However, they are set less frequently and are less uniformly distributed, which creates an opportunity for compression, as most weights cluster around zero Noune et al. (2023). Small but frequent quantization errors in the lower-order bits also accumulate, especially where the affected weights meet large activations, so retaining lower-order bits may matter equally.

This calls for jointly determining the bit grouping and the compression of each component. If compression is cheap enough, we can evaluate every partition and select the one minimizing total quantization error.

Our baseline uses per-channel INT8 quantized weights. The sign bit is uniformly distributed across weight tensors, so we store it uncompressed in the base. The remaining seven magnitude bits are divided into a four-bit base and a three-bit delta, but which bits fall in each is not fixed: all $\binom{7}{3} = 35$ assignments are evaluated, and the one yielding the lowest total quantization error is selected. We then compress the four-bit base to three bits and the three-bit delta to two. With the sign appended, the base occupies four bits and the delta two, reducing each weight from eight bits to six, a 25% smaller memory footprint. Since a stored base is half the size of an INT8 weight, twice as many bases fit permanently in GPU memory, raising cache hit rates and easing the H2D bottleneck, which is especially valuable at batch size one where full precision can be traded for latency.

Both components use the same compression procedure, so we describe it for the delta, which has fewer possible values and is the simpler illustration.

Reducing the delta width replaces each weight's value with an approximation: the $2^3 = 8$ values representable by the delta bits must be mapped onto $2^2 = 4$ representatives. This is a clustering problem, with a LUT resolving each cluster index to its representative at execution time.

Let $\mathcal{D} = \{a, b, c\}$ denote the bit positions assigned to the delta. These bits realize $n = 8$ distinct values $x_0, \ldots, x_7$, where

$$x_i = m_a^{(i)} 2^a + m_b^{(i)} 2^b + m_c^{(i)} 2^c, \tag{1}$$

and $m_*^{(i)} \in \{0, 1\}$ are the bits of index i. Each x_i occurs with empirically measured probability p_i. The values are partitioned into four clusters $C_0, \ldots, C_3$, and every weight falling in C_j is reconstructed as the centroid $\mu_j = \sum_{i \in C_j} p_i x_i / \sum_{i \in C_j} p_i$, which minimizes within-cluster squared error for a fixed assignment. The assignment itself minimizes

$$\text{MSE} = \sum_{j=0}^{3} \sum_{i \in C_j} p_i (x_i - \mu_j)^2. \tag{2}$$

Because the optimal one-dimensional k-means clustering under squared error is contiguous Fisher (1958), the search reduces to placing three boundaries among the seven gaps between consecutive values, giving $\binom{7}{3} = 35$ candidate clusterings. The base is clustered the same way: its four bits realize 16 values mapped to $2^3 = 8$ representatives, i.e. $\binom{15}{7} = 6435$ candidates. We solve both with the standard $O(n^2 k)$ dynamic program for one-dimensional k-means Wang & Song (2011).

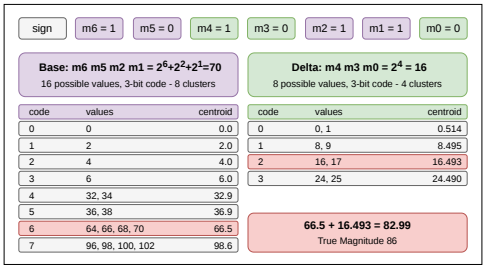

Figure 3: An example of expert weight decomposition and clustering.

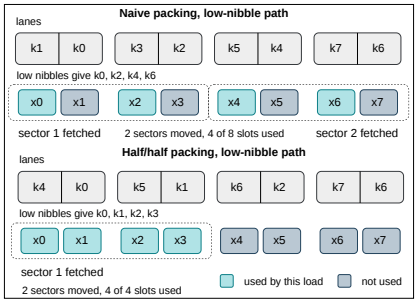

Figure 4: Naive bit packing vs half/half packing. Naive packing wastes most of the fetched sectors.

Code	0	1	2	3
Values	0, 1	8, 9	16, 17	24, 25
Centroid	0.514	8.495	16.493	24.490

(a) Delta clusters, bits $\{0, 3, 4\}$.

Code	0	1	2	3	4	5	6	7
Pattern A	0	2	4	6	32, 34	36, 38	64–70	96–102
Pattern B	0	2	4, 6	32, 34	36, 38	64, 66	68, 70	96–102

(b) Base clusters, bits $\{1, 2, 5, 6\}$.

Table 1: Cluster contents for the $\{0, 3, 4\}$ partition. Ranges denote consecutive even values (64–$70 = 64, 66, 68, 70$).

The bit partition itself is chosen by enumerating all $\binom{7}{3} = 35$ ways to split the seven bits, scoring each by $\mathcal{L} = \mathrm{MSE}_B + \mathrm{MSE}_D$, and keeping the minimizer. $\mathcal{L}$ accounts for both components rather than the base alone, because large-batch serving always uses both planes and needs the full-precision to be accurate. At batch size one, where deltas are fetched only for experts with high routing weights, the accuracy cost of skipping deltas is bounded by separately calibrating the fetch threshold (Section 6.2). Run over every expert in the three models, the search yields a strikingly regular result and almost every tensor selects delta bits $\{0, 3, 4\}$ and base bits $\{1, 2, 5, 6\}$. All exceptions come from Mixtral-8×22B, where 42 of its 1344 expert tensors choose $\{0, 2, 4\}$ instead - too few to justify a second packing strategy and specialized kernels, though it suggests other models may prefer different partitions. Notably, this is not the contiguous split one would choose by inspection: the base interleaves with the delta rather than taking the four most significant bits, since minimizing total MSE favors giving each component both higher- and lower-order bits. Because the partition is constant, we fix it at design time, so bit extraction is identical for every tensor, requiring no per-tensor metadata on the critical path and no kernel divergence. Figure 3 shows an example decomposition from Mixtral-8×7B.

The delta bits $\{0, 3, 4\}$ realize eight values, $\{0, 1, 8, 9, 16, 17, 24, 25\}$, which the optimal four-way clustering pairs as $\{0, 1\}, \{8, 9\}, \{16, 17\}, \{24, 25\}$ across every tensor and model (Table 1a). The clustering thus preserves bits 3 and 4 exactly and averages away bit 0, which contributes least to reconstruction error; that the search recovers this unprompted confirms the objective behaves sensibly.

Base bits $\{1, 2, 5, 6\}$ realize 16 values grouped into eight clusters, and only two boundary patterns occur across the three models (Table 1b). Both isolate the small values 0 and 2, where most of the distribution's mass lies, and both merge 96–102 into one cluster. They differ by a single reallocation: pattern B merges 4 and 6 to afford splitting 64–70 in two, spending a cluster on the upper range instead of the lower. Tensors with more mass at high magnitudes select B; Mixtral-8×7B uses A exclusively, while Mixtral-8×22B uses A for 72% of tensors and Qwen3 for 82.5%. Delta boundaries are invariant across all tensors, but the centroids within them are tightly clustered only for Mixtral-8×7B and Qwen3; Mixtral-8×22B exhibits 38 distinct centroid patterns, none exceeding 12% of tensors.

5 System Implementation

Compression only helps if weights can be reconstructed faster than they can be transferred. This section describes the memory layout of the compressed weights and their reconstruction inside the GEMM kernels. We implement MoE-PACT on MoE-Lightning, a high-throughput engine that

pipelines execution to hide H2D latency, itself built on vLLM. Our host-side code is PyTorch; the fused GEMM kernels are Triton.

Where each part lives. A configurable fraction of the base plane stays resident in GPU memory, with the rest paged in per layer, and the delta plane is pinned on the host. At large batch sizes neither plane depends on routing, so both are streamed a layer ahead and overlapped with the previous layer's compute, leaving no gating decision on the critical path. Base pages and delta slots are double-buffered by layer parity. At batch size one, only the needed experts are prefetched, cached under a pluggable eviction policy (FIFO or cost-aware LCU in our evaluation), and deltas are fetched according to routing weights under a tunable threshold. Per tensor we also keep the two lookup tables (12 floats) and the per-channel INT8 scale, negligible in size but necessarily resident.

Fusing decode into the GEMM. A weight is reconstructed as $\hat{w} = \text{sign} \cdot (\textit{baseTbl}[b] + \textit{deltaTbl}[d]) \cdot \textit{scale}$. Doing this eagerly in separate tensor operations materializes several full-width temporaries per layer and runs unfused on the critical path, so we instead fold the whole operation into the fused MoE kernel: reconstructed weights are produced in registers, consumed by the matrix multiply, and never written to memory. Two further optimizations reduce the inner-loop cost. The per-channel scale moves to the GEMM epilogue, multiplying the FP32 accumulator once after the K-loop so the inner loop runs on unscaled magnitudes in the native compute type. The two table lookups and the sign selection collapse into a single 64-entry table indexed by $(\textit{baseNib} \ll 2) \mid d$, holding the signed reconstructed magnitude for every combination of base index, sign, and delta index; decoding is then one gather from an L1-resident table. Execution splits by phase: decode rebuilds weight tiles from the packed planes on the fly, while prefill reconstructs each page into FP16 once and feeds dense GEMMs, a one-time cost amortized over hundreds or thousands of tokens.

Fixing the bit packing. Even so, the fused GEMM remained roughly $5\times$ slower than the INT8 GEMM. Hardware counters showed it bound on the L1/texture unit at 91% utilization with compute at 24% and DRAM at 2%, and 83% of fetched memory sectors never used. The cause was the obvious nibble packing: consecutive weights along the contraction dimension alternate between the low and high nibble of one byte, so reconstructing a contiguous run required a strided gather, and the activation loads inherited that stride 2 access (Figure 4).

We repack the base plane so that, within each output channel, byte j holds the weight at position j in its low nibble and the weight at position $j + K/2$ in its high nibble, where K is the contraction dimension. One coalesced byte load now feeds two full-width blocks $K/2$ apart, both activation loads are stride-1, and the K-loop issues two full-width matrix multiplies per step instead of two half-width strided ones. The change is purely one of layout: reconstructed values are bit-identical and perplexity is unchanged. It cut the GEMM from 11.4 ms to 6.48 ms at batch 256, enough for compute to hide behind the H2D transfer.

6 Experimental Evaluation

6.1 Experimental Setup

Experiments run in a QEMU virtual machine on an AMD EPYC 9354 host (32 cores) with an NVIDIA H200 NVL (140 GiB HBM3e, PCIe Gen5 $\times$16, driver 580.159.03). The guest has 4 vCPUs (EPYC-v4, with AVX-512 and VAES exposed) and 176 GiB of RAM. CC-mode measurements run in a confidential VM with AMD SEV-SNP and NVIDIA CC mode enabled. We evaluate Mixtral-8$\times$7B, Mixtral-8$\times$22B, and Qwen3-30B-A3B, used in prior offloading work Cao et al. (2024); Liu et al. (2026); Tang et al. (2026), which span Mixtral's few large experts with top-2 routing and Qwen3's 128 small experts with top-8, a far finer streaming granularity (Appendix B).

Baselines. At batch size one, we compare against EARTH Liu et al. (2026) and MoE-APEX Tang et al. (2026), which target the same latency-sensitive, memory- and bandwidth-constrained setting. Neither is open source, so we reimplement both, using EARTH's weight encoding and prefetching but not its accelerator design. At large batch sizes, we compare against per-channel INT8 weights, with both systems on MoE-Lightning's pipeline, isolating our compression and fused reconstruction. Since two of the three models fit in GPU memory, we force offloading and compare at equal cache

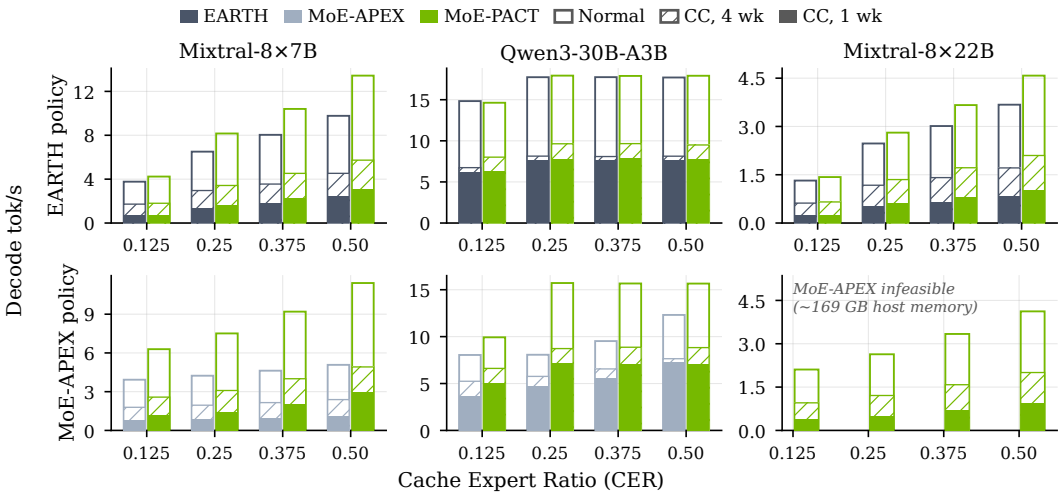

Figure 5: Batch-size-one decode throughput versus cache budget. Bars show normal mode (outline) and CC with four (hatched) and one (solid) H2D processes. MoE-APEX cannot run Mixtral-8×22B: its mixed-precision store needs ∼169 GB of host memory.

budgets, expressed as CER: cached expert bytes over the model's INT8 expert store size. Because MoE-PACT's and EARTH's base planes are half the width of an INT8 weight, the same CER holds twice the fraction of experts as in MoE-APEX and MoE-Lightning.

6.2 Batch-Size-One Serving

We run MoE-PACT under each baseline's caching policy, EARTH's FIFO eviction of bases and MoE-APEX's cost-aware LCU eviction with three-layer lookahead prediction, so differences within each pair isolate the weight encoding. Each system's precision threshold (the routing-weight threshold for delta fetching in EARTH and MoE-PACT, the tier threshold in MoE-APEX) is calibrated to $+0.6\%$ WikiText-2 perplexity over INT8 (Appendix D), a budget chosen because MoE-PACT at full codec precision costs only $+0.30\%$ with unchanged downstream accuracy (Section 6.3).

On the Mixtral models, MoE-PACT beats both baselines at every budget and in every mode (Figure 5). Under the same policy, it is 1.04–$1.38\times$ faster than EARTH and 1.44–$2.74\times$ faster than MoE-APEX on 8×7B, and 1.05–$1.27\times$ faster than EARTH on 8×22B. The gains track transferred bytes: MoE-PACT moves 5–7% fewer bytes per token than EARTH at the smallest budget, 21–26% fewer at the largest, and 30–67% fewer than MoE-APEX, whose dual representation consumes extra memory. The simpler policy performs best: MoE-PACT with FIFO is fastest in 30 of 36 settings, and MoE-APEX's policy helps only at CER 0.125, where MoE-PACT moves 41–43% fewer bytes than EARTH. Since the elaborate policy adds little, comparing encodings under each baseline's own policy is fair. The best MoE-PACT configuration leads the best baseline by 1.16–$1.55\times$ on 8×7B and 1.15–$1.65\times$ on 8×22B, most at the smallest budget, where confidential deployments with little protected memory operate.

Qwen3 is the exception: its experts hold 4.7 M parameters, $37\times$ fewer than Mixtral-8×7B's, so decode is less transfer-bound. Despite moving 11–13% fewer bytes, MoE-PACT is within 3% of EARTH in normal mode and under CC with one process, and leads by 1.17–$1.19\times$ with four. It beats MoE-APEX by up to $1.94\times$, even where MoE-APEX moves fewer bytes, losing only once ($0.97\times$ at CER 0.5, CC, one process).

Encryption slows all systems similarly: with one H2D process, the Mixtral models retain 17–26% of normal-mode throughput and Qwen3 42–58%. Four processes, the saturation point found in Section 3, raise Mixtral-8×7B throughput (MoE-PACT, CER 0.25) from 1.43 to 2.29, 2.80, and 3.09 tok/s, and retention to 41–50% on Mixtral and 45–71% on Qwen3. MoE-PACT's speedups hold (1.05–$1.27\times$ over EARTH and 1.44–$2.06\times$ over MoE-APEX on 8×7B): process parallelism widens the encrypted path while compression shrinks the traffic through it, so the gains compound.

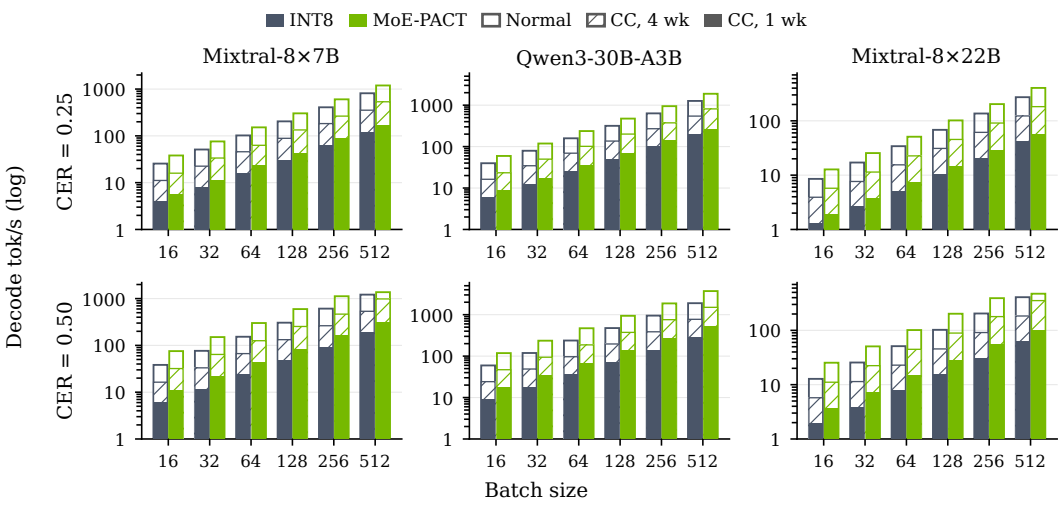

Figure 6: Large-batch decode throughput at equal cache budget. At CER 0.25, INT8 holds $r = 0.25$ of its experts and MoE-PACT $r = 0.50$; at CER 0.5, INT8 holds half and MoE-PACT keeps every base resident, streaming only deltas. Bars as in Figure 5.

6.3 Large-Batch Serving

At large batches, routing touches nearly every expert, so throughput is set by the bytes streamed per decode step. At equal budget MoE-PACT streams one-third fewer bytes than INT8 at CER 0.25 and half as many at CER 0.5, predicting $1.5\times$ and $2\times$ speedups. Under CC, it achieves 1.33–$1.51\times$ and 1.64–$2.01\times$ across all three models and batch sizes (Figure 6). Unlike at batch size one, Qwen3 gains as much as Mixtral, since large batches make even fine-grained experts transfer-bound.

Normal mode shows where this ends. At CER 0.5 with 512 prompts, streaming only deltas leaves computation on the critical path, and MoE-PACT's advantage on the Mixtral models falls to 1.13–$1.16\times$. No such drop appears under CC, where the speedup stays at least $1.64\times$: encryption keeps transfers on the critical path, so the constraint MoE-PACT targets persists exactly where confidential serving operates.

Large batches also expose the full cost of encryption. With one H2D process, both systems retain only 14–16% of normal-mode throughput at CER 0.25, less than at batch size one, because decode is now transfer-bound for every model. Four processes raise throughput 2.6–$3.2\times$, more than at batch size one, lifting retention to 39–46%. Even so, under half of native throughput is recovered, which is why reducing the bytes on the encrypted path remains essential.

Here every expert runs at full codec precision. MoE-PACT raises WikiText-2 perplexity over INT8 by only 0.30% on Mixtral-8×7B and 0.25% on Qwen3, and on Mixtral-8×7B it matches FP16 on GSM8K and stays within noise on TruthfulQA.

7 Conclusion

We presented *MoE-PACT*, an inference system that mitigates the memory and H2D transfer bottlenecks of MoE offloading in both standard and Confidential Computing environments. Exposing VAES and using multiprocess transfers raises CC-mode H2D bandwidth $5.3\times$, to 24 GB/s, yet leaves it under half of normal mode. To reduce the bytes that must cross this encrypted path, MoE-PACT combines base-delta decomposition, DP-optimal 1D k-means clustering, fused GPU kernels that reconstruct weights inside the GEMM, and routing-guided delta fetching. The design cuts the memory footprint by 25% and H2D traffic by up to half with negligible accuracy loss. Under CC, it improves decode throughput by up to $1.65\times$ over the best baseline at batch size one and up to $2.01\times$ over INT8 at large batch sizes. Its benefit tracks how transfer-bound decoding is, and encryption keeps decoding transfer-bound even where normal mode becomes compute-bound, which is precisely the regime confidential serving occupies.

Ethics Statement In this section we note our work's main limitations. First, we target serving bound by expert-weight transfers. Deployments limited by attention or KV-cache capacity might not see the same benefits from our approach. Second, all experiments use a single H200 in pass-through mode with AMD SEV-SNP and NVIDIA CC enabled, so absolute bandwidths are platform-specific, and the bit partition is fixed from three models; other models may prefer different ones. Third, EARTH and MoE-APEX are not open source; our reimplementations use their published caching policies on the same engine but may omit optimizations of the originals. Fourth, at batch size one, routing-dependent delta fetching exposes transfer size and timing to the untrusted host, which may reveal routing decisions. This applies to any on-demand expert fetching, including EARTH and MoE-APEX, and fetching a fixed number of deltas per layer would close it at a throughput cost; at large batch sizes, MoE-PACT prefetches both planes unconditionally. Finally, multiprocess transfers dedicate CPU cores to encryption, which confidential VMs must provision alongside the GPU.

Reproducibility Statement We will release the source code of MoE-PACT upon publication, including the bit-partition search, fused Triton kernels, host-side execution manager, baseline reimplementations, and plotting scripts. The algorithm is specified in Section 4 and Appendix A; both the search and the dynamic program are deterministic, so partitions and codebooks can be reproduced exactly from the public checkpoints (Appendix B). Software versions and the measurement protocol are given in Appendix D, and the microbenchmark in Appendix C. CC-mode results require an H200 with CC enabled in an SEV-SNP confidential VM.

AI Statement Generative AI tools assisted with aggregating experimental data, writing plotting scripts, LaTeX formatting and diagrams, routine software tasks such as file I/O, and copyediting. They were not used in the design or core implementation of our algorithm. The authors reviewed and verified all AI-assisted code, data, and text and take full responsibility for the content of this paper.

REFERENCES

Advanced Micro Devices, Inc. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more. Technical Report 56860, Advanced Micro Devices, Inc., January 2020. URL https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

Emily Apsey, Phil Rogers, Michael O'Connor, and Rob Nertney. Confidential computing on NVIDIA H100 GPUs for secure and trustworthy AI. NVIDIA Technical Blog, August 2023. URL https://developer.nvidia.com/blog/confidential-computing-on-h100-gpus-for-secure-and-trustworthy-ai/.

Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. MoE-Lightning: High-throughput MoE inference on memory-constrained GPUs, 2024. URL https://arxiv.org/abs/2411.11217.

Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel TDX demystified: A top-down approach. *arXiv preprint arXiv:2303.15540*, 2023. URL https://arxiv.org/abs/2303.15540.

Krishna Teja Chitty-Venkata, Sylvia Howland, Golara Azar, Daria Soboleva, Natalia Vassilieva, Siddhisanket Raskar, Murali Emani, and Venkatram Vishwanath. Moe-inference-bench: Performance evaluation of mixture of expert large language and vision models. In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1502–1511, 2025. doi: 10.1145/3731599.3767706.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, et al. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading, 2023. URL https://arxiv.org/abs/2312.17238.

Walter D. Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53:789–798, 1958. URL https://api.semanticscholar.org/CorpusID:120138631.

Elias Frantar and Dan Alistarh. QMoE: Sub-1-bit compression of trillion parameter models. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/c74b624843218d9b6713fcf299d6d5e4-Abstract-Conference.html.

Zhongshu Gu, Enriquillo Valdez, Salman Ahmed, Julian James Stephen, Michael V. Le, Hani Jamjoom, Shixuan Zhao, and Zhiqiang Lin. Blueprint, bootstrap, and bridge: A security look at NVIDIA GPU confidential computing. In *Proceedings of Machine Learning and Systems*, 2026. URL https://arxiv.org/abs/2507.02770v2.

Albert Q. Jiang et al. Mixtral of Experts, 2024. URL https://arxiv.org/abs/2401.04088.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020. URL https://arxiv.org/abs/2006.16668.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

Fangxin Liu, Ning Yang, Jingkui Yang, Zongwu Wang, Chenyang Guan, Yu Feng, Li Jiang, and Haibing Guan. EARTH: An efficient MoE accelerator with entropy-aware speculative prefetch and result reuse. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 633–646, 2026.

Badreddine Noune, Perry Jones, Daniel Chadderton, et al. Fp8 versus int8 for efficient deep learning inference. *arXiv preprint arXiv:2303.17951*, 2023.

NVIDIA. NVIDIA Secure AI with Blackwell and Hopper GPUs. Technical Report WP-12554-001, version 1.3, NVIDIA, August 2025. URL https://docs.nvidia.com/nvidia-secure-ai-with-blackwell-and-hopper-gpus-whitepaper.pdf. GPU pass-through modes: pp. 13–14; authenticated transfer mechanisms: pp. 15–17.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu, 2023. URL https://arxiv.org/abs/2303.06865.

Yifan Tan, Cheng Tan, Zeyu Mi, and Haibo Chen. PipeLLM: Fast and confidential large language model services with speculative pipelined encryption. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2025. doi: 10.1145/3669940.3707224. URL https://arxiv.org/abs/2411.03357.

Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. MoE-APEX: An efficient MoE inference system with adaptive precision expert offloading. In *Proc. ASPLOS*, pp. 1185–1200, 2026. doi: 10.1145/3779212.3790187.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Haizhou Wang and Mingzhou Song. Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29–33, 2011. doi: 10.32614/RJ-2011-015. URL https://journal.r-project.org/articles/RJ-2011-015/.

xAI Team. Open release of grok-1. https://x.ai/news/grok-os, mar 2024.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. MoE-Infinity: Offloading-efficient MoE model serving, 2024. URL https://arxiv.org/abs/2401.14361.

Danyang Zhang, Junhao Song, Ziqian Bi, Xinyuan Song, Yingfang Yuan, Tianyang Wang, Joe Yeong, and Junfeng Hao. Mixture of experts in large language models, 2025. URL https://arxiv.org/abs/2507.11181.

A Bit Partitioning Algorithm

Algorithm 1 gives the offline search of Section 4. It operates on a 128-bin histogram of each tensor's INT8 magnitudes rather than on the tensor itself, enumerates the 35 bit partitions, clusters each component with the one-dimensional k-means dynamic program Wang & Song (2011), and keeps the partition minimizing $\mathcal{L} = \mathrm{MSE}_B + \mathrm{MSE}_D$. Since each component has at most 16 distinct values, every call to CLUSTER is inexpensive, and the search runs once per model.

Algorithm 1 Per-tensor bit partitioning and clustering

Require: *prob*$[x]$, the probability of each INT8 magnitude $x \in \{0, \ldots, 127\}$
Ensure: the base/delta bit split and its lookup tables

minErr $\leftarrow \infty$
for all choices of 3 *deltaBits* out of 7 **do**
  baseBits $\leftarrow$ the remaining 4 bits
  (*deltaErr*, *deltaTbl*, *deltaMap*) $\leftarrow$ CLUSTER(*prob*, *deltaBits*, 4)
  (*baseErr*, *baseTbl*, *baseMap*) $\leftarrow$ CLUSTER(*prob*, *baseBits*, 8)
  if *deltaErr* + *baseErr* $<$ *minErr* **then**
    minErr $\leftarrow$ *deltaErr* + *baseErr*; save this split and its tables
return the saved split and tables

function CLUSTER(*prob*, *bits*, k)
  val$[1..n] \leftarrow$ distinct values of *bits*, sorted; *p*$[i]$ their probabilities
  $D[i][1] \leftarrow \mathrm{cost}(1, i)$ ▷ $\mathrm{cost}(i, j)$: weighted SSE of *val*$[i..j]$
  for $c \leftarrow 2$ **to** k **do**
    for $i \leftarrow c$ **to** n **do**
      $D[i][c] \leftarrow \min_{m<i} \big(D[m][c-1] + \mathrm{cost}(m{+}1, i)\big)$, store $\arg\min$
  backtrack from $D[n][k]$ to get the k cluster boundaries
  tbl$[c] \leftarrow$ probability-weighted mean of cluster c ▷ LUT entry
  map$[v] \leftarrow$ cluster holding v ▷ stored index
  return ($D[n][k]$, *tbl*, *map*)

function ENCODE(w)
  return ($\mathrm{sign}(w)$, *deltaMap*[delta bits of $|w|$], *baseMap*[base bits of $|w|$])
function DECODE(s, d, b)
  return $s \cdot$ (*baseTbl*$[b]$ + *deltaTbl*$[d]$) ▷ per-channel scale applied in the GEMM epilogue

B Model Configurations

Table 2 lists the evaluated models and their expert weight footprints under INT8 and MoE-PACT.

	Mixtral 8×7B	**Mixtral 8×22B**	**Qwen3 30B-A3B**
Layers	32	56	48
Experts / layer	8	8	128
Active experts	2	2	8
Hidden size	4096	6144	2048
Expert interm. size	14336	16384	768
Params / expert	176.2 M	302.0 M	4.7 M
Expert params (total)	45.1 B	135.3 B	29.0 B
MoE-PACT (6-bit)	33.8 GB	101.5 GB	21.7 GB
base plane (4 b)	22.5 GB	67.6 GB	14.5 GB
delta plane (2 b)	11.3 GB	33.8 GB	7.2 GB
Delta streamed / layer	0.35 GB	0.60 GB	0.15 GB

Table 2: Model configurations and expert weight footprints. Each expert holds three tensors (w_1, w_2, w_3); sizes cover expert weights only, and the INT8 store equals the total expert parameter count in bytes. Under MoE-PACT, the base plane is GPU-resident up to the configured fraction; at large batch sizes, the delta plane is streamed from the host once per layer.

C Encrypted H2D Transfers

The microbenchmark of Section 3 transfers data from pinned host memory to GPU memory across payload sizes from 1 MB to 16 GB and reports sustained bandwidth for each size. Host-side profiling uses `py-spy` for Python-level attribution and Linux `perf` for native stack sampling; roughly 90% of profiled execution time falls in internal copy routines in `libcuda.so`, which invoke OpenSSL to encrypt data before writing it to the shared staging buffers.

Saturation. Increasing the transfer process count beyond four produces little additional H2D throughput, including with eight vCPUs assigned to the CVM. Measured separately, CPU-only encryption reaches approximately 80 GB/s across eight cores, so the host retains encryption capacity well beyond the observed 24 GB/s. Sustained throughput also depends on staging-buffer management, driver scheduling, and GPU-side decryption and copying. We therefore report 24 GB/s as the saturation point of our complete transfer configuration; the scaling results alone do not isolate a hardware decryption limit.

D Evaluation Protocol

MoE-PACT runs on Python 3.11, PyTorch 2.1.2 (CUDA 12.1), Triton 2.1.0, and vLLM 0.4.0 kernels. At batch size one, normal-mode throughput is the median of three runs of 64 generated tokens; CC results are single runs of 32 tokens for the Mixtral models and 64 for Qwen3. Large-batch runs use a micro-batch of one, GPU attention, and 8 generated tokens per prompt, with 16 to 512 prompts. H2D bytes per token at batch size one are instrumented in normal mode and stay within 10% of these values under CC; large-batch transfer volumes follow from a byte model of the streamed expert planes.

Accuracy calibration. At batch size one, each system's precision threshold is calibrated so that WikiText-2 perplexity, measured on 16 chunks of 2048 tokens, rises by 0.6% over INT8 (Table 3). EARTH and MoE-PACT threshold the routing weight for delta fetching; MoE-APEX thresholds its precision tier.

	INT8	Target	EARTH	MoE-PACT	MoE-APEX
Mixtral-8×7B	3.9859	4.0098	0.469 / 4.0085	0.250 / 4.0100	0.742 / 4.0093
Mixtral-8×22B	2.9797	2.9976	0.484 / 2.9977	0.219 / 2.9958	n/a
Qwen3-30B-A3B	5.9533	5.9891	0.116 / 5.9895	0.074 / 5.9898	0.915 / 5.9894

Table 3: Batch-size-one operating points: calibrated threshold / resulting WikiText-2 perplexity. MoE-APEX cannot run Mixtral-8×22B, whose dual-representation store needs ≈169 GB of host memory.

E Detailed Results

Tables 4–6 list the measurements behind Figures 5 and 6. Normal denotes CC disabled; CC-1 and CC-4 denote CC with one and four H2D processes. MoE-PACT (EARTH/APEX cache) denotes MoE-PACT run under that baseline's caching policy. In the H2D process sweep (CC, CER 0.25, MoE-PACT with APEX cache), throughput with one to four processes is 1.43, 2.29, 2.80, and 3.09 tok/s on Mixtral-8×7B and 7.16, 7.69, 7.66, and 8.73 tok/s on Qwen3.

		Normal				CC-1				CC-4			
Model	**System**	.125	.25	.375	.5	.125	.25	.375	.5	.125	.25	.375	.5
Mixtral-8×7B	EARTH	3.73	6.46	7.84	9.71	0.68	1.36	1.78	2.38	1.72	2.96	3.55	4.53
	MoE-PACT (EARTH cache)	4.20	7.87	10.20	13.44	0.71	1.61	2.22	3.05	1.80	3.42	4.53	5.73
	MoE-APEX	3.93	4.24	4.60	5.06	0.77	0.84	0.94	1.08	1.79	1.95	2.15	2.39
	MoE-PACT (APEX cache)	6.10	7.45	9.20	11.38	1.15	1.43	2.02	2.96	2.58	3.09	4.00	4.92
Mixtral-8×22B	EARTH	1.30	2.36	3.01	3.65	0.23	0.52	0.63	0.81	0.62	1.17	1.41	1.71
	MoE-PACT (EARTH cache)	1.40	2.78	3.58	4.44	0.24	0.60	0.80	1.02	0.66	1.35	1.72	2.10
	MoE-APEX	infeasible (host store ≈169 GB)											
	MoE-PACT (APEX cache)	2.05	2.54	3.23	4.02	0.38	0.51	0.71	0.95	0.95	1.21	1.58	2.00
Qwen3-30B-A3B	EARTH	14.83	17.52	17.55	17.49	6.19	7.57	7.60	7.58	6.74	8.14	8.09	8.13
	MoE-PACT (EARTH cache)	14.63	17.84	17.75	17.75	6.24	7.73	7.80	7.74	8.01	9.64	9.66	9.50
	MoE-APEX	7.95	8.07	10.60	13.58	3.71	4.71	5.60	7.24	5.24	5.76	6.56	7.65
	MoE-PACT (APEX cache)	9.93	15.63	15.63	15.63	5.02	7.16	7.05	7.04	6.61	8.73	8.87	8.83

Table 4: Batch-size-one decode throughput (tok/s) by cache budget (CER), at the operating points of Table 3.

	Mixtral-8×7B		Mixtral-8×22B		Qwen3-30B-A3B	
System	.125	.5	.125	.5	.125	.5
EARTH	12.41	3.10	36.42	9.11	0.63	0.47
MoE-PACT (EARTH cache)	11.48	2.28	34.54	7.22	0.56	0.41
MoE-APEX	10.16	6.94	n/a	n/a	0.96	0.18
MoE-PACT (APEX cache)	7.09	2.27	21.31	7.15	0.56	0.41

Table 5: Batch-size-one H2D traffic (GB per decoded token) at the smallest and largest cache budgets, instrumented in normal mode; CC values are within 10%.

Model	Mode	16	32	64	128	256	512
CER 0.25: INT8 at $r = 0.25$, MoE-PACT at $r = 0.50$							
	Normal	25.6 / 38.1	51.1 / 76.1	102.2 / 152.0	204.2 / 302.9	408.0 / 603.6	813.7 / 1193.9
Mixtral-8×7B	CC-1	4.0 / 5.8	8.0 / 11.2	16.0 / 23.4	31.0 / 43.7	64.3 / 89.5	122.1 / 166.6
	CC-4	11.1 / 15.9	22.5 / 33.6	45.8 / 62.9	88.7 / 133.9	183.1 / 264.3	353.7 / 535.3
	Normal	8.5 / 12.7	17.1 / 25.5	34.1 / 51.0	68.3 / 101.8	136.4 / 203.2	272.4 / 403.4
Mixtral-8×22B	CC-1	1.3 / 1.9	2.7 / 3.8	5.2 / 7.6	10.4 / 15.0	20.8 / 29.4	42.7 / 56.7
	CC-4	3.9 / 5.7	7.6 / 11.4	15.5 / 22.6	31.1 / 45.4	61.3 / 91.1	123.0 / 182.4
	Normal	39.7 / 59.5	79.4 / 118.7	158.6 / 237.2	317.1 / 474.0	634.1 / 946.0	1265.4 / 1884.5
Qwen3-30B-A3B	CC-1	6.2 / 8.9	12.6 / 17.5	25.2 / 36.3	48.8 / 69.3	100.5 / 143.3	200.2 / 267.6
	CC-4	16.3 / 23.3	34.5 / 49.8	69.3 / 101.4	136.0 / 200.5	270.0 / 374.2	544.6 / 812.9
CER 0.5: INT8 at $r = 0.50$, MoE-PACT base fully resident, delta streamed							
	Normal	38.3 / 75.5	76.5 / 150.5	153.0 / 300.7	305.7 / 597.0	610.5 / 1123.3	1215.7 / 1371.6
Mixtral-8×7B	CC-1	5.9 / 11.0	11.6 / 21.7	24.0 / 43.3	47.7 / 82.6	91.9 / 162.3	184.3 / 307.6
	CC-4	16.4 / 32.0	33.1 / 64.1	66.8 / 125.9	132.6 / 253.8	263.1 / 465.8	535.3 / 983.6
	Normal	12.8 / 25.4	25.6 / 50.7	51.2 / 101.3	102.3 / 202.3	204.4 / 391.4	407.8 / 472.7
Mixtral-8×22B	CC-1	2.0 / 3.6	3.9 / 7.3	7.9 / 14.5	15.6 / 28.2	30.9 / 55.3	61.7 / 101.2
	CC-4	5.7 / 11.1	11.4 / 22.4	22.9 / 44.8	45.5 / 89.1	91.1 / 179.1	183.8 / 352.9
	Normal	59.6 / 118.2	119.0 / 235.8	238.0 / 470.7	475.6 / 940.0	949.9 / 1872.0	1895.0 / 3714.4
Qwen3-30B-A3B	CC-1	8.8 / 17.8	17.7 / 34.1	35.3 / 67.7	70.5 / 134.0	140.6 / 263.2	279.6 / 518.9
	CC-4	24.2 / 47.0	48.8 / 94.1	96.7 / 186.8	196.4 / 374.2	386.7 / 758.4	773.4 / 1505.1

Table 6: Large-batch decode throughput (tok/s), INT8 / MoE-PACT, by number of prompts. Per decode step, MoE-PACT streams $1/1.5$ of INT8's expert bytes at CER 0.25 and only the delta plane, half of INT8's, at CER 0.5, predicting $1.5\times$ and $2\times$ speedups.

Codec accuracy. At large batch sizes, every expert runs at full codec precision. Table 7 reports perplexity on the full WikiText-2 test set and downstream accuracy on Mixtral-8×7B.

		FP16	INT8	MoE-PACT
WikiText-2 ppl	Mixtral-8×7B	3.8394	3.8416	3.8530
WikiText-2 ppl	Qwen3-30B-A3B	6.0870	6.0882	6.1036
GSM8K (100 q, 8-shot)	Mixtral-8×7B	0.510	–	0.510
TruthfulQA MC1 (817 q)	Mixtral-8×7B	0.3195	0.3097	0.3060
TruthfulQA MC2 (817 q)	Mixtral-8×7B	0.4683	0.4662	0.4673

Table 7: Accuracy at full codec precision. MoE-PACT adds 0.30% (Mixtral-8×7B) and 0.25% (Qwen3) perplexity over INT8. Mixtral-8×22B is omitted because its checkpoint has no FP16 reference.