Orion: Online Reduction of Activation Dispatch Overhead in Disaggregated MoE Inference

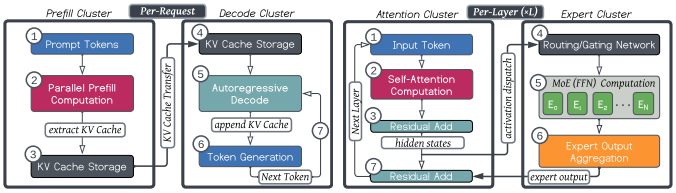
Johnson Umeike **Bahar Asgari**
University of Maryland, College Park
{jumeike, bahar}@umd.edu

***Abstract*—Disaggregated MoE inference separates attention and expert modules onto dedicated clusters, but requires transferring all token activations across the interconnect at every layer, generating hundreds of megabytes of traffic per request that grows with context length and the number of transformer layers. Existing systems transmit activations at full volume, regardless of each token's contribution to expert computation. We present *Orion*, a selective activation compression system built on the observation that not all tokens contribute equally to expert computation: only 20–30% of tokens capture 80% of attention mass across diverse MoE architectures and workloads, while a fixed-budget policy requires retaining as few as 5% of tokens to preserve model accuracy, leaving the majority as candidates for compression. *Orion* uses online attention scores from the forward pass at zero additional cost to identify and selectively compress low-importance token activations before dispatch. Across three MoE models and three datasets, *Orion* reduces inter-node bandwidth by up to 44% with less than 1% perplexity degradation, with attention-score-guided selection outperforming all non-attention baselines, while providing negligible decode latency overhead.**

I. Introduction

Mixture-of-Experts (MoE) architectures have become the dominant paradigm for scaling LLMs to trillion-parameter capacity without proportional increases in inference cost [6], [13], [22], [24], by routing each token to a small subset of expert Feed Forward Networks (FFNs) rather than activating all parameters. Production deployments now routinely serve models with hundreds of billions of parameters [20], placing enormous pressure on inference infrastructure to deliver low latency at high throughput under stringent service-level objectives (SLOs) — typically time-to-first-token (TTFT) $\leq$ 500 ms and time-per-output-token (TPOT) $\leq$ 30–100 ms [27], [35].

Yet MoE's sparse activation introduces a fundamental hardware utilization mismatch during inference. First, compute-intensive prefill and memory-intensive decode compete for the same GPU resources [20], [35]. Second, sparse expert routing leaves GPU compute severely underutilized, as each expert receives far fewer tokens per batch than needed for efficient execution [7], [31]. Two disaggregation strategies have emerged to address these challenges [4], as illustrated in Figure 1. (1) ***PD disaggregation*** [9], [20], [35] decouples the prefill and decode stages onto dedicated machines, eliminating phase interference and transferring the KV cache once per request. (2) ***AE disaggregation*** [16], [34], [36] separates the attention and expert modules onto dedicated GPU clusters within each layer, consolidating tokens across attention replicas to recover expert GPU utilization but dispatches activations at every layer.



(a) *PD disaggregation.* **(b)** *AE disaggregation.*

Fig. 1: ***Two disaggregation paradigms. Prefill-Decode (PD) Disaggregation transfers the KV cache once per request. Attention-Expert (AE) Disaggregation dispatches token activations between clusters at every layer.***

While AE disaggregation helps to recover GPU utilization and to enable scalability, it introduces a new bottleneck: Token activations must traverse the interconnect between attention and expert clusters at every model layer, generating hundreds of megabytes to gigabytes of traffic per request that grows as $\mathcal{O}(C \times L)$, where C is the token context length and L is the number of transformer layers. Existing systems reduce transfer latency through faster communication primitives [34], [36], but treat all token activations uniformly regardless of their contribution to expert computation. This inefficiency is particularly costly in commodity cloud deployments, where bandwidth is limited to 10–50 Gbps [32].

The key to addressing this bottleneck lies in three observations about transformer attention in MoE models. First, token importance within a sequence is highly concentrated — a small fraction of tokens dominate attention output while the majority contribute only minimally [15], [33], and we show this holds consistently across MoE architectures and workloads. Second, activation transfer volume grows as context length and model depth grow, increasing the reduction opportunity when transfer cost is highest. Third, the degree of concentration of token importance varies across workloads, motivating an adaptive rather than a fixed compression policy (more details in § III).

Based on these observations, we present ***Orion***, a selective activation dispatch system for disaggregated MoE inference. *Orion* compresses low-importance token activations using attention scores computed during the attention forward pass at no additional cost, reducing activation transfer volume without the need for offline profiling or accuracy calibration [25]. This

paper makes the following contributions:

- A characterization of token importance concentration across three representative MoE architectures and diverse workloads, revealing that only 20–30% of tokens capture 80% of attention mass, leaving 70–80% as compression candidates, with as few as 5% of tokens sufficient for accurate dispatch under a fixed-budget policy.
- `Orion`, a selective compression framework with two attention-score-guided policies (`Orion-Mass` and `Orion-Rank`) that compress low-importance token activations before dispatch at zero additional scoring cost, reducing inter-node bandwidth by up to 44% with less than 1% perplexity degradation.

II. Background

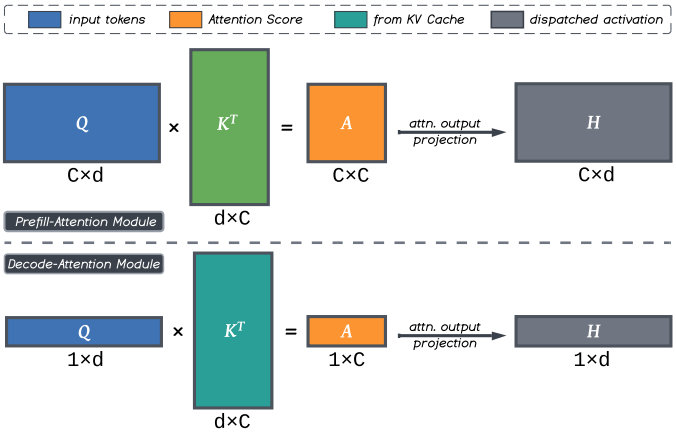
A. Attention-Expert Disaggregation

MoE inference with expert parallelism (EP) distributes expert parameters across GPU devices and routes tokens to remote experts via All2All collectives [11], [22]. While EP enables serving large MoE models, inter-device communication overhead grows with the number of devices and dominates inference latency as cluster size scales [14]. Early systems such as DeepSpeed-MoE [22] and Tutel [11] optimize expert computation and load balancing but treat attention and FFN as co-located components, preventing independent scaling of modules with fundamentally different resource requirements.

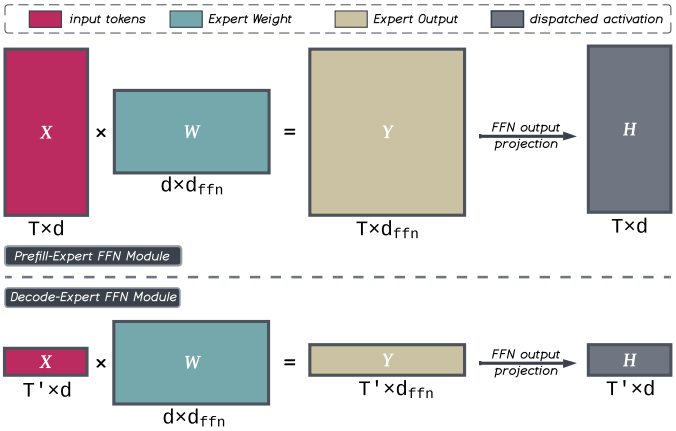
`AE` disaggregation [16], [34], [36] addresses this by placing attention and expert modules on dedicated GPU clusters, as illustrated in Figure 1b. Attention nodes replicate weights and maintain KV caches using data parallelism, while expert nodes distribute expert parameters using EP. For each transformer layer, the attention cluster runs self-attention fully on its local tokens (②), producing post-attention hidden states of size $C \times d$ for a sequence of C tokens and a hidden dimension d (③). A router scores each token (④) and dispatches its hidden states to one or more expert nodes for FFN computation (⑤). Expert outputs are aggregated (⑥) and returned for residual addition (⑦) before the next layer begins, repeating L times per request. To hide transfer latency, systems partition batches into micro-batches and alternate their processing between clusters in a ping-pong fashion [36]. Crucially, KV tensors never leave the attention cluster and only post-attention hidden states cross to expert nodes, while expert computation is stateless across tokens as each token's hidden state is processed independently by its assigned expert [12], [14]. These properties mean that token-level signals from attention computation are available at the exact moment dispatch decisions are made, a structural opportunity that existing systems do not exploit.

B. Attention and Expert FFN Computational Characteristics

Each transformer layer involves two modules in `AE` disaggregation with distinct computational profiles: (1) the ***attention module*** on attention nodes and (2) the ***expert FFN module*** on expert nodes. Figure 2 illustrates the per-request matrix shapes for each phase and module. The attention module computes self-attention across all sequence tokens, as illustrated in



(a) ***Attention module.*** *Prefill (top): $Q \times K^\top$ is a large matrix-matrix multiply (compute-bound). Decode (bottom): $Q \times K^\top$ loads the full KV cache for a single token query (memory-bound).*



(b) ***Expert FFN module.*** *Prefill (top): T tokens per expert yield a compute-bound matrix-matrix multiply. Decode (bottom): $T' = B{\cdot}k/E \ll T$ tokens per expert load the same large weight matrix W (memory-bound), but becomes compute-bound at larger batch sizes.*

Fig. 2: *Matrix shapes for attention and expert FFN modules across prefill and decode phases.*

Figure 2a. During prefill, $Q \times K^\top$ involves matrices of shape $(C \times d)$ and $(d \times C)$ per request, producing a $(C \times C)$ score matrix — a large matrix-matrix multiply with high arithmetic intensity [19], [26], making prefill compute-bound. During decode, each request's query reduces to a single vector of shape $(1 \times d)$ attending to its full KV cache of shape $(d \times C)$, loading the entire cache for a tiny $(1 \times C)$ output with low arithmetic intensity as KV cache scanning degrades locality [26], making decode attention memory-bandwidth-bound. KV tensors are generated and consumed entirely within the attention cluster and never leave it. As shown in Figure 2a, the final per-request post-attention hidden states (obtained via $\text{softmax}(A) \cdot V$ followed by output projection and residual addition) have shape $(C \times d)$ during prefill and $(1 \times d)$ during decode. In `AE` disaggregation, these are aggregated across N_{attn} attention replicas, each handling B concurrent requests, yielding $B_{agg} = B \times N_{attn} \times C$ dispatched tokens during

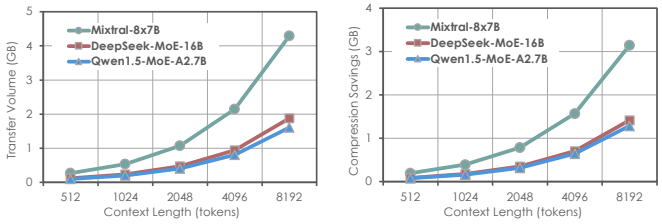
prefill and $B_{agg} = B \times N_{attn}$ during decode, before being routed to expert nodes. Notably, MLA-based architectures [1] reduce KV cache size and shift decode attention toward a compute-bound regime, potentially altering this profile in future deployments.

The expert FFN module applies two linear projections with a nonlinear activation in between, expanding the hidden dimension from d to d_{ffn} before projecting back [11], [22], as illustrated in Figure 2b. During prefill, the expert cluster receives $B_{agg} = B \times N_{attn} \times C$ tokens in total. With top-k routing across E experts, each expert receives $T = B_{agg} \times k/E$ tokens on average (assuming uniform routing). At $B = 8$, $N_{attn} = 4$, and $C = 2048$, top-2 routing across 8 experts yields $T = 16{,}384$ tokens per expert, which results in a matrix-matrix multiply that is compute-bound. During decode, each request generates one new token per step, so the expert cluster receives only $B_{agg} = B \times N_{attn}$ tokens, giving each expert $T' = B_{agg} \times k/E$ tokens. Unlike prefill, $T' \ll T$ is insufficient to amortize loading the full weight matrix W, making decode expert FFN memory-bandwidth-bound [1], [17], [26]. At the same B and N_{attn}, top-2 routing across 8 experts yields $T' = 8$ tokens per expert per step, compared to $T = 16{,}384$ during prefill. However, unlike attention, where the KV cache is request-specific, expert weights are shared across all requests in the expert FFN. AE disaggregation exploits this by accumulating tokens across batches and attention replicas to increase T' and recover compute efficiency [36]. Expert weight matrices reside permanently on expert nodes and never cross the interconnect. Only the input and output hidden states (H in Figure 2) traverse between clusters, making activation transfer the sole inter-cluster data movement cost in AE disaggregation.

III. Motivation

A. The Activation Transfer Bottleneck

Unlike PD disaggregation, which transfers the KV cache once per request [20], [21], [35], AE disaggregation dispatches token activations between attention and expert clusters at every transformer layer [17]. The total transfer volume per request scales as $V = 2 \times d \times p \times C \times L$, where d is hidden dimension, p is bytes per element, C is context length, and L is the number of layers. The factor of 2 accounts for both forward dispatch and return. Figure 3a shows that V grows linearly with context length — at 4K tokens, a single Mixtral-8×7B request generates over 2 GB of activation traffic, reaching 4.3 GB at 8K tokens. Even on 400 Gbps InfiniBand, transferring 4 GB requires over 80 ms per inference step. In commodity cloud deployments with 10–50 Gbps bandwidth [32], activation transfer becomes a dominant inference cost. Existing systems address this through faster transfer infrastructure — custom M2N libraries [36] and two-phase NVLink aggregation [34] — but transmit all token activations uncompressed regardless of their contribution to expert computation. As shown in Figure 3b, compressing low-importance token activations, characterized next in §III-B, could reduce this volume by up to 196 MB, 89 MB, and 81 MB per request at 512 tokens, scaling to 3.1 GB, 1.4 GB, and 1.3 GB at 8K tokens, averaged across evaluated datasets for Mixtral-8×7B, DeepSeek-MoE-16B, and Qwen1.5-MoE-A2.7B respectively, an opportunity no existing AE disaggregation system exploits.



(a) *Activation transfer volume.* **(b)** *Compression opportunity.*

Fig. 3: *Activation transfer volume per request and compression opportunity across MoE models and context lengths.*

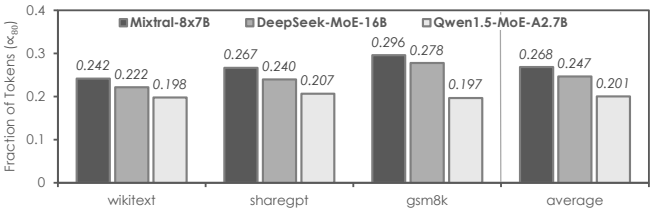


Fig. 4: *α_{80} across MoE models and datasets. Consistently, only 20–30% of tokens capture 80% of attention mass, leaving 70–80% as compression candidates.*

B. Token Importance Concentration in MoE Attention

Prior work shows that a small fraction of tokens dominate attention output in dense models [3], [15], [30], [33], but whether this holds for MoE architectures and what it implies for activation dispatch volume has not been studied. We define α_{80} as the minimum fraction of tokens required to capture 80% of cumulative attention mass, averaged across layers. While token importance varies per layer [3], we report the layer-averaged metric here to characterize the aggregate compression opportunity across the full forward pass. This coverage-based metric is inspired by the heavy-hitter observation in H2O [33] but adapted to quantify the compression opportunity rather than set a fixed eviction budget.

We profile three MoE models across Wikitext-2, ShareGPT, and GSM8K using 200 samples per configuration; α_{80} shows low variance across samples (CV = 0.04–0.12), confirming that 200 samples is sufficient to obtain stable estimates. Figure 4 reveals three findings. First, α_{80} ranges from 0.197 (Qwen, Wikitext-2) to 0.296 (Mixtral, GSM8K), confirming that only 20–30% of tokens capture 80% of attention mass across all MoE architectures and workloads. Second, α_{80} is stable across sequence lengths, meaning that the compressible fraction does not diminish at longer contexts where the transfer cost is highest. Third, α_{80} varies across workloads, with GSM8K yielding up to 0.054 higher α_{80} than Wikitext-2 for the same model, suggesting that an adaptive rather than a fixed compression policy is needed. Together, these findings show

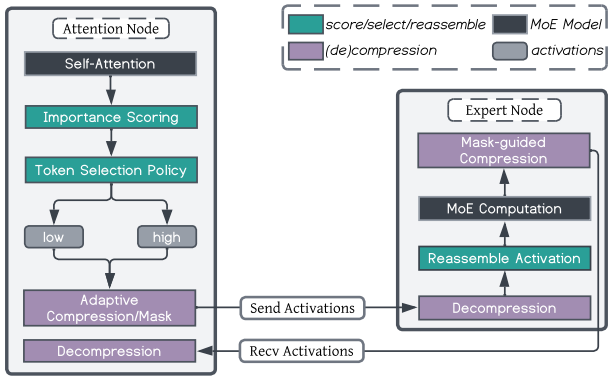


Fig. 5: `Orion` *Overview.*

that on average $1-\alpha_{80}$ of token activations per request are low-importance candidates for compression, ranging from 73.2% for Mixtral-8×7B to 79.9% for Qwen1.5-MoE-A2.7B averaged across datasets, directly motivating selective activation compression in `AE` disaggregation.

IV. ORION: Design and Implementation

To effectively address these challenges, we present `Orion`, which uses token-level importance signals to selectively compress low-importance token activations before dispatch, transmitting a compact binary importance mask alongside compressed activations to enable symmetric compression on the return path without recomputing importance on the expert side. Figure 5 illustrates an overview of `Orion`'s design.

On the attention node, `Orion` intercepts post-attention hidden states before dispatch. An importance scorer derives per-token scores from attention weights already computed during self-attention (§IV-A), adding no additional computation. A token selection policy classifies tokens as high- or low-importance; high-importance tokens are retained at full precision while low-importance tokens are compressed. A binary mask is generated and transmitted alongside the activations. On the expert node, received activations are decompressed and reassembled to their original sequence positions before MoE routing and FFN computation [36]. After expert computation, the same mask guides return path compression, eliminating importance recomputation on the expert side. The attention node decompresses the returned activations before residual addition, completing the layer. This repeats at every transformer layer.

A. Attention-Score-Guided Token Classification

1) ***Importance Scoring****:* `Orion` derives a per-token importance score s_i^l at each transformer layer l as the mean attention weight received from all other tokens across all heads:

$$s_i^l = \frac{1}{H}\sum_{h=1}^{H}\sum_{j=1}^{C} A_{h,j,i}^l \tag{1}$$

where $A_{h,j,i}^l$ is the attention weight from token j to token i in head h, H is the number of attention heads, and C is the context length. This column-sum formulation measures how much each token is attended to by others. Tokens with high cumulative attention carry contextual information critical to expert computation. Since attention weights are already computed during the self-attention forward pass, importance scoring adds no additional computation. Scores are computed independently at each layer since token importance varies across layers [3], allowing the selection policy to adapt its token budget per layer rather than applying a global mask.

2) ***Token Selection Policy****:* Given per-token scores $\mathbf{s}^l$, `Orion` supports two threshold policies for classifying tokens as important or low-importance. `Orion-Mass` selects the minimum number of tokens whose cumulative normalized attention mass meets a coverage target α. This adapts the token budget per sequence and per layer, providing a fidelity guarantee that at least α fraction of attention mass is preserved. `Orion-Rank` keeps a fixed top-k fraction of tokens by attention score at every layer, providing a uniform floor regardless of per-layer concentration. As we show in §V-B, `Orion-Mass`' adaptive budget can under-serve layers with highly concentrated attention at aggressive thresholds, compressing layers to as few as 1–2% of tokens, while `Orion-Rank`'s fixed floor prevents this. They use the same importance score and only differ in the threshold mechanism.

Algorithm 1 `Orion` per-layer selective compression

Require: Hidden states $\mathbf{h} \in \mathbb{R}^{C\times d}$, attn. weights $\mathbf{A} \in \mathbb{R}^{H\times C\times C}$
Ensure: Compressed dispatch $(\hat{\mathbf{h}}, \mathbf{m}, \boldsymbol{\sigma})$

// Attention node
$\mathbf{s} \leftarrow \frac{1}{H}\sum_h \sum_j A_{h,j,:}$ ▷ importance score per token
$\mathbf{r} \leftarrow \text{rank}(\mathbf{s}, \text{descending})$ ▷ rank tokens by importance
if `Orion-Mass` **then**
  $n^* \leftarrow \min\left\{n : \sum_{\mathbf{r}_{:n}} \tilde{s}_i \geq \alpha\right\}, \tilde{\mathbf{s}} = \mathbf{s}/\|\mathbf{s}\|_1$
else
  $n^* \leftarrow \lfloor k \cdot C \rfloor$ ▷ `Orion-Rank`
end if
$\mathbf{m} \leftarrow \mathbf{0}^C;\ \mathbf{m}[\mathbf{r}_{:n^*}] \leftarrow 1$ ▷ binary importance mask
$\hat{\mathbf{h}}[\mathbf{m}{=}1] \leftarrow \mathbf{h}[\mathbf{m}{=}1]$ ▷ important: keep BF16
$\boldsymbol{\sigma}_t \leftarrow \max(|\mathbf{h}_t|)/127$ ▷ per-token INT8 scale, $\forall t : m_t{=}0$
$\hat{\mathbf{h}}[\mathbf{m}{=}0] \leftarrow \lfloor \mathbf{h}[\mathbf{m}{=}0]/\boldsymbol{\sigma} \rceil_{\text{INT8}}$ ▷ lowimp: INT8
Transmit $(\hat{\mathbf{h}}, \mathbf{m}, \boldsymbol{\sigma})$ to expert node
// Expert node
$\mathbf{h}'[\mathbf{m}{=}0] \leftarrow \hat{\mathbf{h}}[\mathbf{m}{=}0] \cdot \boldsymbol{\sigma}$ ▷ dequantize INT8 → BF16
$\mathbf{h}'[\mathbf{m}{=}1] \leftarrow \hat{\mathbf{h}}[\mathbf{m}{=}1]$ ▷ reassemble full sequence
$\mathbf{h}_{\text{out}} \leftarrow \text{MoE}(\mathbf{h}')$ ▷ route + expert FFN
Compress $\mathbf{h}_{\text{out}}$ using $\mathbf{m}$; transmit to attention node
// Attention node
$\mathbf{h}_{\text{next}} \leftarrow \mathbf{h} + \text{Decompress}(\mathbf{h}_{\text{out}})$ ▷ residual add

B. Selective Compression and Dispatch

Algorithm 1 details `Orion`'s per-layer compression procedure. Given the binary importance mask $\mathbf{m} \in \{0,1\}^C$ from the token selection policy, high-importance tokens ($m_i = 1$) are transmitted at full BF16 precision while low-importance tokens ($m_i = 0$) are quantized to INT8 using a per-token scale $\sigma_t = \max(|\mathbf{h}_t|)/127$. The per-token scale ensures each token's dynamic range is preserved independently, avoiding the accuracy loss associated with global quantization scales.

TABLE I: *Evaluated MoE model configurations.*

Model	Top-k	Experts	d	Layers (L)
Mixtral-8×7B	2	8	4096	32
Qwen1.5-MoE-A2.7B	4	60	2048	24
DeepSeek-MoE-16B	6	64	2048	28

The compressed activations, scales, and mask are dispatched together to the expert node. The total dispatch size per layer is $n_{\text{imp}} \cdot d \cdot 2 + n_{\text{low}} \cdot (d + 4)$ bytes, where the second term accounts for INT8 values plus a 4-byte float32 scale per low-importance token. The binary mask adds $\lceil C/8 \rceil$ bytes, negligible relative to the activation tensor of $C \cdot d \cdot 2$ bytes. While this illustration uses INT8 quantization, `Orion` is designed to support different compression policies to achieve higher compression ratios.

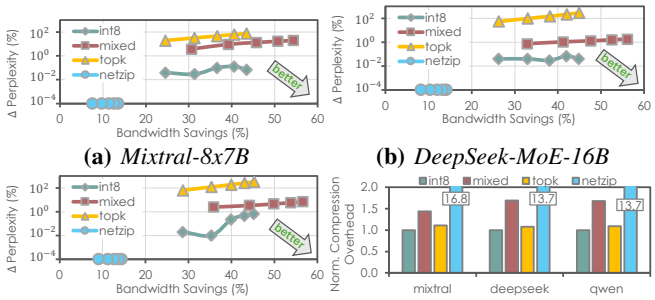
V. Methodology

To evaluate `Orion`, we set out to answer three questions: **Q1:** *How much bandwidth can* `Orion` *reduce while preserving model accuracy, and what is the compression overhead?* **Q2:** *Does attention-score-guided selection outperform alternative token selection policies?* **Q3:** *Does* `Orion`*'s per-layer transfer reduction justify its compression overhead?*

We evaluate `Orion` on three MoE models across three datasets representing distinct workload types: Wikitext-2 (document text), ShareGPT (conversational), and GSM8K (reasoning). Table I summarizes model configurations. We emulate `AE` disaggregation by intercepting post-attention hidden states at each MoE layer during a standard forward pass, applying compression and decompression in-place, and measuring perplexity degradation and bandwidth reduction while accounting for per-token scales. This yields exact measurements equivalent to a full distributed deployment, as (de)compression operates directly on real model activations. All models are evaluated at native BF16 precision on NVIDIA RTX A6000 GPUs. Transfer times are modeled on InfiniBand HDR (IB-HDR) [18], providing 25 GB/s unidirectional bandwidth, representative of commodity cloud inter-node interconnects [32].

A. Bandwidth Reduction vs. Accuracy Trade-off

We evaluate four alternative compression schemes applied under `Orion-Mass`'s token selection policy (§ IV-A) to isolate the effect of the compression scheme. *INT8* quantizes each low-importance token activation to 8-bit integers using a per-token scale factor to preserve dynamic range. *Mixed precision* applies three-tier compression by importance rank: the most important tokens are kept at full FP16 precision, while the remaining tokens are split evenly into INT8 and INT4 tiers by descending importance. *Top-K sparse* [2] retains only 25% of activation dimensions for low-importance tokens in sparse format. *NetZip* [10] losslessly compresses the upper bytes of low-importance token activations using LZ4 [5] by exploiting the structured nature of sign and exponent bits in BF16 representations to incur zero accuracy loss.



(a) *Mixtral-8x7B* **(b)** *DeepSeek-MoE-16B* **(c)** *Qwen1.5-MoE-A2.7B* **(d)** *Compression overhead*

Fig. 6: *(a)–(c) Bandwidth savings vs. ΔPPL% across coverage levels (70–90%). Lower and further right is better. (d) Compression overhead normalized to INT8 at coverage=80%.*

Figures 6a–6c show the Pareto trade-off across five coverage levels (70%, 75%, 80%, 85%, 90%) across all three models, where each point on a curve corresponds to one coverage level. `Orion` achieves 25–44% bandwidth savings while keeping ΔPPL below 1% across all models and datasets. Mixed precision saves up to 57% bandwidth but degrades perplexity by up to 31% on Mixtral. Top-K sparse matches INT8 bandwidth savings but incurs 20–440% perplexity degradation, confirming that dropping dimensions is far more destructive than quantizing them. NetZip is lossless (ΔPPL≈0) but saves only 8–15% bandwidth. The Pareto ranking is consistent across all three models, with DeepSeek and Qwen showing slightly lower absolute ΔPPL than Mixtral due to their smaller hidden dimensions. In Figure 6c, Qwen shows higher INT8 sensitivity, likely because its smaller active parameter count leaves each token activation carrying more unique information, thus amplifying the quantization noise.

Figure 6d shows the compression compute overhead normalized to INT8 quantization, averaged across datasets for each model. Mixed precision cost 1.44–1.69× more compared to INT8 quantization due to the additional INT4 tier. Top-K sparse adds only 1.08–1.11× overhead but is disqualified by its accuracy cost. Without hardware acceleration [10], NetZip is prohibitive at 13.7–16.8× despite being lossless.

> ***Takeaway:*** `Orion` *with INT8 compression reduces activation bandwidth by up to 44% with less than 1% perplexity degradation across all models and datasets, while also incurring the lowest compression overhead.*

B. Attention-Score-Guided Token Selection

We compare five token selection policies under INT8 compression, sweeping `Orion-Mass` over coverage $\alpha \in \{0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$ and all fixed-budget policies over $k \in \{0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35\}$. `Orion-Mass` selects the minimum number of tokens whose cumulative attention mass meets α, adapting the budget per layer. `Orion-Rank` keeps a fixed top-k fraction of tokens based on the attention column-sum score. *StreamingLLM* [30] retains four attention sink tokens plus the most recent k tokens.

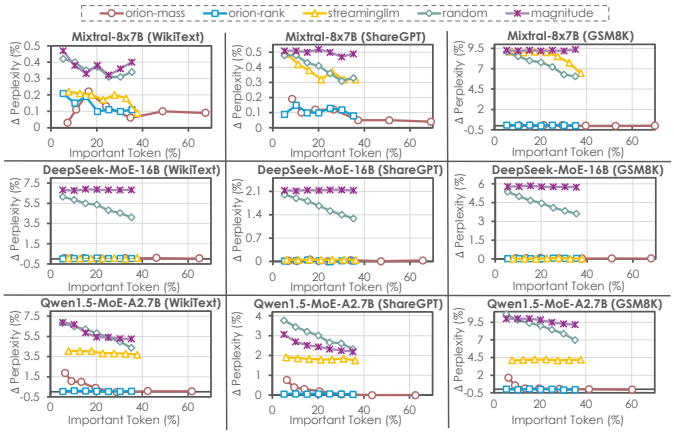
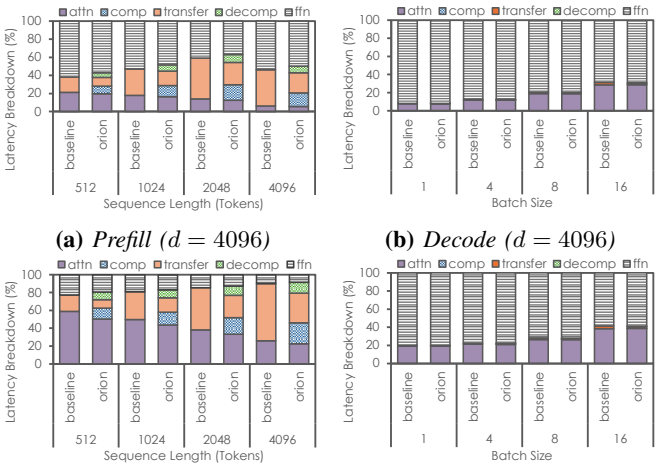


Fig. 7: *ΔPPL% vs. important token fraction (%) across five policies, three models, and three datasets. All policies use INT8 compression.*

Random selects k tokens uniformly at random. *Magnitude* selects the top-k tokens by their hidden-state L2 norm.

Figure 7 reveals consistent trends across all nine configurations. Both attention-guided policies dominate non-attention baselines because they concentrate the token budget on positions that most influence expert computation, as measured by attention column-sum scores. `Orion-Rank` achieves less than 0.15% ΔPPL at just 5% token retention across all models and datasets. At low token fractions, `Orion-Mass` incurs higher degradation because its coverage target can compress individual layers to very few tokens when attention is highly concentrated, while `Orion-Rank`'s fixed-k floor prevents this. `Orion-Mass` is expected to outperform `Orion-Rank` at longer sequences, where attention concentration decreases, and an adaptive per-layer budget better captures the true importance distribution than a fixed-k floor. StreamingLLM [30] performs within 0.1% ΔPPL of attention-guided policies on DeepSeek across all workloads and on Mixtral/Wikitext, where conversational and document tokens tend to be locally concentrated. However, it degrades PPL by 4.05% on Qwen, even for Wikitext, and severely on GSM8K across all models by 9.07% because reasoning tokens are distributed throughout the sequence rather than concentrated at recent positions, breaking StreamingLLM's recency assumption. Random and Magnitude degrade consistently up to 10.62% and 10.00% on Qwen/GSM8K across all workloads, confirming that neither activation magnitude nor random selection captures the token importance signal that attention scores provide.

> ***Takeaway:*** *Attention-score-guided selection achieves less than 0.15% perplexity degradation at 5% token retention, while non-attention policies degrade by up to 10% on reasoning tasks, confirming that attention scores are the critical signal for accurate token selection.*



(a) *Prefill* ($d = 4096$) **(b)** *Decode* ($d = 4096$)

(c) *Prefill* ($d = 2048$) **(d)** *Decode* ($d = 2048$)

Fig. 8: *Per-layer latency breakdown for the uncompressed baseline and* `Orion` *on IB-HDR (25 GB/s) at* $k = 20\%$*. Prefill sweeps context length* $C \in \{512, 1024, 2048, 4096\}$*, while decode sweeps batch size* $B \in \{1, 4, 8, 16\}$*.*

C. Compression Overhead vs. Transfer Savings

Figure 8 shows the per-layer latency breakdown across the prefill and decode phases, quantifying how `Orion`'s transfer savings compare to the added (de)compression overhead. We use `Orion-Rank` at $k = 20\%$ to provide a fixed per-layer token budget, with transfer times computed analytically for IB-HDR [18] at 25 GB/s using actual compressed byte counts. In prefill (Figures 8a, 8c), transfer grows from 17% to 40% of the uncompressed baseline layer time as C increases from 512 to 4096 tokens for $d = 4096$, and from 18% to 64% for $d = 2048$, confirming the $\mathcal{O}(C \times L)$ transfer bottleneck characterized in §III. `Orion` reduces this fraction while incurring some (de)compression overhead. When $d = 4096$, FFN dominates, so (de)compression remains a small fraction. When $d = 2048$, FFN is cheaper, making (de)compression more visible. Since transfer savings grow with C while (de)compression cost scales only with the number of low-importance tokens, `Orion` becomes increasingly favorable at longer contexts.

In decode, depicted in Figures 8b, 8d, only B new tokens are dispatched per step, making transfer less than 3% of layer time at $B = 16$ — leaving no meaningful per-step latency to optimize. `Orion`'s value in decode is instead aggregate: reducing per-token IB traffic by up to 40% across all L layers per generated token. At high request concurrency, this directly increases the number of requests the interconnect can sustain before saturation, improving throughput without violating TPOT SLOs [35]. The current (de)compression overhead is due to unoptimized kernels, and a fused CUDA quantization implementation [29] would reduce this substantially.

> ***Takeaway:*** `Orion` *reduces per-token IB traffic by up to 40% at* $k = 20\%$ *with negligible decode latency overhead, while prefill transfer savings grow with context length and (de)compression overhead remains bounded.*

VI. Future Work

Several directions extend this work. ***First***, a novel compression method co-designed with token importance, exploiting the structured sparsity and score-magnitude correlation of low-importance activations, could achieve higher compression ratios with lower accuracy cost [8], [29]. ***Second***, hardware support for importance-guided compression, such as a near-network accelerator or attention-score-aware NIC, would eliminate GPU-side overhead [23], [28]. ***Third***, a full end-to-end `Orion` implementation within an existing AE disaggregation system [34], [36], including pipeline scheduling and an open-source simulation framework, would enable rigorous end-to-end evaluation and benefit the broader research community.

VII. Conclusion

We presented `Orion`, a selective activation compression system for disaggregated MoE inference. By exploiting attention scores already computed during the forward pass, `Orion` identifies and compresses low-importance token activations before dispatch, reducing inter-node bandwidth by up to 44% with less than 1% perplexity degradation across three MoE architectures and diverse workloads. Attention-score-guided selection consistently outperforms non-attention baselines, with `Orion-Rank`, which uses a fixed top-k by attention score, achieving less than 0.15% perplexity degradation at just 5% token retention. In decode, `Orion` reduces per-token IB traffic with negligible latency overhead, directly improving serving throughput under SLO constraints.

References

[1] V. Adhinarayanan and N. Jayasena, "The qs inequality: Quantifying the double penalty of mixture-of-experts at inference," *arXiv preprint arXiv:2603.08960*, 2026.

[2] S. Bian, D. Li, H. Wang, E. P. Xing, and S. Venkataraman, "Does compressing activations help model parallel training?" *Proceedings of Machine Learning and Systems*, vol. 6, pp. 239–252, 2024.

[3] Z. Cai, Y. Zhang, B. Qi *et al.*, "PyramidKV: Dynamic KV cache compression based on pyramidal information funneling," in *arXiv preprint arXiv:2406.02069*, 2024.

[4] J. Chen, Y. Zhuang, and H. Zhang, "Disaggregated inference: 18 months later," https://haoailab.com/blogs/distserve-retro/, 2025.

[5] Y. Collet, "LZ4: Extremely fast compression algorithm," https://lz4.github.io/lz4/, 2011.

[6] D. Dai, C. Deng, C. Zhao, R. X. Xu *et al.*, "DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models," *arXiv preprint arXiv:2401.06066*, 2024.

[7] Y. Feng *et al.*, "MoE-Lens: Towards the hardware limit of high-throughput MoE LLM serving under resource constraints," in *arXiv preprint arXiv:2504.09345*, 2025.

[8] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *International conference on machine learning*. PMLR, 2023, pp. 10 323–10 337.

[9] C. Hu, H. Huang, L. Xu, X. Chen, J. Xu, S. Chen, H. Feng, C. Wang, S. Wang, Y. Bao *et al.*, "Inference without interference: Disaggregate llm inference for mixed downstream workloads," *arXiv preprint arXiv:2401.11181*, 2024.

[10] J. Huang, H. Kim, N. Wang, J. Kang, H. Shah, E. K. Lee, M. Zhang, F. Lai, and N. S. Kim, "Netzip: Algorithm/hardware co-design of in-network lossless compression for distributed large model training," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 609–625. [Online]. Available: https://doi.org/10.1145/3725843.3756079

[11] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram, J. Chau, P. Cheng, F. Yang, M. Yang, and Y. Xiong, "Tutel: Adaptive mixture-of-experts at scale," in *MLSys*, 2023.

[12] R. Hwang, J. Wei, S. Cao, C. Hwang, X. Tang, T. Cao, and M. Yang, "Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference," in *ISCA*, 2024.

[13] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[14] Y. Li *et al.*, "Semantic parallelism: Redefining efficient MoE inference via model-data co-scheduling," in *arXiv preprint arXiv:2503.04398*, 2025.

[15] Y. Li, Y. Huang, B. Yang, B. Venkitesh *et al.*, "SnapKV: LLM knows what you are looking for before generation," *arXiv preprint arXiv:2404.14469*, 2024.

[16] B. Lin, C. Zhang, T. Peng, H. Zhao, W. Xiao, M. Sun, A. Liu, Z. Zhang, L. Li, X. Qiu *et al.*, "Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache," *arXiv preprint arXiv:2401.02669*, 2024.

[17] G. Liu, H. Li, Y. Guo, Y. Lyu, M. Zhou, Y. Liu, Z. Li, and Y. Wang, "Revealing the challenges of attention-ffn disaggregation for modern moe models and hardware systems," *arXiv preprint arXiv:2602.09721*, 2026.

[18] NVIDIA, "Introducing 200G HDR InfiniBand solutions," NVIDIA, Tech. Rep., 2020. [Online]. Available: https://network.nvidia.com/files/doc-2020/wp-introducing-200g-hdr-infiniband-solutions.pdf

[19] S. J. Park *et al.*, "Exploiting intel advanced matrix extensions (AMX) for large language model inference," in *IEEE ISPASS*, 2024.

[20] P. Patel, E. Choukse, C. Zhang, Í. Goiri, A. Shah, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative LLM inference using phase splitting," in *51st ACM/IEEE Annual International Symposium on Computer Architecture (ISCA 24)*, 2024, pp. 118–132.

[21] R. Qin, Z. Li, W. He, J. Cui, H. Tang, F. Ren, T. Ma, S. Cai, Y. Zhang, M. Zhang, Y. Wu, W. Zheng, and X. Xu, "Mooncake: A KVCache-centric disaggregated architecture for LLM serving," *ACM Transactions on Storage*, 2024, best Paper Award, FAST 2025.

[22] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International conference on machine learning*. PMLR, 2022, pp. 18 332–18 346.

[23] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtárik, "Scaling distributed machine learning with {In-Network} aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 785–808.

[24] Q. Team, "Qwen1.5-MoE: Matching 7B model performance with $\frac{1}{3}$ activated parameters," *Qwen Blog*, 2024.

[25] J. Umeike and B. Asgari, "Xami: Expert-aware adaptive compression for mixture-of-experts on sparse tensor cores," *IEEE Computer Architecture Letters*, 2025.

[26] H. Wang, X. Xiao, M. Yan, Z. Zhu, D. Han, D. Wang, W. Li, X. Ye, C. Hu, H. Chen *et al.*, "A systematic characterization of llm inference on gpus," *arXiv preprint arXiv:2512.01644*, 2025.

[27] Y. Wang, Z. Wang, H. Hu, Y. Chen, and T. Yang, "Towards efficient and reliable LLM serving: A real-world workload study," *arXiv preprint arXiv:2401.17644*, 2024.

[28] X. Xiang, R. Joshi, Y. Liu, J. Yao, C. Zhao, J. Jiang, Y. Zhou, E. Kohler, and M. Yu, "Shadowserve: Interference-free kv cache fetching for distributed prefix caching," *arXiv preprint arXiv:2509.16857*, 2025.

[29] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International conference on machine learning*. PMLR, 2023, pp. 38 087–38 099.

[30] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *International Conference on Learning Representations*, vol. 2024, 2024, pp. 21 875–21 895.

[31] T. Zhang, Z. Feng, and Z. Wang, "MoE-Gen: High-throughput MoE inference on a single GPU with module-based batching," in *arXiv preprint arXiv:2503.09716*, 2025.

[32] Z. Zhang, H. Shen, S. Vargaftik, R. B. Basat, M. Mitzenmacher, and M. Yu, "Hack: Homomorphic acceleration via compression of the key-value cache for disaggregated llm inference," in *Proceedings of the ACM SIGCOMM 2025 Conference*, 2025, pp. 1245–1247.

[33] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Jia, R. Cai, Z. Song, Y. Tian, C. Re, C. Barrett, Z. Wang, and B. Chen, "H2O: Heavy-hitter oracle for efficient generative inference of large language models," 2023.

[34] Z. Zhang, Y. Wang, X. Wang, Y. Zhao *et al.*, "Janus: Disaggregating attention and experts for scalable MoE inference," in *arXiv preprint arXiv:2512.13525*, 2025.

[35] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, 2024, pp. 193–210.

[36] R. Zhu, Z. Jiang, C. Jin, P. Wu, C. A. Stuardo *et al.*, "MegaScale-Infer: Serving mixture-of-experts at scale with disaggregated expert parallelism," in *ACM SIGCOMM*, 2025.