# ASYNCEP: INTRA-LAYER EXPERT ASYNCHRONY FOR MOE PREFILL WITH IMBALANCED ATTENTION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Mixture-of-experts (MoE) serving commonly deploys data and expert parallelism (DEP): attention replicas run distinct request batches while routed experts are sharded across an expert-parallel (EP) group. During prefill, attention replicas finish dispatch at different times, but synchronous EP delays expert feed-forward network (FFN) computation until routed inputs from all replicas are ready. Request schedulers seek to balance load while reusing the key-value (KV) cache of shared prompt prefixes to avoid redundant prefill computation. These goals can conflict when a replica holding a matching prefix is already overloaded, leaving residual attention imbalance. We present ASYNCEP, a distributed execution engine for MoE prefill. ASYNCEP proposes three mechanisms. Asynchronous EP allows expert computation to start before tokens from all attention replicas are ready. *streamFFN* batches ready tokens to balance early execution with FFN computation efficiency. Opportunistic expert weight fetching (*OEWF*) allows a faster replica to fetch expert weights and execute unstarted work from other GPUs. We evaluate ASYNCEP on DeepSeek-V4-Flash, DeepSeek-V4-Pro, and GLM-5.3, and our results show that ASYNCEP achieves up to 1.48× speedup in p95 time-to-first-token (TTFT) and improves the inference throughput by up to 1.17×.

## 1 INTRODUCTION

Mixture-of-experts (MoE) has become a mainstream architecture for scaling large language models, as exemplified by DeepSeek-V4 and Kimi K3 (DeepSeek, 2026; Kimi Team, 2026), which expand parameter capacity while activating only a subset of experts per token. For large-scale MoE serving, a widely adopted deployment strategy is data and expert parallelism (DEP): attention uses data parallelism (DP), while routed experts use expert parallelism (EP) (vLLM, 2026). Attention replicas process distinct request batches and maintain the associated key-value (KV) caches, while routed experts are distributed across GPUs within an EP group. Token dispatch transfers token activations—the inputs to routed experts—to the GPUs hosting the selected experts for feed-forward network (FFN) computation. Token combine returns expert outputs to the originating attention replicas and aggregates them (DeepEP, 2025). Batching routed tokens from multiple attention replicas can increase per-expert batch sizes and improve expert GEMM efficiency (DeepSeek, 2025). This deployment can also integrate other parallelism techniques, such as tensor parallelism (TP) within each attention replica (Zhu et al., 2025).

**DP attention imbalance in MoE prefill.** During MoE prefill, attention replicas process different numbers of requests with varying context lengths, causing them to complete attention computation and reach EP dispatch at different times (DeepSeek, 2025; Zhu et al., 2025). Existing DEP implementations typically use synchronous EP, in which each GPU starts FFN computation only after all attention replicas have finished dispatching their tokens (DeepEP, 2025; MoonEP, 2026). Consequently, replicas that finish attention early remain idle before FFN computation while waiting for other replicas to reach dispatch, as illustrated in Figure 1(a). In our DeepSeek-V4-Flash (DeepSeek, 2026) prefill experiment on the AgentX dataset (SemiAnalysis, 2026) with DP8+EP8 in SGLang (Zheng et al., 2024), replicas spend an average of 28.86% of end-to-end runtime idle while waiting for other replicas to reach dispatch (Appendix A).

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.

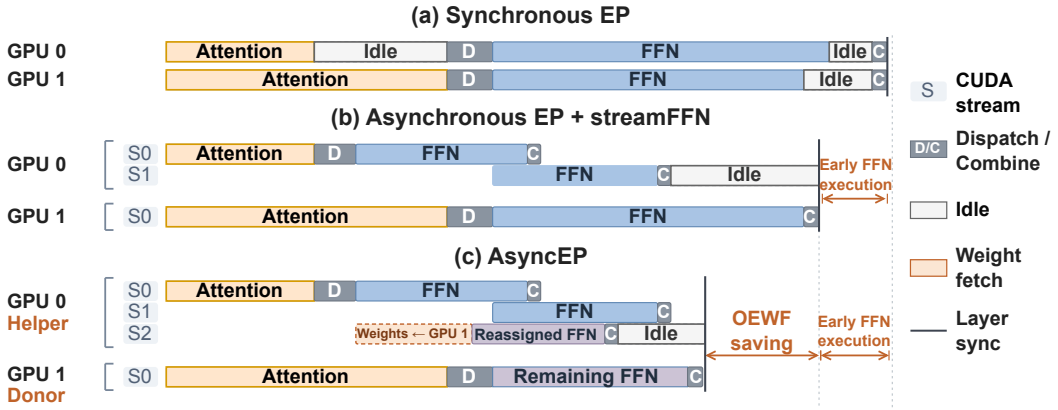



Figure 1: Illustrative MoE prefill timelines for two representative GPU ranks. (a) Synchronous EP synchronizes all replicas before starting FFN computation. (b) Asynchronous EP with *streamFFN* overlaps independent FFN GEMM kernels on multiple CUDA streams. (c) AsyncEP further allows GPU 0 to fetch weights and execute reassigned FFN work on a dedicated helper stream.

**Limits of request scheduling.** Existing request scheduling policies balance workloads across serving instances while reusing KV caches for matched request prefixes (Srivatsa et al., 2025; Yuan et al., 2026; Zhang et al., 2026), which can also be adapted to DEP's attention replicas. However, request scheduling policies alone cannot eliminate the imbalance in DP attention workloads for two reasons. First, cache reuse constrains request scheduling: a replica holding a matched prefix may be overloaded, whereas choosing a less loaded replica without the matched prefix may require KV recomputation or additional transfer (Qin et al., 2025). This creates a trade-off between cache-hit rate and attention load balance (ALB), as shown in Figure 2. Second, even setting aside cache-locality constraints, attention-stage execution time depends on input sequence lengths and batch composition (Zhong et al., 2024; Zhu et al., 2025). Unpredictable online arrivals make it difficult to consistently balance attention-stage execution times across replicas (see Appendix B).

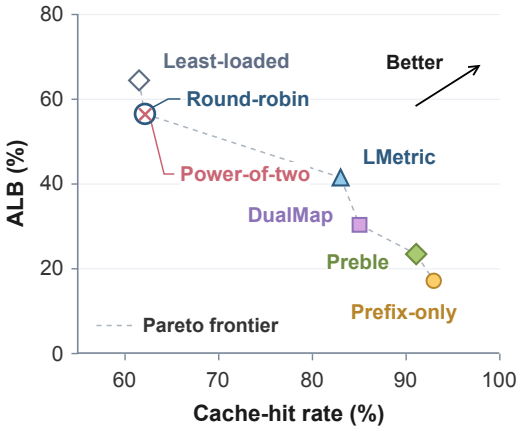



Figure 2: Cache reuse vs. ALB for state-of-the-art scheduling policies, evaluated with SGLang serving DeepSeek-V4-Flash during prefill on the AgentX dataset. ALB is computed for each layer as the mean attention-stage duration across DP replicas divided by the maximum.

**Design principles: asynchronous EP with expert weight fetching.** To address DP attention imbalance that persists under state-of-the-art request scheduling policies, we propose AsyncEP, a distributed execution engine for MoE prefill that combines asynchronous expert execution with expert weight fetching. AsyncEP allows FFN GEMM kernels to be launched before all routed expert tokens have arrived and to return output asynchronously (Figure 1(b)). AsyncEP also enables replicas that have completed dispatch to fetch expert weights from those still computing attention and take over part of their pending FFN work. This reduces the remaining FFN workload on replicas with longer attention times, helping mitigate the layer-wise straggler effect (Figure 1(c)).

**Challenge 1: preserving FFN computation efficiency.** Under synchronous EP, each GPU waits for tokens from all attention replicas before launching FFN GEMM kernels. Asynchronous EP allows FFN to start on ready expert tokens but splits a complete GEMM kernel into smaller ones, reducing batch sizes and potentially degrading computation efficiency. To preserve FFN computation efficiency, we introduce *streamFFN*, guided by a key observation: as the number of expert tokens increases, effective FFN throughput generally rises rapidly at first, then improves more gradually

before approaching a plateau. Accordingly, *streamFFN* accumulates ready expert tokens until their count reaches a launch threshold. Meanwhile, *streamFFN* schedules independent FFN GEMM kernels for different input groups on multiple CUDA streams, allowing concurrent kernels to use GPU resources left idle by partially occupied final thread-block waves (NVIDIA, 2023; 2026a).

**Challenge 2: utilizing idle GPU time.** Although asynchronous EP allows replicas that finish attention early to complete their FFNs sooner, the next attention layer cannot begin until all replicas have received their required expert outputs. This layer-wise barrier leaves earlier-finishing replicas idle while waiting for the slowest. To exploit this idle time, we introduce *Opportunistic Expert Weight Fetching* (*OEWF*), which enables replicas that have completed token dispatch to asynchronously fetch expert weights from those still computing attention and take over part of their unstarted FFN work. *OEWF* first uses a coordinated greedy algorithm to select the source replicas and expert weights to fetch. Once the weights are ready, a cost model compares the completion times of the current and revised FFN execution plans to determine whether to reassign FFN work.

**Results.** We evaluate AsyncEP on DeepSeek-V4-Flash, DeepSeek-V4-Pro (DeepSeek, 2026), and GLM-5.3 (Z.ai, 2026) using Tool&Agent (Qin et al., 2025) and AgentX under five request-scheduling policies. End-to-end experiments show that AsyncEP achieves 1.13× speedup in p95 time-to-first-token (TTFT) on average (up to 1.48×) and improves inference throughput by 1.06× on average (up to 1.17×).

Our contributions in this paper are as follows:

- We characterize DP attention imbalance in MoE prefill and show how synchronous EP converts uneven attention completion times into GPU idle time.
- We propose AsyncEP, a distributed MoE prefill engine that combines asynchronous EP, *streamFFN*, and *OEWF* to execute ready tokens efficiently and reassign FFN work to idle GPUs.
- We comprehensively evaluate AsyncEP across three models, two real-world workloads, and five request-placement policies. Our experiments show AsyncEP achieves significant end-to-end improvement compared with baselines.

## 2 Related Work

**EP communication optimizations.** In large-scale DEP serving, dispatch and combine incur substantial communication overhead from large data volumes and uneven traffic across GPUs. Existing works mitigate this overhead through two complementary approaches. *Communication–computation overlap* pipelines token transfers with expert computation: DeepEP provides specialized dispatch and combine kernels with asynchronous interfaces (DeepEP, 2025), while Comet schedules communication and GEMM execution at tile granularity within fused GPU kernels (Zhang et al., 2025). *Fused MoE execution* reduces kernel-launch and orchestration overhead: FlashMoE integrates dispatch, expert computation, and combine into a single persistent GPU kernel, pipelining them via device-initiated communication and task scheduling (Aimuyo et al., 2025). These operator optimizations, together with expert placement and load balancing (Li et al., 2023; Yang et al., 2026), target communication and expert execution within the MoE layer. They are largely orthogonal to our focus on synchronization stalls caused by DP attention imbalance.

**Request scheduling for DP load balancing.** Round-robin assigns requests cyclically across instances (SGLang, n.d.), while prefix-only routing selects the instance with the longest matching cached prefix (Srivatsa et al., 2025). Power-of-two-choices selects the less loaded of two sampled instances (Mitzenmacher, 2001), whereas least-loaded routing selects the least loaded overall (Zhang et al., 2026). LMetric minimizes the product of the cache-aware queued prefill-token count after assignment and the current batch size (Zhang et al., 2026). Preble's global E2 scheduler combines prefix reuse with load-aware exploration based on computation and KV-cache eviction costs (Srivatsa et al., 2025). DualMap uses independent prefix hashes to identify two candidates, then applies SLO-aware routing and hotspot rebalancing (Yuan et al., 2026).

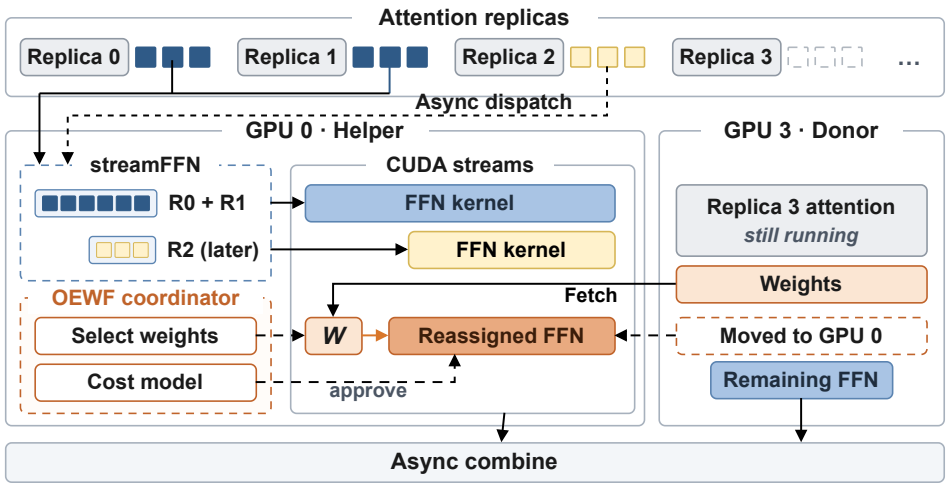



Figure 3: AsyncEP system overview.

# 3 AsyncEP Design

## 3.1 Overview

AsyncEP is a distributed execution engine for MoE prefill. Figure 3 shows its main components: asynchronous dispatch and combine for communication, *streamFFN* (§3.2) for FFN GEMM kernel batching and scheduling, and opportunistic expert weight fetching (*OEWF*, §3.3) for fetching expert weights and reassigning FFN work. Each replica dispatches tokens independently after completing its attention stage, and *streamFFN* batches ready tokens for FFN execution. A replica that has completed token dispatch can act as a *helper*. For each helper, the coordinator selects a replica still computing attention as its *donor* and determines which expert weights to fetch. After the helper finishes fetching the selected experts' full weights, the coordinator uses a cost model to decide whether to reassign the corresponding FFN work from the donor to the helper.

## 3.2 Asynchronous EP and streaming FFN

AsyncEP enables FFN to begin before all replicas complete token dispatch. Our extension to DeepEP (DeepEP, 2025) exposes per-replica completion signals at each destination, indicating when an attention replica's routed expert tokens have arrived and are ready for computation. These tokens become eligible for *streamFFN* scheduling without waiting for other replicas. Appendix C details our implementation in SGLang (Zheng et al., 2024).

However, launching a separate FFN GEMM kernel whenever a replica's routed tokens become ready can fragment the workload into small per-expert batches and reduce computation efficiency. *streamFFN* balances early execution with FFN efficiency through two mechanisms: threshold-based batching to form larger FFN GEMM kernels from ready expert tokens, and multi-stream execution to overlap independent kernels and utilize otherwise idle GPU resources.

**Launch threshold.** Our key observation is that FFN throughput rises rapidly at low expert-token counts but gradually saturates as the token count increases (Figure 4). This motivates starting FFN computation at near-peak efficiency without waiting for all tokens to arrive. Accordingly, *streamFFN* sets the launch threshold $\theta$ to the smallest profiled expert-token count that achieves at least a target fraction $\rho$ of the observed peak throughput. A larger $\rho$ favors computational efficiency, whereas a smaller $\rho$ enables earlier launches. The threshold is calibrated for each model–GPU configuration to account for expert shapes, compute precision, and kernel tiling. Once all tokens have arrived, *streamFFN* bypasses the threshold and processes the remaining expert tokens.

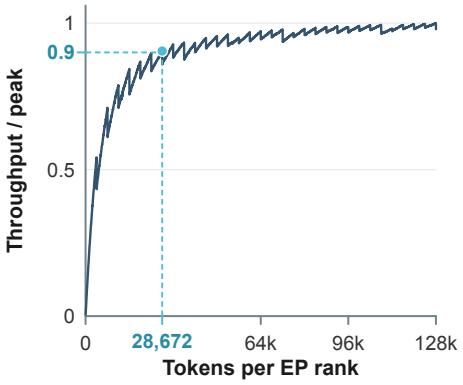



Figure 4: FFN throughput of DeepSeek-V4-Flash with EP8 using DeepGEMM on B200, normalized to its observed peak.

**Algorithm 1** Greedy expert-weight selection

**procedure** SELECTWEIGHTS($h$)
  $\mathcal{D} \leftarrow$ CANDIDATEDONORS($S, h$)
  **if** $\mathcal{D} = \varnothing$ **then return**
  $d^* \leftarrow \arg\max_{d \in \mathcal{D}} T_d(S)$
  $\mathcal{L} \leftarrow \mathrm{sort}^{\downarrow}_{e \in \mathcal{E}_{d^*}} \left( |\mathcal{T}_{hd^*e}| / b_e \right)$
  $\mathcal{E}_h \leftarrow \varnothing$; $R \leftarrow B_h$
  **for** $e \in \mathcal{L}$ **do**
    **if** $b_e \leq R$ **then**
      $\mathcal{E}_h \leftarrow \mathcal{E}_h \cup \{e\}$; $R \leftarrow R - b_e$
    **else**
      **break**
    **end if**
  **end for**
  **if** $\mathcal{E}_h \neq \varnothing$ **then**
    $P_h \leftarrow (h, d^*, \mathcal{E}_h, \{\mathcal{T}_{hd^*e}\}_{e \in \mathcal{E}_h})$
    RECORDFETCH($P_h$)
    ASYNCFETCH($P_h$)
  **end if**
**end procedure**

**Multi-stream execution.** A launch threshold improves FFN computation efficiency but does not eliminate *wave quantization*, also known as the *tail effect* (NVIDIA, 2023). The number of blocks in a full wave equals the SM count multiplied by the kernel's resident-block capacity per SM. When a kernel's remaining block count is not a multiple of this capacity, its final wave is only partially occupied. *streamFFN* schedules independent FFN GEMM kernels for different input groups on separate CUDA streams, allowing eligible kernels to use GPU resources left idle by another kernel's final wave. Our experiments show that *streamFFN* incurs about a 2–3% loss in effective FFN throughput relative to synchronous execution, while substantially advancing FFN completion through earlier launches, as detailed in §4.3.

### 3.3 OPPORTUNISTIC EXPERT WEIGHT FETCHING

Although asynchronous EP allows replicas that finish attention early to complete their FFN computation sooner, the next attention layer cannot begin until all replicas have received their required expert outputs. This layer-wise barrier leaves earlier-finishing replicas idle while waiting for the slowest. *OEWF* exploits this idle GPU time to help slower replicas execute their unstarted FFN work. We call a replica that has completed its token dispatch a *helper*, while a slower replica still computing attention is a candidate *donor*, from which the helper may fetch expert weights. Each replica will make two decisions: a coordinated greedy algorithm selects a donor and the expert weights to fetch, and once the weights are ready, a cost model decides whether the helper should take over the corresponding FFN work.

**Coordinated greedy expert-weight selection.** Starting from the second MoE layer, each helper makes one fetch decision immediately after dispatch. Candidate donors are replicas still computing attention, and $\mathcal{D}$ denotes this candidate set.

We first estimate the remaining attention time. For donor replica $d \in \mathcal{D}$ in the current MoE layer, let $I_d$ and $P_d$ denote its current input lengths and matched prefix lengths, and let $t_d^{\mathrm{att}}$ be the elapsed attention time. The remaining attention-stage time is estimated as

$$A_d = \left[ f(I_d, P_d) - t_d^{\mathrm{att}} \right]. \tag{1}$$

where $f$ predicts the complete attention-stage duration from the input lengths (Appendix B).

Second, we model dispatch and FFN GEMM kernel latencies. Let $D_d$ denote the dispatch latency for rank $d$ in the current layer, which is calculated by dividing the dispatched-token count, determined from the input lengths, by the profiled effective dispatch throughput. We estimate the received expert-token counts $N_d$ from the preceding layer's per-rank token count and model FFN latency as

$$F(N_d) = \frac{N_d}{\eta(N_d) P_{\mathrm{peak}}} \tag{2}$$

where $\eta(N_d) \in (0, 1]$ is the normalized effective FFN throughput and $P_{\text{peak}}$ is the corresponding profile's observed peak throughput.

Finally, we combine these estimates under the current execution plan $S$, and the predicted completion time of donor replica $d$ is

$$T_d(S) = A_d + D_d + F(N_d) \tag{3}$$

The coordinator selects the donor with the latest completion time:

$$d^* = \arg\max_{d \in \mathcal{D}} T_d(S) \tag{4}$$

For the selected donor $d^*$, let $\mathcal{T}_{hd^*e}$ denote the set of tokens dispatched by helper $h$ to expert $e$, and let $b_e$ be the number of bytes required to fetch that expert's weights. Experts in the eligible set $\mathcal{E}_{d^*} = \{e : \mathcal{T}_{hd^*e} \neq \varnothing\}$ are sorted in descending order of $|\mathcal{T}_{hd^*e}|/b_e$. The per-helper byte budget $B_h$ is capped by the available capacity of the helper's temporary weight buffer. Starting with $R = B_h$, the coordinator adds each expert to the selected set $\mathcal{E}_h$ if $b_e \leq R$, then reduces $R$ by $b_e$, until adding the next expert would exceed the capacity limit. The coordinator records each fetch plan $P_h$ and its resource usage to update the FFN execution plan, keeping expert selection consistent with a shared global view across all replicas. Finally, the weight transfers are performed asynchronously without interfering with the helper's computation. Algorithm 1 details the procedure.

**Cost-based FFN reassignment.** After helper $h$ finishes fetching the selected weights, the coordinator compares the current plan $S$ with a revised plan $S'$ that transfers $N_{trans}$ unstarted expert tokens from donor $d$ to the helper $h$. The transferred work executes in a separate FFN GEMM kernel on another CUDA stream and can overlap with the helper's remaining FFN work. The completion times of the revised plan are

$$\begin{aligned} T_d(S') &= A_d + D_d + F(N_d - N_{\text{trans}}) \\ T_h(S') &= T_h(S) + \alpha_h F(N_{\text{trans}}) \end{aligned} \tag{5}$$

Here, $\alpha_h \in [0, 1]$ is the fraction of the reassigned FFN latency that extends beyond that time. The coordinator assigns $\alpha_h$ from the candidate CUDA-stream schedule under the current plan $S$: $\alpha_h = 1$ corresponds to serial execution, while $\alpha_h = 0$ means that the reassigned work is fully hidden within the helper's remaining execution. The revised plan is adopted only if

$$\max\{T_d(S'), T_h(S')\} \leq \max\{T_d(S), T_h(S)\}. \tag{6}$$

Under our cost-model and execution assumptions, we prove that applying *OEWF*'s policy across all replicas guarantees $M_{\text{OEWF}} \leq M_{\text{no-fetch}}$, where $M$ denotes layer completion time (Appendix E). Its practical effectiveness depends on these predictions. The complete attention-stage duration and the received expert-token count have accuracies of 90.3% and 85.9%, respectively (Appendix D). Section 4.3 further evaluates its layer-level benefits in real model deployment.

# 4 Evaluation Results

## 4.1 Evaluation Settings

**Workloads.** We evaluate two workloads separately. Mooncake's Tool&Agent trace captures production tool and agent requests characterized by long, repeated system prompts (Qin et al., 2025). The AgentX trace corpus contains multi-turn coding-agent sessions, including both main-session and subagent requests (SemiAnalysis, 2026). For each workload, we scale the original interarrival times to match the target request rate. For each model and target rate, all baselines replay the same request trace under the same KV-cache budget. Figure 5 shows the input-length distributions.

**Models and environments.** We evaluate AsyncEP on DeepSeek-V4-Flash, DeepSeek-V4-Pro (DeepSeek, 2026), and GLM-5.3 (Z.ai, 2026). For all three models, we use prefill–decode (PD) disaggregation across two nodes and evaluate only the prefill instance. The prefill node contains eight NVIDIA B200 GPUs interconnected via NVLink, with eight-way data parallelism for attention and eight-way expert parallelism for routed experts, placing one attention replica and one EP rank on each GPU. Attention uses FP8 for all three models; routed-expert weights use MXFP4 for DeepSeek-V4-Flash and DeepSeek-V4-Pro, and FP8 for GLM-5.3. We set *streamFFN*'s throughput target to $\rho = 0.90$ for all three models. Table 1 summarizes the model architectures.

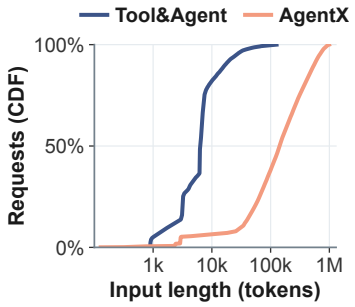



Figure 5: Input-length CDFs of two datasets.

| | DeepSeek-V4-Flash | DeepSeek-V4-Pro | GLM-5.3 |
|---|---|---|---|
| **Total / active parameters** | 284B / 13B | 1.6T / 49B | 753B / ∼40B |
| **Attention** | CSA and HCA | CSA and HCA | DSA |
| **Routed experts / layer** | 256 | 384 | 256 |
| **Shared experts / layer** | 1 | 1 | 1 |
| **Routed experts / token** | 6 | 6 | 8 |
| **Expert intermediate dim.** | 2,048 | 3,072 | 2,048 |

Table 1: Model architecture details. GLM-5.3 parameter counts follow NVIDIA (2026b).

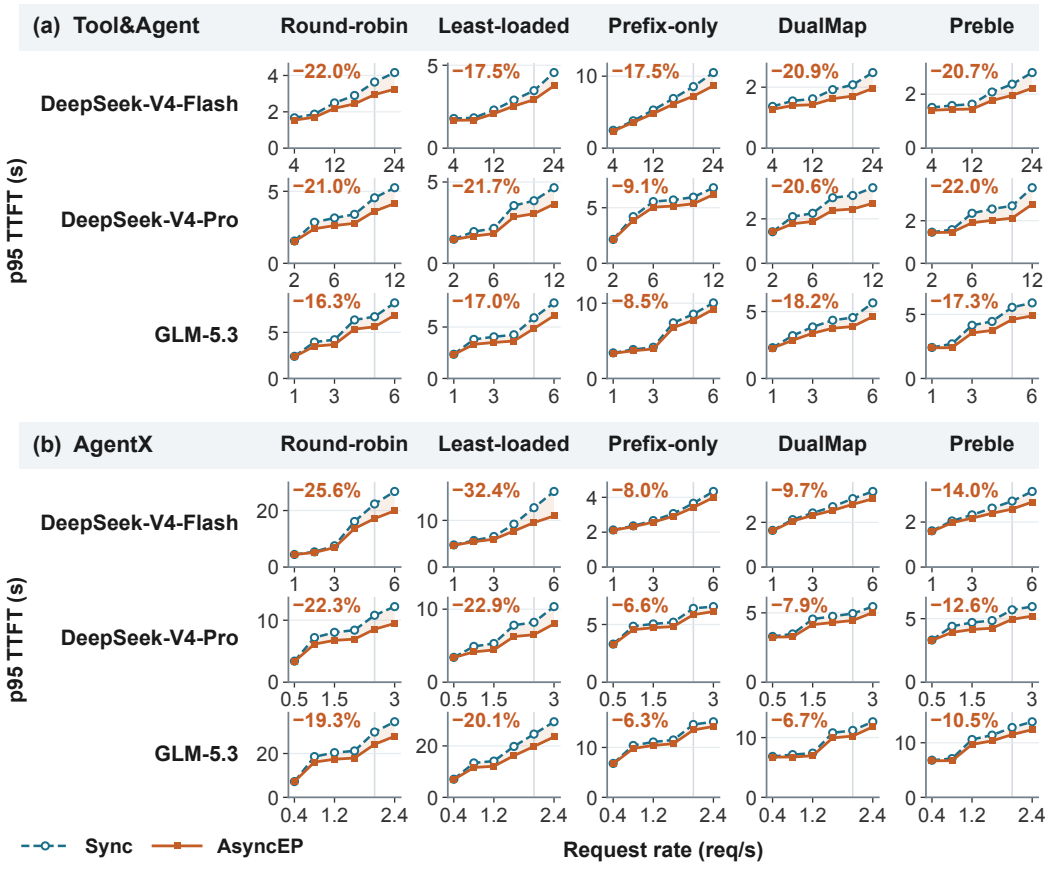



Figure 6: Prefill p95 TTFT across models, workloads, and request scheduling policies. Percentages denote reductions at the highest request rates.

**Baseline and placement policies.** Our baseline, SYNC, uses synchronous SGLang DEP deployment with DeepEP for dispatch and combine (DeepEP, 2025) and DeepGEMM for FFN GEMM computation. FFN starts after dispatch completes on all replicas (Figure 1(a)). We compare SYNC and ASYNCEP under five request-scheduling policies: round-robin (SGLang default), least-loaded, prefix-only, DualMap, and Preble (§2) (SGLang, n.d.; Zhang et al., 2026; Srivatsa et al., 2025; Yuan et al., 2026). We implement these policies in the SGLang router to assign requests across DP attention replicas within the prefill instance. For Preble, we use a prefill-oriented adaptation of its global E2 policy. We use round-robin placement for the ablation experiments.

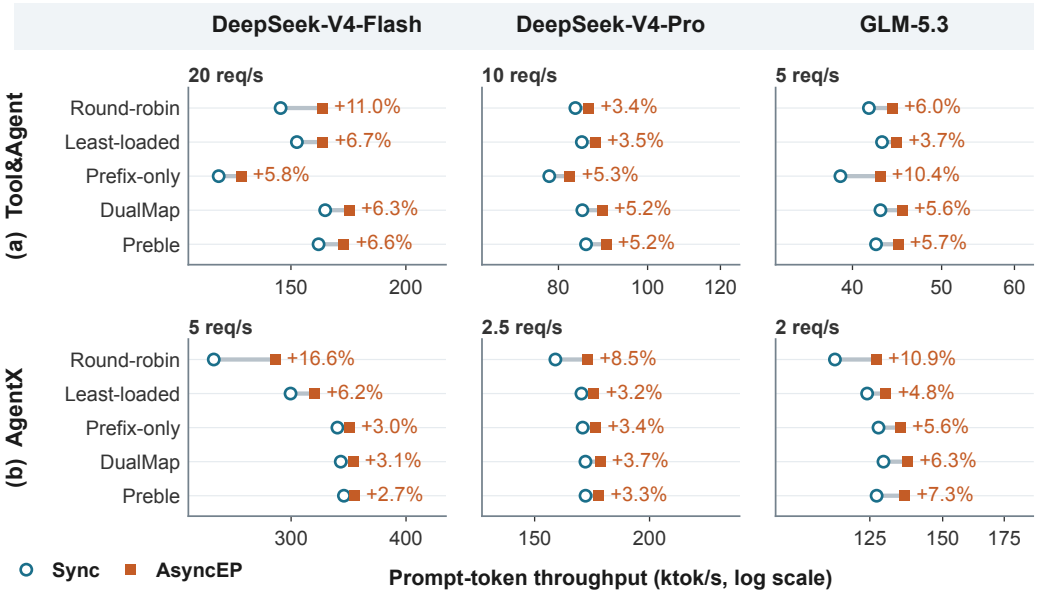



Figure 7: Prompt-token throughput across models and workloads.

**Metrics.** We report p95 time-to-first-token (TTFT) across request rates. We also report prompt-token throughput at selected high-load request rates near saturation. Each point in the end-to-end and component-ablation results is the arithmetic mean of the corresponding per-run metric over five independent runs.

### 4.2 End-to-end prefill performance

Figure 6 compares the p95 TTFT of Sync and AsyncEP across request rates for three models, two workloads, and five request-scheduling policies. At the highest request rate in each panel, AsyncEP achieves an average p95 TTFT speedup of 1.21× and a maximum of 1.48× over Sync. The largest reduction, 32.4%, occurs on DeepSeek-V4-Flash with AgentX under least-loaded scheduling at 6 req/s. On AgentX, reductions at the highest tested rates average 22.4% and 25.1% across the three models under round-robin and least-loaded scheduling, respectively, compared with 7.0%, 8.1%, and 12.4% under prefix-only, DualMap, and Preble. Although the gains are smaller under the latter policies, the improvements under DualMap and Preble demonstrate that intra-layer execution optimization complements cache- and load-aware request scheduling.

Figure 7 compares prompt-token throughput at selected high-load request rates. AsyncEP improves throughput across all three models, both workloads, and all five policies, with an average increase of 6.0% and a maximum of 16.6% (1.06× and 1.17× the baseline throughput, respectively). The largest gain occurs on DeepSeek-V4-Flash with AgentX under round-robin scheduling at 5 req/s. Averaged across workloads and policies, throughput increases by 6.8%, 4.5%, and 6.6% for DeepSeek-V4-Flash, DeepSeek-V4-Pro, and GLM-5.3, respectively, showing consistent throughput improvements alongside the TTFT reductions.

### 4.3 Deep Dive

**Ablation study.** We compare four variants on DeepSeek-V4-Flash using AgentX at 5 req/s under round-robin placement, keeping all other settings fixed. Sync uses synchronous DEP execution. Async executes each source replica's ready expert inputs in a separate FFN GEMM kernel on a single compute stream, without a launch threshold. Async+SF adds *streamFFN*'s launch threshold and multi-stream execution. AsyncEP further enables *OEWF* to fetch expert weights and reassign eligible unstarted FFN work.

Figure 8 shows the incremental benefits of these mechanisms. ASYNC starts FFN computation earlier than SYNC, but small FFN GEMM kernels limit its gains. Adding *streamFFN* improves computation efficiency through threshold-based batching and multi-stream execution and yields the largest incremental TTFT reduction. The matched-work comparison below examines this efficiency improvement separately. On top of ASYNC+SF, *OEWF* uses idle GPU time to execute reassigned, unstarted FFN work, further reducing average and p95 TTFT by 7.8% and 6.2%, respectively.

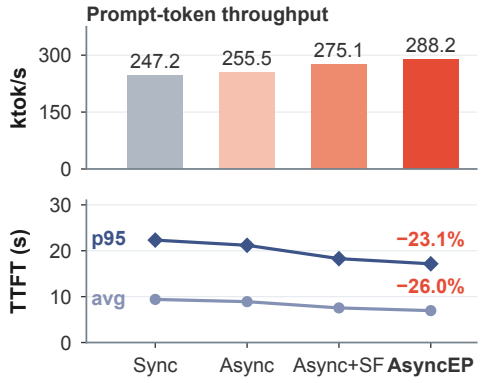



Figure 8: Incremental ablation of ASYNCEP. Top: prompt-token throughput. Bottom: average and p95 TTFT, annotated with the reduction of ASYNCEP over SYNC.

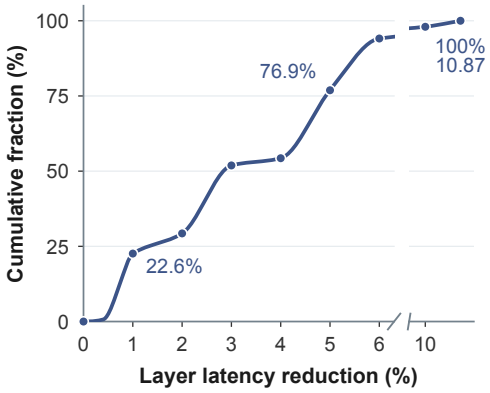



Figure 9: CDF of per-layer latency reduction from enabling *OEWF* for DeepSeek-V4-Flash on AgentX, with *streamFFN* enabled in both configurations.

**FFN computation efficiency.** To isolate FFN computation efficiency from the benefits of early execution, we compare *streamFFN* with synchronous EP's complete FFN GEMM kernels under identical expert weights and per-expert token counts. On DeepSeek-V4-Flash with AgentX, *streamFFN* retains 97.3% of synchronous EP's effective FFN throughput, incurring only a 2.7% loss. The main reason is that the launch threshold preserves near-peak GEMM efficiency by avoiding overly small kernels, while multi-stream execution reduces GPU underutilization caused by the tail effect.

**Layer-level acceleration.** We analyze *OEWF* by comparing execution with and without it for DeepSeek-V4-Flash on AgentX, with *streamFFN* enabled in both configurations. Figure 9 shows that *OEWF* reduces latency by at least 1% in 77.4% of layer forwards and by at least 5% in 23.1%, with a maximum observed reduction of 10.87%. The main reason is that *OEWF* uses otherwise idle GPUs to execute part of the FFN work assigned to replicas with longer attention times, reducing their remaining workload and shortening the layer tail.

## 5 CONCLUSION

This paper presented ASYNCEP, an intra-layer execution engine that mitigates the impact of attention imbalance on MoE prefill. Asynchronous EP starts expert computation before tokens from all attention replicas arrive. *streamFFN* combines kernel launch threshold and multi-stream execution to balance early launches with FFN efficiency, while *OEWF* fetches expert weights and reassigns unstarted FFN work to use GPU idle time. We evaluate ASYNCEP on DeepSeek-V4-Flash, DeepSeek-V4-Pro (DeepSeek, 2026), and GLM-5.3 (Z.ai, 2026) using Tool&Agent (Qin et al., 2025) and AgentX under five request-scheduling policies. The results show that ASYNCEP achieves 1.13× speedup in p95 time-to-first-token (TTFT) on average (up to 1.48×) and improves inference throughput by 1.06× on average (up to 1.17×). These results show that intra-layer asynchronous execution complements request scheduling in improving MoE prefill performance.

### Reproducibility Statement

Section 4 describes the implementation, experimental setup, workload replay procedure, baselines, and evaluation metrics. Appendix A defines the timing boundaries and aggregation methods for attention-imbalance measurements, and Appendix C details the runtime implementation. Appendix E states the assumptions and provides the proof of *OEWF*'s conditional non-degradation result. Appendix D reports the accuracy of the runtime estimates used for donor selection.

### AI Use Statement

We used generative AI tools as assistants throughout this work, under author supervision. First, we used LLM assistance to aid and polish the writing; the authors wrote and verified every technical claim, number, and conclusion. Second, we used LLM-based coding assistants for research execution, to help implement parts of the serving system and the experiment scripts and to help debug and run experiments; the authors reviewed the resulting code and validated all reported results. We also used LLM assistance for retrieval and discovery, including finding related work; the authors checked the suggested literature and verified every citation. We did not use generative AI to generate synthetic datasets, to prove mathematical claims, or to fabricate data, citations, or experimental results.

## References

Osayamen Jonathan Aimuyo, Byungsoo Oh, and Rachee Singh. FlashMoE: Fast distributed MoE in a single kernel. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/918d938bd209e5b56072777366f8a211-Abstract-Conference.html.

Yutian Chen, Cong Li, Yucheng Wang, and Ming Wei. MoonEP: A perfectly balanced expert parallelism library via dynamic redundant experts. GitHub repository, 2026. URL https://github.com/MoonshotAI/MoonEP. Accessed 2026-09-15.

DeepSeek-AI. DeepSeek-V3 technical report, 2024. URL https://arxiv.org/abs/2412.19437. Version 2, revised February 2025; Section 3.4 describes inference deployment.

DeepSeek-AI. DeepSeek-V3/R1 inference system overview. Official Open Infra technical documentation, 2025. URL https://github.com/deepseek-ai/open-infra-index/blob/main/202502OpenSourceWeek/day_6_one_more_thing_deepseekV3R1_inference_system_overview.md. Accessed 2026-09-08.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Kimi Team. Kimi K3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed MoE training and inference with Lina. In Julia Lawall and Dan Williams (eds.), *Proceedings of the 2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, pp. 945–959. USENIX Association, 2023. URL https://www.usenix.org/conference/atc23/presentation/li-jiamin.

Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distributed Syst.*, 12(10):1094–1104, 2001. doi: 10.1109/71.963420. URL https://doi.org/10.1109/71.963420.

NVIDIA. Matrix multiplication background user's guide. Online documentation, 2023. URL https://docs.nvidia.com/deeplearning/performance/

dl-performance-matrix-multiplication/index.html. Sections 3.1–3.2; documentation accessed September 15, 2026.

NVIDIA. CUDA C++ best practices guide. Online documentation, 2026a. URL https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#concurrent-kernel-execution. Concurrent Kernel Execution; documentation accessed September 15, 2026.

NVIDIA. GLM-5.3: Model card. NVIDIA NIM model catalog, 2026b. URL https://build.nvidia.com/z-ai/glm-5-3/modelcard. 753B total parameters and approximately 40B activated per token; accessed September 26, 2026.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation - A KVCache-centric architecture for serving LLM chatbot. In Haryadi S. Gunawi and Vasily Tarasov (eds.), *23rd USENIX Conference on File and Storage Technologies, FAST 2025, Santa Clara, CA, February 25-27, 2025*, pp. 155–170. USENIX Association, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

SemiAnalysis. cc-traces-weka-062126. Hugging Face dataset, 2026. URL https://huggingface.co/datasets/semianalysisai/cc-traces-weka-062126. Trace corpus used by AgentX v1.0. Accessed September 16, 2026.

SGLang Team. DP, DPA and SGLang DP router. Official documentation, n.d. URL https://docs.sglang.io/docs/advanced_features/dp_dpa_smg_guide. Accessed September 26, 2026.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for LLM serving. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=meKEKDhdnx.

vLLM Team and Inferact. vLLM x AgentX: Optimizing for real-world agentic serving. vLLM Blog, 2026. URL https://vllm.ai/blog/2026-09-08-vllm-agentx. September 8, 2026. Accessed September 13, 2026.

Jaehoon Yang, Yushin Kim, Seokwon Moon, Yeonhong Park, and Jae W. Lee. Libra: Effective yet efficient load balancing for large-scale MoE inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=WhxNwgGkAS.

Ying Yuan, Pengfei Zuo, Bo Wang, Zhangyu Chen, Zhipeng Tan, and Zhou Yu. DualMap: Enabling both cache affinity and load balancing for distributed LLM serving. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=zCadrJ32Xn.

Z.ai. GLM-5.3: Model card and configuration. Hugging Face model repository, 2026. URL https://huggingface.co/zai-org/GLM-5.3. Architecture fields from config.json; accessed September 25, 2026.

Dingyan Zhang, Jinbo Han, Kaixi Zhang, Xingda Wei, Sijie Shen, Chenguang Fang, Wenyuan Yu, Jingren Zhou, and Rong Chen. Simple is better: Multiplication may be all you need for LLM request scheduling. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 55–73, 2026. URL https://www.usenix.org/conference/osdi26/presentation/zhang-dingyan.

Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, Quan Chen, and Xin Liu. COMET: fine-grained computation-communication overlapping for mixture-of-experts. In Matei Zaharia, Gauri Joshi, and Yingyan (Celine) Lin (eds.), *Proceedings of the Eighth Conference on Machine Learning and Systems, MLSys 2025, Santa Clara, CA, USA, May 12-15, 2025*. OpenReview.net/mlsys.org, 2025. URL https://openreview.net/forum?id=fGgQS5VW09.

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. DeepEP: An efficient expert-parallel communication library. GitHub repository, 2025. URL https://github.com/deepseek-ai/DeepEP. Official software; Version 2 implementation consulted; accessed 2026-09-15.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In Ada Gavrilovska and Douglas B. Terry (eds.), *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, pp. 193–210. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A. Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, Jianzhe Xiao, Xinyi Zhang, Lingjun Liu, Haibin Lin, Li-Wen Chang, Jianxi Ye, Xiao Yu, Xuanzhe Liu, Xin Jin, and Xin Liu. MegaScale-Infer: Efficient mixture-of-experts model serving with disaggregated expert parallelism. In Marília Curado, Christian Esteve Rothenberg, George Porter, and Srikanth Kandula (eds.), *Proceedings of the ACM SIGCOMM 2025 Conference, SIGCOMM 2025, São Francisco Convent, Coimbra, Portugal, September 8-11, 2025*, pp. 592–608. ACM, 2025. doi: 10.1145/3718958.3750506. URL https://doi.org/10.1145/3718958.3750506.

# A LONG-TAIL STATISTICS OF DEP PREFILL

We profile synchronous DEP prefill with DeepSeek-V4-Flash on the eight-GPU prefill instance described in §4.1. Mooncake's Tool&Agent trace and AgentX are evaluated separately. We measure cross-replica differences in attention-stage duration and in the times at which replicas enter token dispatch.

**Timing and readiness gaps.** We use NVIDIA Nsight Systems to extract attention-stage durations and GPU-side dispatch-arrival timestamps from per-replica execution timelines. SGLang's scheduler coordinates each model forward through a DP metadata all-gather; we match layer-forward invocations across replicas by the coordinated forward index and layer index. This coordination identifies the same execution round but does not imply simultaneous attention starts across GPUs. In the synchronous DeepEP V2 intranode path used for profiling, replicas enter dispatch independently after local attention, routing, and input preparation. For layer-forward invocation $i$, we define $r_{d,i}$ as the GPU start timestamp of replica $d$'s `dispatch_impl` kernel. The kernel performs cross-rank coordination internally, so its start timestamp precedes that coordination rather than marking the start of payload transfer (DeepEP, 2025). We define the cross-replica readiness spread as $s_i = \max_d r_{d,i} - \min_d r_{d,i}$ and each replica's readiness gap relative to the last-arriving replica as $w_{d,i} = \max_j r_{j,i} - r_{d,i}$. We use $w_{d,i}$ to measure replica idle time while waiting for other replicas to reach dispatch: the interval from replica $d$'s entry into dispatch until the last replica enters dispatch, during which synchronous FFN computation cannot begin. This measures waiting for replica arrival, rather than the full dispatch duration.

**Normalization and aggregation.** Let $T_{\mathrm{E2E}}$ denote the wall-clock interval from issuance of the first measured request until the prefill instance produces the first token for the last measured request to complete. Let $\mathcal{I}$ denote the measured layer-forward invocations within this same interval. We define the long-tail ratio as

$$R_{\mathrm{tail}} = \frac{\sum_{i\in\mathcal{I}} \sum_{d=1}^{D} w_{d,i}}{D T_{\mathrm{E2E}}}, \tag{7}$$

where $D = 8$ is the number of attention replicas. This ratio normalizes the accumulated idle time while waiting for other replicas to reach dispatch, averaged across replicas, by the run's elapsed time. We compute it separately for each workload.

The long-tail ratio is 28.00% on Mooncake's Tool&Agent trace and 28.86% on AgentX. On both workloads, idle time while waiting for other replicas to reach dispatch therefore averages more than one quarter of the end-to-end runtime per replica.

**DP attention load balance.** The attention stage includes query and key–value projections, attention-related normalization and positional encoding, local KV-cache updates, core attention computation, and output projection. For DeepSeek-V4-Flash, it also includes KV compression and, where applicable, sparse indexing (DeepSeek, 2026).

For each layer-forward invocation $i$, let $a_{d,i}$ denote the attention-stage duration of replica $d$, measured using identical stage boundaries across replicas. We define attention load balance (ALB) as

$$\mathrm{ALB}_i = \frac{D^{-1} \sum_{d=1}^{D} a_{d,i}}{\max_{1\le d\le D} a_{d,i}} \times 100\%. \tag{8}$$

Higher ALB indicates more balanced attention-stage execution, with 100% corresponding to equal durations across replicas. For Figure 2, we average the per-invocation ALB scores, excluding invocations in which all replicas have zero attention-stage duration. ALB measures relative imbalance within each invocation, whereas the long-tail ratio measures accumulated readiness gaps relative to end-to-end runtime.

# B ATTENTION EXECUTION CHARACTERISTICS

Two attention replicas can receive the same number of requests but different amounts of prefill work, because requests can differ in prompt length and reusable KV-cache prefixes. We compare

attention-stage execution times at a fixed total token count $N = BL$ across batch sizes $B$ and input lengths $L$, using attention-module profiles of DeepSeek-V4-Flash, GLM-5.3, and DeepSeek-V3-0324 (DeepSeek, 2024). DeepSeek-V3-0324 is included only in this comparison, not in the end-to-end evaluation.

**Token count and context length.** At fixed model dimensions, projection GEMM FLOPs scale with the number of processed tokens $N$. Context-dependent attention work need not follow the same scaling. For $B$ equal-length sequences of length $L$ with no reusable prefix, dense causal attention contains

$$P(B, L) = \frac{BL(L+1)}{2} = \frac{N(L+1)}{2}, \qquad N = BL, \tag{9}$$

query–key pairs. At fixed head dimensions, attention-score and value-aggregation FLOPs scale with $P(B, L)$. Thus, at the same total token count, fewer long sequences require more dense-attention arithmetic than more short sequences. Sparse selection and KV compression reduce the context-dependent work. These reductions can make token-proportional operations more prominent without making the full attention stage independent of batch composition.

**Attention timing distributions.** DeepSeek-V4-Flash, DeepSeek-V3-0324 and GLM-5.3 are profiled as complete attention-module calls on eight B200 GPUs. The measurements use no prefix-cache reuse. Each panel of Figure B1 is normalized by its largest displayed mean time. Box spread reflects differences across configurations rather than run-to-run variability. At a fixed total token count, DeepSeek-V3-0324 and GLM-5.3 exhibit substantial execution-time variation across batch-size and input-length combinations, whereas DeepSeek-V4-Flash shows a smaller relative spread in the displayed profiles. FLOPs explain the workload dependence but do not directly determine execution time. Total token count is therefore a useful workload indicator, but does not guarantee equal attention-stage execution times across different batch compositions.

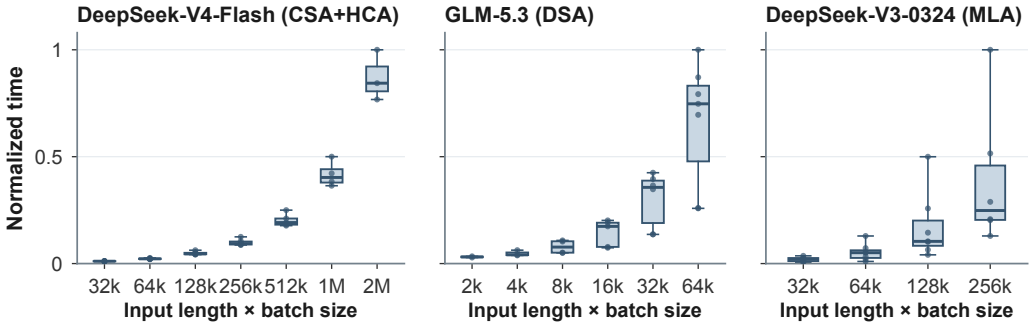



Figure B1: Attention execution times across batch-size and input-length configurations at fixed total token counts, without prefix-cache reuse. Each panel is normalized by its largest displayed time.

**Online batching and cache locality.** Request placement selects a replica, whereas the local scheduler forms prefill batches from queued requests under token and memory budgets (Zheng et al., 2024). A request's batch is therefore not fixed by placement alone. Cache locality further constrains the choice: assigning a request to a less-loaded replica that lacks its reusable prefix requires additional KV recomputation or transfer. Cache-aware policies account for this trade-off (Srivatsa et al., 2025), but must do so as queues and cache contents change. These constraints make it difficult to consistently balance attention-stage execution times across replicas through request placement alone.

**Estimating the remaining attention time.** With the replica placement fixed by the scheduling policy and the cache hits known, we estimate the remaining attention time. The remaining attention-stage time in Equation 1 is

$$A_d = \left[ f(I_d, P_d) - t_d^{\text{att}} \right],$$

where $f(I_d, P_d)$ is the predicted duration of replica $d$'s complete attention stage, including projections, KV compression and sparse indexing where applicable, core attention computation, and the

output projection, and $t_d^{\text{att}}$ is the elapsed attention time. If $f(I_d, P_d) - t_d^{\text{att}}$ is negative, we set $A_d$ to 0. Replica $d$ is then excluded from the candidate donors. Let $\mathcal{R}_d$ denote the requests in replica $d$'s current batch, so that $I_d = \{I_i\}_{i\in\mathcal{R}_d}$ and $P_d = \{P_i\}_{i\in\mathcal{R}_d}$. Request $i$ computes $q_i = I_i - P_i$ new tokens, and its $j$-th new token attends to at most $\min(k_i, P_i + j)$ keys, where $k_i$ is the per-query key budget of sparse selection and $k_i = \infty$ for dense attention. Following prefill cost models that aggregate per-request token counts and squared lengths (Zhong et al., 2024) and account for prefix-cache hits (Qin et al., 2025), we accumulate two workload features over the requests in the batch:

$$Q_d = \sum_{i\in\mathcal{R}_d} q_i, \qquad \Pi_d = \sum_{i\in\mathcal{R}_d} \sum_{j=1}^{q_i} \min\big(k_i,\, P_i + j\big), \tag{10}$$

and predict the attention-stage duration as

$$f(I_d, P_d) = \alpha_0 + \alpha_1 Q_d + \alpha_2 \Pi_d. \tag{11}$$

$Q_d$ accounts for token-proportional operations such as the projections, and $\Pi_d$ counts the query–key pairs of core attention. For $B$ equal-length requests of length $L$ with dense attention and no reusable prefix, $Q_d = N$ and $\Pi_d = P(B, L)$ in Equation 9; $\Pi_d$ generalizes this pair count to heterogeneous lengths, matched prefixes, and sparse selection. Because both features are summed over requests, length variation within a batch is retained rather than averaged away. For each deployed model, GPU, and attention configuration, we fit $\alpha_0$, $\alpha_1$, and $\alpha_2$ to offline profiles by weighted least squares that minimizes relative error. Layers with the same attention configuration share one set of coefficients; in DeepSeek-V4-Flash, for example, layers with different KV-compression settings are fitted separately. The profiles cover batch sizes $B \in \{1, 2, 4, 8, 16, 32, 64\}$ and input lengths $I \in \{1\text{K}, 2\text{K}, 4\text{K}, 8\text{K}, 16\text{K}, 32\text{K}, 64\text{K}\}$ tokens, omitting infeasible combinations that exceed GPU memory. For these equal-length profiles, the matched prefix length is $P_i = rI$, with hit ratio $r \in \{0\%, 20\%, 40\%, 60\%, 70\%, 80\%, 90\%, 95\%\}$. The linear form does not model kernel tiling or wave-quantization effects; Appendix D reports the measured error.

## C Runtime Implementation Details

**Asynchronous dispatch.** Our DeepEP extension maintains a separate communication lane for each source–destination pair. At the destination, a lane becomes ready after the source's routed activations and associated metadata have been received and packed into expert-major order. A generation-tagged completion signal identifies the corresponding dispatch instance, while system-scope release/acquire ordering ensures that the data are visible before consumption. The SGLang adapter uses CUDA stream waits on these signals, allowing inputs from one attention replica to become eligible for computation without waiting for the other replicas. Readiness does not immediately trigger a grouped FFN invocation: *streamFFN* accumulates ready work and applies the launch threshold and final-flush rule described in §3.2.

**Grouped FFN invocations.** For each grouped FFN invocation, a GPU-side Triton layout transform combines the selected rows from ready source lanes into contiguous groups by expert. It computes per-expert row offsets from the selected token counts, aligns each group to the GEMM tile size, and packs the activations together with their quantization metadata. This allows rows for the same expert to share one compute group even when they arrive through different source lanes. An inverse map preserves each row's source replica, token index, and routing slot for returning and combining the outputs. Within an invocation, the two expert GEMMs and the intervening activation and quantization operations execute in stream order. Independent invocations use separate CUDA streams and scratch buffers, preserving these dependencies while permitting concurrent execution.

**One-shot weight fetching.** *OEWF* uses a preallocated temporary weight buffer on each helper to store the fetched expert matrices and quantization metadata. A per-rank, per-layer-forward flag permits one fetch decision after local dispatch completes. Following the selection rule in §3.3, the coordinator chooses the eligible donor with the latest predicted FFN completion time and fixes the expert and token sets within a byte budget bounded by the buffer's available capacity. The implementation considers only tokens dispatched by the helper whose inputs it retains locally. Before issuing asynchronous copies, the coordinator validates shared reservations and reserves the protected transfer service required by the design. The selected experts' weights are copied in full from

the donor into this buffer. During prefetching, the selected tokens remain in the donor's workload and retain their original launch eligibility; the donor does not wait for the copies. Weight readiness triggers a separate FFN reassignment decision. Empty selections, reservation failures, failed copies, and rejected FFN reassignments do not trigger another fetch selection.

**FFN reassignment commit and fallback.** Once weights are ready, the coordinator considers tokens recorded in the helper's fetch plan. A token is eligible only if its donor batch has a fixed token set and its FFN execution has not been submitted. It evaluates the admission test in Equation 6 against the latest committed plan, including earlier helper commitments, and requires a protected interval that accommodates the complete helper invocation without delaying existing work. A version-checked atomic commit revalidates the plan, invocation state, and token ownership, then transfers execution ownership and updates the donor's residual token counts and the helper's reservation together. If validation fails, the donor has already submitted the work, or another helper has claimed it, the attempted FFN reassignment is abandoned and execution remains with the current owner. Successfully reassigned tokens execute on their retained local inputs at the helper. The donor retains its original launch eligibility even if reassignment reduces its remaining work below the launch threshold. The non-degradation guarantee remains conditional on the timing bounds and resource protections stated in Appendix E.

**Output return and combine.** Expert outputs are returned asynchronously to their originating attention replicas, or passed directly to local combine when computed there. Source-token and routing-slot metadata associate each contribution with the corresponding token and selected expert. Each routed-expert contribution is weighted by its original routing coefficient exactly once. Combine returns and accumulates these contributions at the originating attention replica, and the result is added to the shared-expert output. The runtime tracks output readiness separately from buffer reclamation: buffers remain live until their dependent transfers and computations have completed. Although contributions can arrive and be accumulated independently, ASYNCEP retains the common layer boundary. The next layer is admitted only after every attention replica has received its required MoE outputs.

## D RUNTIME ESTIMATES AND THEIR ACCURACY

Coordinated greedy expert-weight selection (§3.3) ranks candidate donors using two predicted quantities: the remaining attention-stage time $A_d$ in Equation 1 and the received expert-token count $N_d$ in Equation 3. Appendix B describes the attention-time estimator. This appendix describes how the received expert-token count is estimated and reports the accuracy of the complete attention-stage duration used to form $A_d$, together with the accuracy of $N_d$. The non-degradation analysis in Appendix E assumes accurate timing; these measurements quantify the estimation error observed in practice and do not extend that guarantee.

**Estimating received expert-token counts.** Let $N_{\ell,d}$ denote the number of expert tokens that rank $d$ receives in MoE layer $\ell$. When the current layer's count is not yet known, we estimate it with the count that the same rank received in the preceding MoE layer:

$$N_d = N_{\ell-1,d}. \tag{12}$$

Within a forward pass, every MoE layer routes the same tokens, each to the same number of experts, so the total number of token–expert pairs is identical across layers; only its distribution across ranks changes with routing. The estimate therefore approximates the current layer's per-rank total with that of the preceding layer. The first MoE layer has no preceding layer, and *OEWF* skips it.

**Error metric.** For each sample, we measure the gap between the predicted value $\hat{y}$ and the measured value $y > 0$ relative to the measured value, $\epsilon = |\hat{y} - y|/y$, and report accuracy as $1 - \bar{\epsilon}$, where $\bar{\epsilon}$ is the mean of $\epsilon$ over all samples. Both estimates are validated online while serving DeepSeek-V4-Flash, DeepSeek-V4-Pro, and GLM-5.3 on the Tool&Agent and AgentX workloads at the request rates in Figure 6.

**Attention-stage time.** For each replica and MoE layer, we record the input lengths and matched prefix lengths of the current batch together with the measured attention-stage duration, using the

stage boundary defined for $f$ in Appendix B. The accuracy of 90.3% compares $f(I_d, P_d)$ with this measured complete attention-stage duration.

**Received expert-token count.** For each rank and layer, we compare $N_d$ from Equation 12 with the number of expert tokens that rank $d$ actually receives in layer $\ell$. Because *OEWF* skips the first MoE layer, these samples start from the second MoE layer. The estimate achieves an accuracy of 85.9%.

## E Non-degradation of Opportunistic Expert Weight Fetching

This appendix shows how the FFN reassignment rule in §3.3 gives non-increasing MoE layer completion time relative to execution without weight fetching, under the assumptions stated below. We first consider a donor–helper pair, then extend its completion-time comparison to all ranks and successive plan updates.

**Proposition 1** (From two ranks to any number of ranks)**.** *Under the conditions stated in the proof, sequential updates satisfying Equation 6 give $M_{\mathrm{OEWF}} \le M_{\text{no-fetch}}$ for any number of ranks.*

*Proof.* Fix the layer inputs and routing. At a weight-ready decision, let $T_r(S)$ denote rank $r$'s remaining FFN completion time under the current plan $S$, measured from the same decision instant for both plans. Assume the completion-time model in §3.3, including FFN latencies under concurrent execution and the overlap estimate, is accurate, and weight fetching does not delay the current plan. A reassignment preserves the donor's attention and dispatch times and the completion times of the helper's existing work. Using the same FFN latency function $F$ as Equation 2, with $F(0) = 0$, transferring $0 \le N_{\mathrm{trans}} \le N_d$ unstarted expert tokens gives

$$\begin{aligned} T_d(S) &= A_d + D_d + F(N_d), \\ T_d(S') &= A_d + D_d + F(N_d - N_{\mathrm{trans}}), \\ T_h(S') &= T_h(S) + \alpha_h F(N_{\mathrm{trans}}). \end{aligned} \tag{13}$$

For $N_{\mathrm{trans}} > 0$, let $s_h \in [0, T_h(S)]$ be the planned launch time of the reassigned FFN kernel, measured from this decision. Setting $\alpha_h = \max\{0, s_h + F(N_{\mathrm{trans}}) - T_h(S)\}/F(N_{\mathrm{trans}})$ gives $\alpha_h \in [0, 1]$ and $T_h(S') = \max\{T_h(S), s_h + F(N_{\mathrm{trans}})\}$, accounting for partial or complete overlap. With no transferred tokens, the plan is unchanged. Equation 6 accepts the update only when

$$\max\{T_d(S'), T_h(S')\} \le \max\{T_d(S), T_h(S)\}. \tag{14}$$

Reassignment does not delay ranks outside $\{d, h\}$. Taking the maximum over all ranks therefore gives

$$T_{\max}(S') = \max_r T_r(S') \le \max_r T_r(S) = T_{\max}(S). \tag{15}$$

Let $M(S)$ be the layer completion time measured from layer entry, $t$ the decision time on this same origin, and $L(S) = M(S) - t - T_{\max}(S)$ the residual return/combine and layer-synchronization tail after the last FFN completes. Assume this tail does not increase, including for helper outputs, so $L(S') \le L(S)$. Then

$$\begin{aligned} M(S') &= t + T_{\max}(S') + L(S') \\ &\le t + T_{\max}(S) + L(S) = M(S). \end{aligned} \tag{16}$$

A rejected update retains $S$. The coordinator applies each accepted update to the latest joint plan, so its token counts, completion times, and overlap estimates include all previous reassignments. Starting from the no-fetch plan $S_0$, whose execution is unaffected by fetching, induction gives

$$M_{\mathrm{OEWF}} = M(S_K) \le M(S_{K-1}) \le \cdots \le M(S_0) = M_{\text{no-fetch}}. \tag{17}$$

Here $S_K$ is the final *OEWF* plan. □