ARK: Avoiding Routing Collisions for KV Cache Transfer in Disaggregated LLM Inference

Hung-Chun Lin*
Georgia Institute of Technology
Atlanta, GA, USA
hlin464@gatech.edu

Ting-Wei Hsu*
Georgia Institute of Technology
Atlanta, GA, USA
thsu49@gatech.edu

Chung-En Ho
Georgia Institute of Technology
Atlanta, GA, USA
cho322@gatech.edu

Ahmed Saeed
Georgia Institute of Technology
Atlanta, GA, USA
asaeed@cc.gatech.edu

Abstract

Disaggregated LLM inference separates the prefill and decode phases across GPU pools, generating massive KV-cache transfers. Because these transfers last hundreds of milliseconds to seconds, they behave as mega elephant flows that dominate link bandwidth utilization. In this regime, stateless ECMP can perform poorly: hash collisions may overload one spine link while leaving others idle, stretching transfer times by seconds. Yet this same persistence makes coordination practical. Since these flows are long-lived, even lightweight one-to-all coordination can be amortized over their lifetime. We present ARK, a distributed elephant-flow path reservation mechanism. ARK coordinates senders to choose source ports whose hashes map concurrent flows onto distinct spines, without requiring switch changes or receiver-side packet reordering. Packet-level RDMA simulations show that ARK reduces mean and P95 FCT by up to 27.3% and 34.0% under moderate load, and further reduces mean TTFT by up to 12.9%.

CCS Concepts

• **Networks** → **Network simulations**; **Data center networks**; *Network resources allocation*; • **Computing methodologies** → *Simulation evaluation*; Discrete-event simulation.

Keywords

LLM serving, disaggregated inference, network load balancing

ACM Reference Format:
Hung-Chun Lin, Ting-Wei Hsu, Chung-En Ho, and Ahmed Saeed. 2026. ARK: Avoiding Routing Collisions for KV Cache Transfer in Disaggregated LLM Inference. In *Workshop on Networks for AI Computing (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3789240.3828750

*Both authors contributed equally to this research.

SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3828750

1 Introduction

The rise of large language models (LLMs) is reshaping data-center utilization. Modern LLM serving pipelines split execution into two phases with disjoint resource profiles: a compute-bound *prefill* phase that ingests the user prompt, and a memory-bandwidth-bound *decode* phase that emits tokens autoregressively. Co-locating both phases on the same GPU pool over-provisions one resource for the other, motivating recent serving systems—DistServe [31] and Splitwise [16]—to physically *disaggregate* prefill from decode across separate GPU pools. The price of this architecture is a new dominant communication pattern: the key–value (KV) cache produced by a prefill GPU must be shipped over the data-center fabric to the decode GPU before token generation can begin. Open-source inference platforms such as vLLM [11], SGLang [30], NVIDIA Dynamo [15], and llm-d [25] already implement this prefill–decode disaggregation pattern, and major LLM operators have adopted it at production scale: Moonshot AI's Kimi runs on the Mooncake disaggregated architecture [20], and DeepSeek-V3 is served via SGLang's disaggregated reference deployment [23].

KV cache flows are unlike anything the fabric has seen. Where a traditional data-center "elephant flow" typically peaks around tens of megabytes [24], a single 100K-token KV cache for LLaMA-3.1-405B already reaches 48 GB.[1] A single such flow can saturate a 400 Gbps link for 960 ms. Because this transfer sits on the critical path of every request, even a small increase in flow completion time (FCT) directly amplifies time-to-first-token (TTFT), potentially leading to Service Level Objective (SLO) violations.

Production fabrics commonly use Equal-Cost Multi-Path (ECMP) routing to spread flows across redundant equal-cost paths. ECMP selects a next hop by hashing the packet 5-tuple (source IP, destination IP, source port, destination port, and protocol), thereby pinning each flow to a single path. Modern AI cluster fabrics retain ECMP rather than replace it: Meta's RoCE deployment for distributed training [7] and Alibaba HPN [19] both keep commodity ECMP and engineer around its weaknesses, and the production inference stacks surveyed in §2 ship on the same baseline. The hash, however, is oblivious to flow size, so two KV cache mega-flows that happen to hash to the same spine collide and share its capacity for

[1] KV-cache footprint per token is $2 \cdot L_h \cdot H_{kv} \cdot d_h \cdot s$, where L_h is the number of decoder layers, H_{kv} the number of KV heads, d_h the head dimension, and s the precision in bytes. For LLaMA-3.1-405B (126 layers, 8 KV heads, d_h=128, FP16), it has 504 KB per token.

the multi-second lifetime of the transfer, potentially leaving other paths idle. Existing techniques for avoiding such collisions in the network [3, 4, 21, 24, 26] introduce significant cost either through hardware changes or specific software requirements.

We propose ARK, a lightweight distributed elephant-flow path reservation mechanism that runs entirely in commodity end-host software while leaving the switch data plane untouched. Each host maintains a replicated reservation table recording which spines are currently occupied by active KV cache transfers; when a new flow arrives, the local controller selects an unreserved spine and dispatches the flow with a precomputed source-port offset that maps to it under ECMP, and broadcasts the reservation to peers via a low-bandwidth control channel. Because the mechanism only manipulates the source port—a field every host already controls—the data plane requires no switch modification and no packet reordering at the receiver. We evaluate ARK in a packet-level RDMA simulator (SimAI/NS-3) driven by realistic disaggregated-inference workloads from Vidur [1]. In a two-tier fat-tree of 16 GPUs (8 prefill / 8 decode) under a LLaMA-3-8B Zipf workload, ARK reduces mean / P95 flow completion time (FCT) by up to 27.3% / 34.0% and end-to-end time-to-first-token (TTFT) by up to 12.9%, with the largest gains in the load regime where the leaf–spine fabric is congested but not yet saturated.

2 Background and Motivation

Prefill–decode disaggregation. LLM serving alternates between compute-bound *prefill* (attention $\Theta(L^2)$ in FLOPs) and high-bandwidth memory (HBM) bandwidth-bound *decode* (each iteration reads the full KV cache from HBM) [16, 31]. Co-locating both phases either inflates inter-token latency through prefill–decode interference or forces a coupled pool to over-provision against the independent TTFT and time-per-output-token (TPOT) service-level objectives (SLOs), each gated by a different resource axis [31]. Disaggregating onto separate pools lets each pool be sized to its own bottleneck, optionally with a different accelerator model per phase [16]. Disaggregated serving is now common practice. Mooncake [20], which serves Moonshot AI's Kimi chatbot, operates disaggregated KV-cache pipelines at production scale, while the SGLang reference deployment of DeepSeek-V3 on 96 H100 GPUs adopts prefill–decode disaggregation [23]. In the open-source ecosystem, both vLLM [11] and SGLang [30] support disaggregated serving. More recently, two industry-led efforts—NVIDIA Dynamo [15] and llm-d [25]—have made prefill–decode disaggregation a central architectural principle.

Application-level KV cache optimizations Once prefill–decode disaggregation is taken as given, two orthogonal lines of work address the resulting communication cost: one above the fabric, on the KV cache itself, and one inside the fabric, on how its bytes are routed. *Multi-tier KV stores* externalize the cache into pooled memory hierarchies so state can be shared across machines and reused across requests—MemServe [9] provides an elastic memory pool with locality-aware routing, and LMCache [6] exposes a cross-engine KV layer for transfer between different serving entities. A complementary *transfer-optimization* thread shrinks or reroutes the bytes themselves: CacheGen [14] compresses and adaptively streams the cache to shrink bytes-on-wire, DualPath [28] routes reads through decode engines to rebalance storage NIC load, and Yue et al. [29] analyze per-block protocol overhead in geo-distributed deployments. None of these approaches controls *which* equal-cost path the remaining bytes traverse once admitted to the fabric—a dimension that ARK addresses and that compounds with, rather than competes against, the bytes-on-wire reductions above. We turn next to how production fabrics balance the bytes that do hit the wire.

Load balancing in modern AI fabrics. Industry has demonstrated willingness to build custom hardware and software for AI workloads whenever the cost-benefit warrants it. For elephant-flow load balancing on AI fabrics, however, the deployed reality is uniformly commodity ECMP. Meta's RoCE deployment for distributed training [7] chose to enrich the input space ("Enhanced ECMP") and layer centralized traffic engineering rather than replace ECMP outright; Alibaba HPN [19] kept ECMP and rebuilt the topology as a dual-plane, dual top-of-rack (ToR) fabric to suppress hash polarization.

The relevant question for an in-network elephant-flow mechanism is therefore comparative cost on top of this baseline, not technical feasibility. CONGA [4] adds custom congestion-aware leaf ASICs across the fabric; ConWeave [24] requires programmable ToRs paired with per-flow reordering buffers; LetFlow [26] adds switch-level flowlet detection. While packet spraying effectively mitigates ECMP collisions, it delivers packets out of order; for mega-flows like KV cache transfers, reassembling them imposes a heavy CPU burden at the receiver. Host-side PLB [21] avoids switch upgrades but is *reactive*: it only repaths after M consecutive ECN-marked rounds, and only when the connection is idle, since mid-flow IPv6 Flow Label changes cause reordering that RoCEv2 RNICs misread as loss [24]. This reactive, congestion-control-first posture matches TCP workloads where flow size and arrival are unknown, but it is a poor fit for KV cache transfers, whose size and destination are known the instant the prefill engine finalizes the cache. Worse, RDMA's hardware-paced KV transfers leave essentially no idle gaps [24], so PLB's safe-repath precondition rarely fires within a KV flow's lifetime. ARK instead exploits this admission-time knowledge (§3.2) to fix the path *before the first byte ships*, sidestepping congestion-control feedback, reactive lag, and the idle-gap requirement.

Opportunity for coordination. KV cache flows differ from the flows that motivated prior datacenter load balancers along two axes that, together, expose a design space earlier mechanisms could not exploit.

Flow lifetime is long enough to amortize coordination. A 1.6 GB mean KV transfer occupies a 100 Gbps spine link for at least 128 ms, and longer when sharing the spine—three to four orders of magnitude above the millisecond-class flows targeted by reactive flowlet schemes [21, 26]. Millisecond-scale admission-time coordination therefore amortizes well under 1% of flow lifetime, where reactive flowlet schemes have no comparable budget.

Each flow's identity is known at admission. The destination GPU, byte count, and start time of every KV transfer are determined the moment the prefill engine finalizes the cache; no runtime elephant detection (as in Hedera [3]) is required. The equal-cost paths are likewise enumerable at the host, so both the flow and its paths are exactly known before any byte ships.

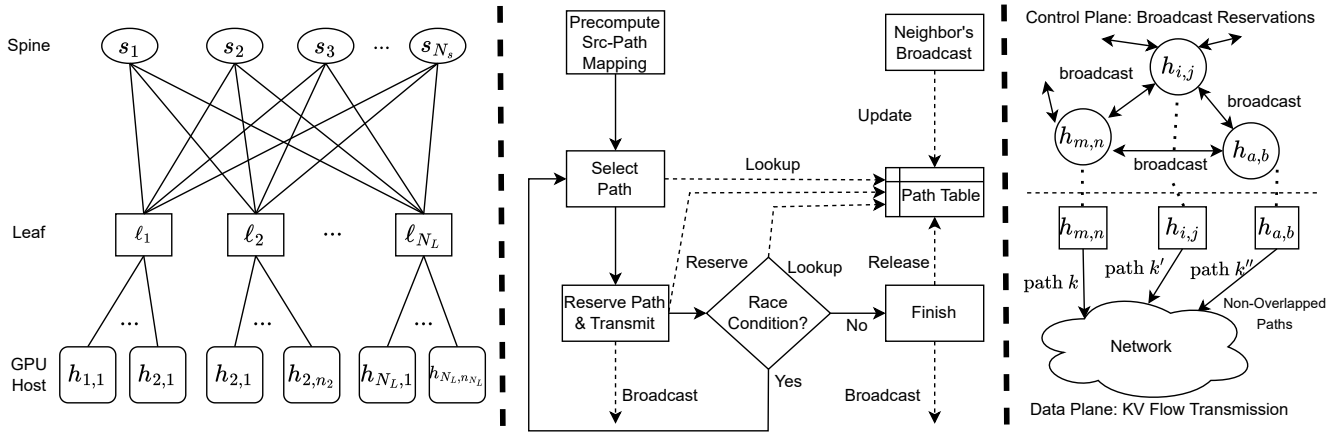


Figure 1: (a) Network topology (b) ARK path reservation controller (c) ARK deployment with separated control and data planes

ARK exploits both. At admission, a local controller picks an unreserved spine from a host-replicated reservation table, dispatches the flow with a precomputed source-port offset that hashes onto it, and broadcasts the reservation to peers. The decision needs no runtime elephant detection and no in-flight congestion signaling (CONGA [4]); each flow uses a single path, so no receiver-side reordering is required.

3 ARK System Design

ARK enables ECMP hash collision avoidance through a distributed path reservation controller on each server. By maintaining a local replica of the reservation table via lightweight broadcasts between peers, each controller is aware of which paths are currently available when it admits an outgoing KV cache flow.

ARK as minimal scheduling. Avoiding ECMP collisions is fundamentally a scheduling problem, and ARK sits at the lightweight end of a wide design space. Centralized schedulers coordinate the whole fabric through an arbiter—Fastpass [18] per packet, Flowtune [17] per flowlet—while a more recent line schedules *collective* communication for AI training (FAST [12] for MoE `alltoallv`, Crux [5] for training collectives). ARK occupies the opposite extreme by design: it needs no central arbiter and no traffic matrix, scheduling only the choice of spine, once per flow at admission. This is a fit to the long-lived, admission-known KV workload, not a shortcut.

Assumptions. We make two assumptions. First, the KV cache transfer application uses a single, well-known destination port across all peers, leading to fixed src IP, dst IP, dst port, and IP protocol per peer pair, so only the source port varies (the precompute below relies on this; multi-dport scenarios extend trivially to a per-(peer, dport) table). Second, host clocks are synchronized to within the broadcast delay (a few microseconds in our deployment), achievable on commodity datacenter fabrics via PTP [10]. This assumption is needed by the race-condition resolution rule in §3.2.

3.1 Path Reservation Controller

Topology: We model the disaggregated LLM inference fabric as a two-tier spine-leaf network [2] (Fig. 1(a)) consisting of N_S spine switches $V_S = \{s_1, \ldots, s_{N_S}\}$, N_L leaf (top-of-rack) switches $V_L = \{\ell_1, \ldots, \ell_{N_L}\}$, and a set of GPU hosts V_H in which $h_{i,m}$ denotes the m-th host attached to leaf ℓ_i. Each leaf is fully connected to every spine, and every host has a single uplink to its leaf. For any cross-leaf host pair $(h_{i,m}, h_{j,n})$, the fabric exposes $K = N_S$ equal-cost paths

$$\begin{aligned} \mathcal{P}(h_{i,m}, h_{j,n}) &= \{p_1, \ldots, p_K\}, \\ p_k &= (h_{i,m} \to \ell_i \to s_k \to \ell_j \to h_{j,n}), \end{aligned} \tag{1}$$

each traversing exactly one spine. For derivation convenience we present the formalism on a two-tier spine-leaf instance, taking Alibaba HPN [19] as a representative production example, while ARK extends to deeper Clos fabrics such as Meta's multi-tier RoCE deployment [7] by inverting the ECMP hash at each ECMP decision along the path.

ECMP routing and hash collisions: Switches forward each flow f_i using standard ECMP, hashing the 5-tuple $\mathbf{q}_i = (\text{srcIP}_i, \text{dstIP}_i, \text{srcPort}_i, \text{dstPort}_i, \text{proto}_i)$ and selecting

$$\pi_i = p_{k_i}, \qquad k_i = \mathcal{H}(\mathbf{q}_i) \bmod K, \tag{2}$$

where $\mathcal{H}$ is the switch hash function. Because $\mathcal{H}$ is oblivious to flow size, two elephant flows whose hashes coincide modulo K share the same spine link and split its bandwidth, regardless of whether sibling spines are idle. AI Clos fabrics commonly oversubscribe the leaf–spine layer for cost reasons, which further amplifies collision impact: with less spare bandwidth, collision-induced queueing fills buffers faster. Their flow completion times T_i^{FCT} scale roughly with the number of competing flows on that link, with RDMA retransmissions further amplifying the penalty when buffers overflow. We write τ_i for the time at which flow f_i is injected into the fabric.

Reservation table: Each host runs a path reservation controller that maintains a local path reservation table $\mathcal{R}(t) \subseteq \mathcal{P}$, recording the paths currently occupied by active elephant flows across the fabric.

Boot-time precompute: Under this assumption, the controller precomputes a per-peer src port-to-path mapping

$$\mathcal{M}_{h_j}(p_k) = \left\{ \delta \;:\; \mathcal{H}(\mathbf{q}^{\delta}) \bmod K = k \right\}, \tag{3}$$

once at server boot, where $p_k \in \mathcal{P}(\text{self}, h_j)$ is the k-th path toward h_j and $\mathcal{M}_{h_j}(p_k)$ enumerates the src-port offsets that route flows toward h_j onto p_k.

Flow arrival: When a new flow f_i arrives at the host at time τ_i, the controller selects an unreserved path and reads any one offset from the corresponding bucket:

$$\pi_i^{\text{ARK}} \;=\; p_{k_i^*}, \quad k_i^* \;=\; \min\{\, k \;:\; p_k \notin \mathcal{R}(\tau_i) \,\}, \tag{4}$$

a short scan of $\mathcal{R}$ that probes at most K paths. The corresponding source-port offset δ_i^* is then taken from the precomputed bucket $\mathcal{M}_{\text{dst}_i}(p_{k_i^*})$. Upon assignment, the chosen path is added to the local path reservation table and broadcast to all peers:

$$\mathcal{R}(\tau_i^+) \;=\; \mathcal{R}(\tau_i) \cup \{\, p_{k_i^*} \,\}. \tag{5}$$

When the flow completes, the path is released from the table and a corresponding RELEASE message is broadcast:

$$\mathcal{R}\big((\tau_i + T_i^{\text{FCT}})^+\big) \;=\; \mathcal{R}\big(\tau_i + T_i^{\text{FCT}}\big) \setminus \{\, p_{k_i^*} \,\}. \tag{6}$$

Peer controllers apply incoming RESERVE/RELEASE updates to their own path reservation tables, keeping them eventually consistent across the fabric. When all paths are reserved, i.e., $|\mathcal{R}(t)| = K$, the controller falls back to vanilla ECMP by randomly drawing a path index $k_i^* \in \{1, \ldots, K\}$ and taking δ_i^* from $\mathcal{M}_{\text{dst}_i}(p_{k_i^*})$. The full procedure is summarized in Algorithm 1.

3.2 ARK Deployment

Each host runs the controller of §3.1 as user-space code. Following Fig. 1(c), ARK separates the two kinds of traffic the controller produces into a control plane and a data plane.

Control plane: The control plane carries the RESERVE/RELEASE broadcasts that synchronize the path reservation table $\mathcal{R}$ across hosts. These messages are tiny and latency-sensitive: the broadcast latency directly bounds the race window between two hosts contending for the same path. We therefore deploy the control plane on a low-bandwidth network or an existing management channel, so reservation latency is bounded by propagation rather than by queueing behind elephant flows on the data fabric. This is a deployment choice, not an algorithmic requirement: ARK still functions when control messages share the data fabric, but the race window grows under load.

Data plane: The data plane carries the KV cache flows themselves over the regular RDMA fabric. On each flow arrival the controller determines δ_i^* and hands it to the RDMA stack, which creates the queue pair using δ_i^* and transmits the KV cache; switches apply standard ECMP hashing over the 5-tuple as in vanilla deployments, so no switch modification is required.

Handling race condition: Because $\mathcal{R}$ is replicated and updated asynchronously, two hosts may reserve the same path within a broadcast delay window. The conflict is resolved by a deterministic timestamp-based rule: each RESERVE broadcast carries the key (t_i, id), the smaller key wins, and the loser re-hashes to another path (Algorithm 1, OnRemoteUpdate). The (t_i, id) ordering relies on the clock-synchronization assumption stated in §3. Existing approaches, such as sub-μs precision via PTP [10], Sundial [13], and Huygens [8], more than meet this requirement.

Algorithm 1 ARK Path Reservation Controller

Require: Path set $\mathcal{P}$; switch hash $\mathcal{H}(\cdot)$; host identifier id; peer set $\mathcal{V}$

$\mathcal{R} \leftarrow \emptyset$ ▷ $p_k \mapsto (t, \text{id})$

procedure Initialize ▷ run once at server boot
  for each peer $h_j \in \mathcal{V}$ **do**
    $\mathcal{M}_{h_j}[p_k] \leftarrow \emptyset$ for all $p_k \in \mathcal{P}(\text{self}, h_j)$
    for $\delta = 0, 1, \ldots, \delta_{\max}$ **do**
      $\mathbf{q}^\delta \leftarrow$ 5-tuple to h_j with src port $= \delta$
      $k \leftarrow \mathcal{H}(\mathbf{q}^\delta) \bmod K$
      $\mathcal{M}_{h_j}[p_k] \leftarrow \mathcal{M}_{h_j}[p_k] \cup \{\delta\}$
    end for
  end for
end procedure

procedure OnFlowArrival(f_i)
  $t_i \leftarrow$ current timestamp
  if $|\mathcal{R}| = K$ **then** ▷ Fall back to ECMP
    $k_i^* \leftarrow$ random $k \in \{1, \ldots, K\}$; $\delta_i^* \leftarrow$ any $\delta \in \mathcal{M}_{\text{dst}_i}[p_{k_i^*}]$
  else
    $k_i^* \leftarrow \min\{k : p_k \notin \mathcal{R}\}$
    $\delta_i^* \leftarrow$ any $\delta \in \mathcal{M}_{\text{dst}_i}[p_{k_i^*}]$
    $\mathcal{R}[p_{k_i^*}] \leftarrow (t_i, \text{id})$
    Broadcast $(p_{k_i^*}, \text{RESERVE}, t_i, \text{id})$ to peers
  end if
  $\pi_i \leftarrow p_{k_i^*}$
  Create RDMA queue pair for f_i with src port δ_i^*
  Register OnFlowCompletion(f_i, k_i^*) callback
end procedure

procedure OnFlowCompletion(f_i, k_i^*)
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \{p_{k_i^*}\}$
  Broadcast $(p_{k_i^*}, \text{RELEASE})$ to peers
end procedure

procedure OnRemoteUpdate(p_k, op, t_p, id_p)
  if op = RELEASE **then**
    $\mathcal{R} \leftarrow \mathcal{R} \setminus \{p_k\}$
  else if $p_k \notin \mathcal{R}$ **then**
    $\mathcal{R}[p_k] \leftarrow (t_p, \text{id}_p)$
  else if $(t_p, \text{id}_p) < \mathcal{R}[p_k]$ **then** ▷ Race condition
    $\mathcal{R}[p_k] \leftarrow (t_p, \text{id}_p)$
    OnFlowArrival(f) ▷ Re-hash local flow on p_k
  end if
end procedure

4 Experiments and Results

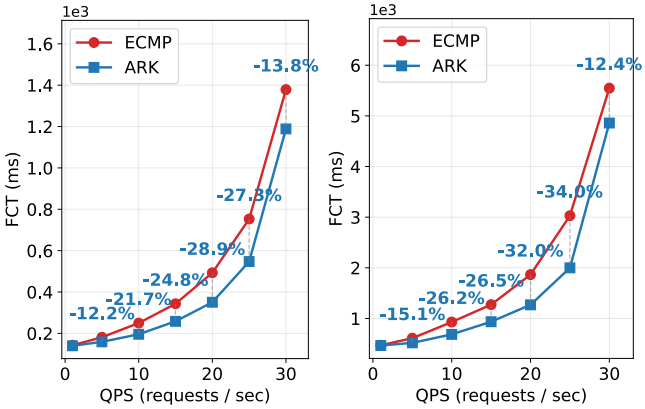
We evaluate ARK by integrating an LLM inference serving simulator with a packet-level network simulator.

Our evaluation answers the following questions:

- **Q1:** Does ARK reduce per-flow flow completion time (FCT) compared to vanilla ECMP, and how does the gap scale with load? (§4.2)
- **Q2:** Does ARK improve fabric utilization by distributing flows more evenly across spines? (§4.2)
- **Q3:** How do network-level FCT improvements translate to user-visible end-to-end serving latency (TTFT)? (§4.3)
- **Q4:** What is ARK's runtime overhead, and is it negligible compared to KV transmission time? (§4.4)

4.1 Experimental Setup

Simulation platform: SimAI [27] models compute through Vidur [1], which predicts prefill and decode latency from RandomForest regressors trained on per-layer LLaMA-3-8B profiles measured on



(a) Mean FCT (b) P95 FCT

Figure 2: Mean and P95 flow completion time (FCT) as a function of request rate (QPS) under Poisson arrival. ARK consistently reduces both mean and tail FCT, with the largest improvement at QPS = 25 (mean FCT − 27.3%, P95 − 34.0%).

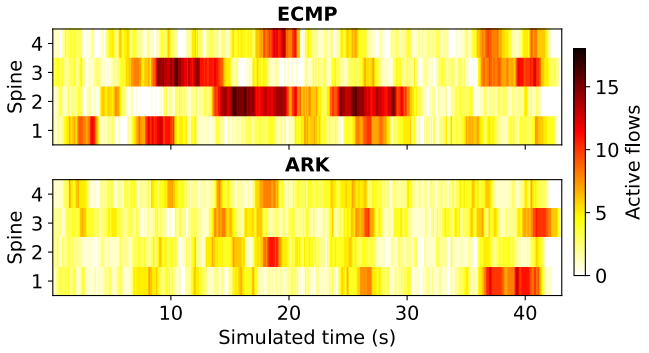
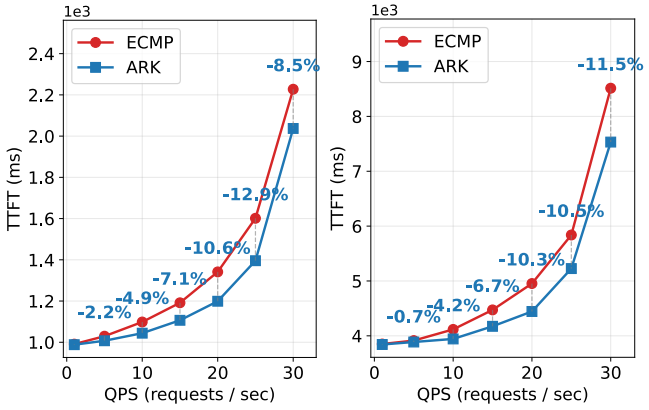


Figure 3: Per-spine active-flow heatmap at QPS = 25 (top: ECMP, bottom: ARK). ECMP drives persistent hotspots of up to 18 concurrent flows onto Spines 2 and 3 while Spines 1 and 4 stay lightly loaded; ARK keeps peaks below ∼8 across all four spines.



(a) Mean TTFT (b) P95 TTFT

Figure 4: TTFT across the same QPS sweep as Fig. 2.

an NVIDIA H100 GPU (varying token count, batch size, and prefill chunk size). KV cache transfers are simulated at packet level on the ECMP-based NS-3 backend from AstraSim [22], on top of which we implement the ARK path reservation controller as a host-side application.

Topology: We use a two-tier spine-leaf Clos topology [2], matching the architecture published by Alibaba HPN [19]. The specific instance we evaluate has 16 GPUs across two 8-GPU servers, two leaf switches, and four spines, with each GPU-leaf and leaf-spine link running at 100 Gbps. This yields 800 Gbps aggregate ingress per leaf against 400 Gbps egress to the spines, creating a 2:1 oversubscription ratio that amplifies the impact of ECMP collisions under elephant-flow workloads.

Workload: We model disaggregated LLaMA-3-8B inference (chosen for its public H100 latency profile) with Server 1 (GPU 1-8) as prefill nodes and Server 2 (GPU 9-16) as decode nodes; each flow is assigned a random prefill-decode pair. Prompt token counts are drawn from a Zipf distribution with $\alpha = 0.7$, capturing the heavy-tailed prompt-size pattern of real LLM serving traces and emphasizing the elephant-flow regime ARK targets. The mean of ∼13,300 tokens yields KV cache flows from 125 MB to 6.7 GB (mean 1.6 GB) at 128 KB per token [6]. Requests arrive as a Poisson process with varying queries per second (QPS). Each experiment contains 1000 requests.

4.2 Multi-Flow Evaluation with Realistic Workload

To evaluate the effectiveness of ARK, we first conduct experiments individually on the NS-3 network backend, with either ARK or ECMP configured. We replay the LLM inference request trace and inject the request to NS-3 as flows, where the flow size is equivalent to the KV cache that the request generates.

QPS Sensitivity Analysis: We sweep the request rate from QPS = 1 to 30 under Poisson arrival (Fig. 2). At QPS = 1, flows rarely overlap and ECMP collisions are unlikely, so ARK provides only marginal FCT reduction (−1.9% mean, −1.2% P95). As QPS increases, ARK delivers progressively larger improvements by distributing elephant flows across spines, peaking at QPS = 25 with 27.3% mean and 34.0% P95 reduction. At QPS = 30 the leaf–spine fabric approaches saturation, so the benefit narrows to 13.8% mean / 12.4% P95.

Spine Utilization: Fig. 3 visualizes the number of concurrent flows on each spine switch over time at QPS = 25. With ECMP, hash collisions concentrate load on Spines 2 and 3 (peaking at 18 concurrent flows) while Spines 1 and 4 remain intermittently idle. This imbalance directly drives the FCT degradation observed in Fig. 2, as flows routed onto a congested 100 Gbps spine experience queueing delays proportional to the number of competing flows sharing that link. ARK, in contrast, spreads load uniformly across all four spines, leading to hotspot reduction and 34.0% P95 FCT improvement.

Table 1: ARK's controller microbenchmarks on an Intel i9 CPU (single thread, single core), and a reference KV transmission time.

Phase / Operation	Time
Microbenchmarks (per call)	
Hash computation	4.10 μs
Reservation-table lookup	0.87 μs
Boot phase (one-time per peer)	
Initialize ($\delta_{\max} = 2^{16}$)	~326 ms
Runtime per flow (worst case)	
OnFlowArrival at $K = 4$ paths	≤3.48 μs
OnFlowArrival at $K = 1000$ paths	≤870 μs
Reference	
1.6 GB KV cache transfer at 100 Gbps	~137 ms
K=4 reservation / KV transfer ratio	≈0.0025%

4.3 End-to-End Serving Evaluation

We translate the network-layer FCT improvements into user-visible serving latency. For each request,

$$\text{TTFT} = T_{\text{prefill}}(L) + T_{\text{KV}} + T_{\text{decode-1}}(L),$$

where L is the prompt token count, T_{KV} is the per-flow NS-3 FCT, and T_{prefill}, $T_{\text{decode-1}}$ are predicted from the Vidur Llama-3-8B with H100 profiling table.

Figure 4 extends the QPS sweep of Fig. 2 from network FCT to user-visible TTFT. The advantage of ARK widens steadily with load: a marginal 2% reduction in mean TTFT at QPS = 5 grows to roughly 13% on the mean (QPS = 25) and 12% on the P95 tail (QPS = 30). Prefill (~840 ms) and the first decode step (~10 ms) contribute a fixed offset to every request; only T_{KV} scales with QPS, so as load rises the relative TTFT gain converges toward the FCT gain. TPOT, dominated by GPU-local per-token decode, is unaffected by the routing scheme.

4.4 Computation Time

We measure ARK's path reservation controller on an Intel i9 CPU (single thread, single core); Table 1 summarizes the results. Hash computation takes 4.10 μs per call and a path-table lookup 0.87 μs. Initialize populates Eq. (3) with $\delta_{\max} = 2^{16}$ src-port offsets per peer at ~326 ms per peer, paid once at boot. At runtime, OnFlowArrival probes at most K paths in $\mathcal{R}$; the $\mathcal{M}$ access is a static array dereference. The per-flow cost is therefore at most $4 \times 0.87 = 3.48$ μs at $K = 4$ and scales linearly to ~870 μs at $K = 1000$, well under 1% of the ~137 ms needed to transmit a 1.6 GB KV cache flow at 100 Gbps. The controller overhead is therefore negligible compared to KV cache transmission and supports real-time KV cache flow orchestration.

5 Conclusion

Prefill–decode disaggregation in modern LLM serving has elevated KV cache transfers to a scale that traditional data-center fabrics were never designed to absorb. We presented ARK, a distributed collision-avoidance framework that pairs (i) a path-reservation controller steering KV cache flows onto non-overlapping spines to reduce per-flow FCT and improve spine utilization, with (ii) a deployment that separates lightweight reservation broadcasts from KV cache transfers, enabling deployment on commodity switches. Experiments show ARK reduces mean FCT by up to 27.3% and P95 FCT by 34.0%, translating to up to 12.9% mean TTFT reduction in end-to-end serving.

Ethics Statement

This work does not raise any ethical concerns. It relies entirely on packet-level network simulation with synthetically generated workloads, and involves no human subjects, user data, or experiments on production systems.

References

[1] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S. Gulavani, Ramachandran Ramjee, and Alexey Tumanov. 2024. Vidur: A Large-Scale Simulation Framework for LLM Inference. In *Proceedings of the 7th Conference on Machine Learning and Systems (MLSys) (Proceedings of Machine Learning and Systems, Vol. 6)*. mlsys.org, Santa Clara, CA, USA, 351–366. https://proceedings.mlsys.org/paper_files/paper/2024/hash/b74a8de47d2b3c928360e0a011f48351-Abstract-Conference.html

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*. ACM, Seattle, WA, USA, 63–74. doi:10.1145/1402958.1402967

[3] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)*. USENIX Association, San Jose, CA, USA, 281–296. https://www.usenix.org/conference/nsdi10-0/hedera-dynamic-flow-scheduling-data-center-networks

[4] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. ACM, Chicago, IL, USA, 503–514. doi:10.1145/2619239.2626316

[5] Jiamin Cao, Yu Guan, Kun Qian, Jiaqi Gao, Wencong Xiao, Jianbo Dong, Binzhang Fu, Dennis Cai, and Ennan Zhai. 2024. Crux: GPU-Efficient Communication Scheduling for Deep Learning Training. In *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM, Sydney, NSW, Australia, 1–15. doi:10.1145/3651890.3672239

[6] Yihua Cheng, Yuhan Liu, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Kuntai Du, and Junchen Jiang. 2025. LMCache: An Efficient KV Cache Layer for Enterprise-Scale LLM Inference. arXiv preprint arXiv:2510.09665. doi:10.48550/arXiv.2510.09665

[7] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed AI Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM, Sydney, NSW, Australia, 15 pages. doi:10.1145/3651890.3672233

[8] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2018. Exploiting a Natural Network Effect for Scalable, Fine-grained Clock Synchronization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*. USENIX Association, 81–94.

[9] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024. MemServe: Context Caching for Disaggregated LLM Serving with Elastic Memory Pool. arXiv preprint arXiv:2406.17565. https://arxiv.org/abs/2406.17565

[10] IEEE. 2019. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008). doi:10.1109/IEEESTD.2020.9120376

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. ACM, Koblenz, Germany, 611–626. doi:10.1145/3600006.3613165

[12] Yiran Lei, Dongjoo Lee, Liangyu Zhao, Daniar Kurniawan, Chanmyeong Kim, Heetaek Jeong, Changsu Kim, Hyeonseong Choi, Liangcheng Yu, Arvind Krishnamurthy, Justine Sherry, and Eriko Nurvitadhi. 2026. FAST: An Efficient Scheduler for All-to-All GPU Communication. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. USENIX Association, Renton, WA, USA, 2515–2523. https://www.usenix.org/conference/nsdi26/presentation/lei-yiran

[13] Yuliang Li, Rui Miao, Mohammad Alizadeh, and Minlan Yu. 2020. Sundial: Fault-tolerant Clock Synchronization for Datacenters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. USENIX Association, 1171–1186.

[14] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM, Sydney, NSW, Australia, 38–56. doi:10.1145/3651890.3672274

[15] NVIDIA. 2025. NVIDIA Dynamo: A Datacenter-Scale Distributed Inference Serving Framework. Open-source project, https://github.com/ai-dynamo/dynamo. Announced at NVIDIA GTC 2025; accessed 2026-04-30.

[16] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, Buenos Aires, Argentina, 118–132. doi:10.1109/ISCA59077.2024.00019

[17] Jonathan Perry, Hari Balakrishnan, and Devavrat Shah. 2017. Flowtune: Flowlet Control for Datacenter Networks. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*. USENIX Association, Boston, MA, USA, 421–436. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/perry

[18] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A Centralized "Zero-Queue" Datacenter Network. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. ACM, Chicago, IL, USA, 307–318. doi:10.1145/2619239.2626309

[19] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. 2024. Alibaba HPN: A Data Center Network for Large Language Model Training. In *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM, Sydney, NSW, Australia, 691–706. doi:10.1145/3651890.3672265

[20] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. Mooncake: Trading More Storage for Less Computation — A KVCache-centric Architecture for Serving LLM Chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. USENIX Association, Santa Clara, CA, USA, 155–170. https://www.usenix.org/conference/fast25/presentation/qin

[21] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: Congestion Signals are Simple and Effective for Network Load Balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. ACM, Amsterdam, Netherlands, 207–218. doi:10.1145/3544216.3544226

[22] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2020. ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Boston, MA, USA, 81–92. doi:10.1109/ISPASS48437.2020.00018

[23] SGLang Team. 2025. Deploying DeepSeek with PD Disaggregation and Large-Scale Expert Parallelism on 96 H100 GPUs. LMSYS Blog Post, https://www.lmsys.org/blog/2025-05-05-large-scale-ep/. Accessed 2026-04-30.

[24] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network Load Balancing with In-network Reordering Support for RDMA. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. ACM, New York, NY, USA, 816–831. doi:10.1145/3603269.3604849

[25] The llm-d Project. 2025. llm-d: Kubernetes-Native Distributed Inference for Large Language Models. Open-source project, https://llm-d.ai. Accessed 2026-04-30.

[26] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*. USENIX Association, Boston, MA, USA, 407–420. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini

[27] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, Kunling He, Jiaqi Gao, Ennan Zhai, Dennis Cai, and Binzhang Fu. 2025. SimAI: Unifying Architecture Design and Performance Tuning for Large-Scale Large Language Model Training with Scalability and Precision. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, USA, 541–558. https://www.usenix.org/conference/nsdi25/presentation/wang-xizheng-simai

[28] Yongtong Wu, Shaoyuan Chen, Yinmin Zhong, Rilin Huang, Yixuan Tan, Wentao Zhang, Liyue Zhang, Shangyan Zhou, Yuxuan Liu, Shunfeng Zhou, Mingxing Zhang, Xin Jin, and Panpan Huang. 2026. DualPath: Breaking the Storage Bandwidth Bottleneck in Agentic LLM Inference. arXiv preprint arXiv:2602.21548. doi:10.48550/arXiv.2602.21548

[29] Shengnan Yue, Mowei Wang, Yu Yan, Weiqiang Cheng, Zihan Jiang, and Zhenhui Zhang. 2025. RTT- or Bandwidth-Bound? Demystifying the KV Cache Transfer in Large Language Model Serving. In *Proceedings of the 2nd Workshop on Networks for AI Computing* (Coimbra, Portugal) *(NAIC '25)*. Association for Computing Machinery, New York, NY, USA, 5–7. doi:10.1145/3748273.3749196

[30] 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. 2024. SGLang: Efficient Execution of Structured Language Model Programs. *Advances in Neural Information Processing Systems* 37 (2024), 62557–62583.

[31] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. 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, Santa Clara, CA, USA, 193–210. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin