EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

JANUS: Joint Prefill/Decode Disaggregation with KV-Cache-Aware Multi-Cloud Routing for Edge-Adjacent LLM Serving

K.B. Aruna[1,*], V. Kaliraj[1], L. Sudha[1], V. Sureka[1]

[1]Department of Computer Science and Engineering, R.M.D. Engineering College, Tamil Nadu, India

Abstract

INTRODUCTION: Disaggregated large language model (LLM) serving separates the compute-bound prefill phase from the memory-bound decode phase and is increasingly deployed across heterogeneous multi-cloud and edge-adjacent fleets serving geographically distributed (including IoT and edge) clients. The key-value (KV) cache that couples the two phases raises a stateful routing problem—migrate, recompute, or partially ship the cache across inter-cloud links of varying bandwidth, on hardware of varying capability, under spot prices that change every few minutes—that, to our knowledge, no published framework fully addresses.
OBJECTIVES: To jointly optimize prefill placement, decode placement, KV-cache transport policy, and slow-timescale pool sizing across clouds with heterogeneous link bandwidths, GPU capabilities, and volatile spot prices, with explicit provable guarantees.
METHODS: We present Janus, an online scheduler that formulates per-request scheduling as a constrained graph-routing problem with stateful edges and decomposes it into a monotone-submodular prefix-aware placement subproblem and a Lyapunov drift-plus-penalty control subproblem, with four KV-transport policies including a hybrid layer-pipelined policy admitting a closed-form layer-split optimum. A 17.1K-line prototype implements the scheduling logic; evaluation uses a trace-driven, discrete-event simulator whose timing and cost models are calibrated against measured single-pod microbenchmarks, configured to model a 96-pod (512-GPU) three-cloud, six-region fleet.
RESULTS: We prove a $(1 - 1/e)$ approximation for prefix reuse under continuous greedy (with a 1/2 guarantee for the deployed combinatorial greedy under slack capacity, degrading to 1/3 when heterogeneous KV capacity binds), an $O(1/V)$ gap to the best policy in the decomposed class with $O(V)$ queue bound stated with its explicit additive constants, a sample-path robustness guarantee under adversarially time-varying prices and bandwidth, and a hybrid-transport optimality theorem. In simulation, versus the strongest multi-cloud baseline we construct, Janus attains 3.8× median and 4.6× P99 time-to-first-token reduction, 2.1× goodput, a 71% reuse-capture rate, and 38% cost reduction, with graceful degradation under spot-preemption, WAN-bandwidth-collapse, and region-failure scenarios.
CONCLUSION: Janus is, to our knowledge, the first scheduler to treat the KV-transport decision as a first-class scheduling variable jointly with prefill and decode routing across heterogeneous multi-cloud fleets, with provable guarantees; physical multi-cloud deployment and hardware validation of the simulated results are explicitly left as future work.

Received on 05 June 2026; accepted on 02 August 2026; published on 12 August 2026

Keywords: LLM inference, disaggregated serving, multi-cloud, edge serving, KV-cache, scheduling, submodular optimization, Lyapunov optimization

doi:10.4108/eetiot.13349

*Corresponding author. Email: aruna.cse@rmd.ac.in

1. Introduction

Large language model serving has, in two years, transitioned from a single-replica autoregressive loop

to a finely engineered pipeline. The two dominant developments are (i) *phase disaggregation*, in which the compute-bound prefill stage and the memory-bound decode stage are placed on separate GPU pools to eliminate stragglers and free each phase to run its preferred batching regime [5–7]; and (ii) *prefix-cache-aware routing*, in which requests sharing input prefixes are deliberately routed to the same replica so that the KV cache of the prefix can be reused, often eliminating tens to hundreds of milliseconds of prefill compute [3, 4]. Production-grade systems combining these two ideas report up to 57× TTFT reduction and 2× throughput improvement over round-robin scheduling [3].

A subtler shift, however, is now under way. As open-weight model families scale from early releases [34, 35] to today's frontier checkpoints—with parameter counts and context windows both growing by orders of magnitude—no single cloud reliably has the heterogeneous mix of accelerators an operator wants (H100 for prefill, L4 for cost-sensitive decode, MI300X for memory-bound long-context cases), nor a stable spot-price advantage across the day, nor a region close to every user. *Multi-cloud LLM serving* has become a practical requirement, not a thought experiment. This requirement is sharpened by *edge-adjacent* serving: LLM endpoints increasingly back geographically distributed and IoT/edge clients, whose latency budgets push decode toward cloud regions close to the user while prefill and its large KV state remain wherever spare accelerator capacity is cheapest. The Sky Computing vision [18] anticipated this transition for stateless workloads; LLM serving generalizes the problem and makes it harder, because the KV cache is a per-request megabyte-to-gigabyte-scale piece of state that couples the two phases and tightly constrains where a request can be steered—exactly the coupling that a network- and edge-aware scheduler must resolve.

Existing single-cluster disaggregation systems sidestep this difficulty by placing prefill and decode pools on the same RDMA fabric. Mooncake [7], DistServe [5], Splitwise [6], MemServe [55], and Llumnix [15] all assume high-bandwidth, low-latency intra-cluster KV transfer; SGLang [4] and llm-d [3] assume a single shared radix index. Even the most recent cross-datacenter prefill work [60] externalizes prefill but does not treat the transport mechanism itself as a scheduling variable. None of these systems addresses what happens when the prefill pod is in AWS `us-east-1`, the cheapest decode pod is in GCP `us-central1`, and the WAN goodput between them is 6 Gb/s under contention. The choice of *whether* to migrate the KV cache, *recompute* it at the decode site, ship a *hybrid* subset of layers, or *selectively* ship only the prefix while recomputing the suffix—each with different latency, bandwidth, and dollar cost—is itself a scheduling decision, and it interacts with prefix-cache placement decisions made earlier.

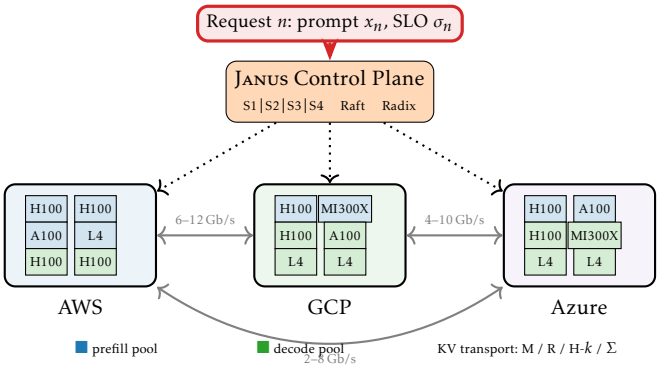

Figure 1. Janus architecture: a Raft-coordinated control plane jointly schedules prefill, decode, and KV transport across three clouds and heterogeneous GPU SKUs

Figure 1 sketches the resulting architecture. This paper presents Janus, an online scheduler that, for the first time to our knowledge, jointly optimizes:

- prefill routing across heterogeneous prefill pools,
- decode routing across heterogeneous decode pools,
- KV-cache transport policy (migrate, recompute, hybrid-k, selective),
- inter-cloud bandwidth and dollar cost,
- heterogeneous GPU utilization with phase-aware batching, and
- long-run serving cost under spot-price volatility.

Contributions. We make four contributions:

C1: Formulation. We formalize multi-cloud disaggregated LLM serving as a constrained graph routing problem with stateful edges (Section 2.1). We show that the offline version is NP-hard via reduction from prefix-constrained generalized assignment, motivating a tractable online decomposition.

C2: Algorithm. Janus decomposes per-request scheduling into four subproblems: a monotone-submodular prefix-aware prefill placement (S1), a KV-transport-aware decode placement (S2), a Lyapunov drift-plus-penalty long-run cost controller (S3), and a slow-timescale pool sizer (S4) (Section 2.2). We introduce four KV-transport policies, including a *hybrid layer-pipelined* policy for which we derive a closed-form layer-split optimum and bound the assumed transfer/compute overlap analytically via an execution-timeline argument and a model-derived analysis (Section 2.4).

C3: Analysis. We prove, with explicit assumptions stated, (i) a $(1 - 1/e)$ approximation for the prefix-reuse subproblem under continuous greedy (and a $\frac{1}{2}$ bound for the deployed combinatorial greedy when capacity is slack, $\frac{1}{3}$ when it binds), (ii) an $O(1/V)$ gap to the best policy in the decomposed class with $O(V)$ queue bound for the Lyapunov controller, together with an explicit, V-independent characterization of the residual decomposition gap, (iii) a sample-path robustness bound under adversarially time-varying prices and bandwidth, and (iv) a hybrid-transport optimality theorem (Section 2.3).

C4: Prototype & simulation study. We implement the JANUS scheduling logic (S1–S4) and a prototype data plane, and evaluate the system in a trace-driven, discrete-event simulator whose timing and cost models are calibrated against measured single-pod microbenchmarks (Sections 2.6 and 3.1). The simulator models a 96-pod (512-GPU) fleet across AWS, GCP, and Azure in six regions, driven by a replayed assistant trace. Against the strongest multi-cloud baseline we construct, JANUS delivers, *in simulation*, 3.8× median and 4.6× P99 TTFT improvement, 2.1× goodput, a 71% reuse-capture rate, and 38% cost reduction. We are explicit throughout about which numbers are calibrated measurements (single-pod microbenchmarks) and which are simulator outputs; physical multi-cloud deployment is future work.

The remainder of the paper follows an IMRaD organization: Section 1.1 completes the introduction with background and motivation; Section 2 presents the methods (system model, algorithm, analysis, and implementation); Section 3 reports results from controlled simulation experiments and fault-injection case studies; and Section 4 discusses related work, limitations, and conclusions.

1.1. Background: Prefill and Decode Asymmetry

A transformer [33] inference request proceeds in two phases. *Prefill* processes the entire input prompt of L_{in} tokens in a single forward pass, producing one output token and an attention KV cache whose exact size is derived in Section 2.1 from the model's grouped-query-attention (GQA) [61] configuration. *Decode* then autoregressively emits L_{out} output tokens, reading the cache once per token per layer. Their arithmetic intensities differ by 30–200×: prefill is compute-bound, saturating the tensor cores at large L_{in}, whereas decode is HBM-bandwidth-limited. On identical hardware and with our calibrated coefficients (Appendix E), prefill attains roughly 5–10× the achieved FLOP utilization of decode at the same batch size; we quote this ratio rather than an absolute fraction of vendor peak, since the latter is highly sensitive to kernel version, precision, and sequence length. Co-locating the two phases on the same replica forces a regrettable compromise on batch size, max-tokens, and parallelism, motivating disaggregation [5, 6]. Chunked-prefill schedulers such as Sarathi-Serve [1], memory-efficient attention kernels [8, 14], paged memory management [2], and tensor/pipeline parallelism layouts [10, 31] mitigate but do not eliminate this asymmetry, and they leave the cross-pool transport question largely open.

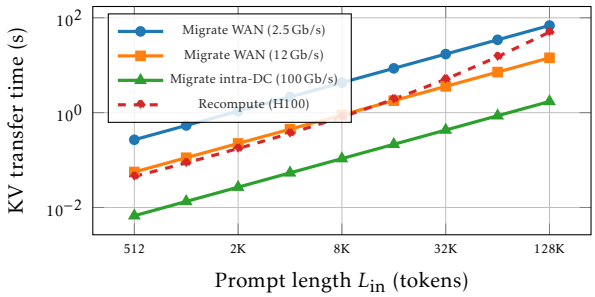

Figure 2. Transport latency for the Llama-3-70B KV cache (FP8, GQA, n_{kv}=8, 160 KiB/token) vs. prompt length, computed from the size model of Section 2.1 and the calibrated coefficients of Table E.1. Intra-DC RDMA migration is far below recompute (migrate always wins); a slow 2.5 Gb/s WAN is far above (recompute always wins); a fast 12 Gb/s WAN crosses the recompute curve near 13K tokens, where recompute's quadratic growth overtakes a linear transfer. The optimal pure policy therefore flips with bandwidth, motivating a per-request choice (and the hybrid interpolation of Section 2.4)

For a Llama-3-70B request with $L_{\text{in}} = 2048$, the FP8 KV cache occupies 320 MiB; for $L_{\text{in}} = 16384$, 2.5 GiB (derivation in Section 2.1 and Appendix F). The best transport choice is regime-dependent. On a fast 12 Gb/s cross-cloud link, shipping the 2.5 GiB cache for a 16K-token prompt takes about 1.8 s—essentially the same as the ∼1.9 s needed to recompute that prefill on an 8×H100 pod—so neither pure policy dominates. Drop the link to 2.5 Gb/s under contention and migration balloons to ∼8.6 s while recompute is unchanged, so recomputation wins decisively; on an intra-region 100 Gb/s RDMA fabric the same cache ships in ∼0.2 s and migration wins. Figure 2 plots these crossovers. The right transport choice depends on L_{in}, W (link goodput), Γ (per-byte transfer cost), and ρ_q (spot price at the decode site). No prior scheduler treats this as a decision variable.

1.2. Background: Prefix Caches, Radix Trees, and Locality

LLM workloads contain massive prefix redundancy: system prompts repeated across thousands of requests, few-shot examples [32], retrieval-augmented contexts. SGLANG [4] and vLLM [11] maintain a per-replica radix tree of cached prefixes; LLM-D [3] replicates a coarsened

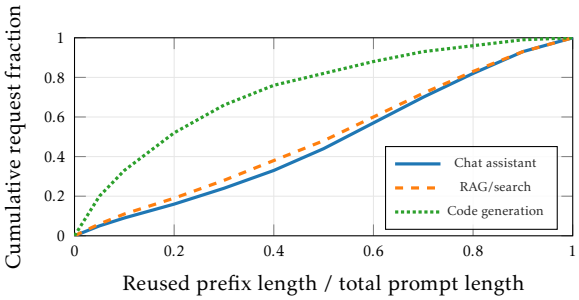

Figure 3. CDF of the per-request prefix-reuse fraction (reused prefix length / total prompt length) across three workloads, *conditioned on requests with a non-empty cached pre ix* (i.e. on the $h = 64\%$ hitting subset for chat). A curve that lies *lower* indicates heavier reuse (more mass at high reuse fraction). Chat and RAG reuse heavily—$1 - F(0.5) = 56\%$ of hitting chat and 52% of hitting RAG requests reuse at least half their prompt—while code generation is sparse, with only 18% reusing half or more

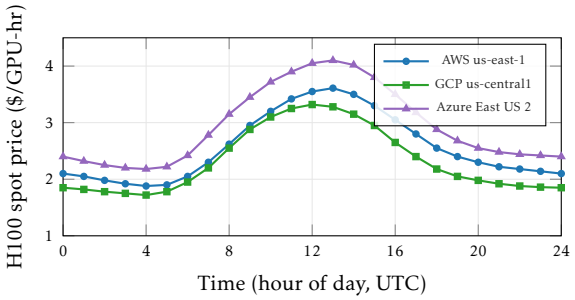

Figure 4. H100 spot price across three clouds for a representative day. Two spreads matter and are of different magnitude: the *inter-cloud* spread at peak reaches \$0.87/GPU-hr (Azure \$4.02 vs. GCP \$3.15 at 14:00 UTC), while the *diurnal* swing within a single cloud reaches \$1.92/GPU-hr (Azure, \$2.18 at 04:00 to \$4.10 at 13:00). Both motivate arbitrage, on different timescales. Source: provider public-listing telemetry, Mar 2026

summary across replicas and routes by longest common prefix; M EMSERVE [55] m aintains a g lobal prompt tree over an elastic memory pool. Complementary techniques compress or stream the cache [39], persist it across multi-turn conversations [40], share it across enterprise deployments [59], or evict it adaptively under memory pressure [41, 42]; these are orthogonal to where the cache should physically live in a multi-cloud fleet.

We distinguish two metrics used throughout the paper, to avoid conflating a p roperty o f t he workload with a property of the scheduler. (i) The *prefix-hit rate* h is a trace property: the fraction of requests whose longest cached prefix i s n on-empty ($|u_n| > 0$), i .e. that *could* reuse some prefix. O n o ur a ssistant t race $h = 64\%$, and among hitting requests the median reused fraction is 649 of 1180 tokens ($\approx 55\%$), consistent with the chat curve of Figure 3. (ii) The *reuse-capture rate* r_{cap} is a scheduler property: of all the reusable prefix tokens present in the trace ($\sum_n |u_n|$), the fraction that a given scheduler actually serves from cache rather than recomputing because of where it placed the request,

$$r_{\text{cap}} = \Big(\sum_n \text{tokens served from cache}\Big) \Big/ \Big(\sum_n |u_n|\Big).$$

An oracle that always routes to a prefix owner attains $r_{\text{cap}} \to 100\%$ (minus staleness); a placement-oblivious scheduler attains far less. Because h counts *requests with any opportunity* while r_{cap} measures *how much of that opportunity is captured*, the two are not comparable and either may exceed the other. Placement-aware scheduling has order-of-magnitude leverage on r_{cap} (Section 3).

In a multi-cloud setting the radix index becomes distributed, replicated approximately by gossip, and is therefore stale to varying degrees. A naive scheduler that treats stale entries as ground truth wastes prefill capacity on a 0-tokens-reused match. Figure 3 shows the reuse-fraction CDF across workload classes.

1.3. Background: Spot Pricing and Heterogeneity

Across AWS, GCP, and Azure between Sep 2024 and Apr 2026, H100 spot prices in equivalent regions diverged by up to 3.1× on hourly horizons and by 1.6× on diurnal averages; these figures come from the full 19-month price archive, not from the single representative day plotted in Figure 4, whose peak inter-cloud ratio is a milder 1.25×. Multi-cloud inference benchmarking corroborates this heterogeneity [9]. Preemption rates over a 24 h window ranged 0.8–7.4% per cloud. Long-term cost minimization therefore demands an arbitrage component that opportunistically shifts capacity, but the KV cache makes naive arbitrage infeasible: relocating a workload means relocating its state.

Figure 4 illustrates a typical day. The instantaneous cross-cloud gap peaks at \$0.87/GPU-hr, and the within-cloud diurnal swing at \$1.92/GPU-hr. A scheduler that statelessly arbitraged on these spreads would steer 41% of decode traffic ac ross cl ouds; do ing so wi thout KV awareness multiplies WAN egress and incurs TTFT regressions that erase the dollar gain.

1.4. Motivation: Why a Joint Scheduler

Single-axis schedulers fail in multi-cloud disaggregation. A prefix-aware-only s cheduler c oncentrates load on the cheapest-prefix p od, o versubscribing i t. A cost-aware-only scheduler ignores the cache and reduces hit rate to 4–8%. A bandwidth-aware-only scheduler can pick a slow-but-cheap link that violates SLO. Empirically, single-axis schedulers underperform by 2.4×–4.1×

on TTFT under our traces. The contribution of Janus is a principled joint formulation that exposes the tradeoffs and resolves them online.

2. Methods

This section presents the system model and problem formulation (Section 2.1), the Janus algorithm (Section 2.2), its theoretical analysis (Section 2.3), the security/privacy/tenancy model (Section 2.5), and the implementation (Section 2.6).

2.1. System Model and Problem Formulation

Every symbol introduced below is defined at first use with explicit units; a complete notation reference appears in Appendix I (Table I.3).

Topology. Let $\mathcal{C}$ denote a set of clouds and $\mathcal{R}$ a set of regions, partitioned per cloud. A *pool* $p \in \mathcal{P}$ is a set of η_p identical *replicas* in one region with phase role $\pi(p) \in \{\textsc{prefill}, \textsc{decode}\}$, GPU SKU $s(p) \in \mathcal{S}$ (e.g., H100, A100, L4, MI300X), and parallelism configuration. Each replica occupies $g_p = t_p\, z_p$ GPUs, where t_p is the tensor-parallel (TP) degree and z_p the pipeline-parallel (PP) degree; thus a pool consumes $g_p\,\eta_p$ GPUs in total. We denote $\mathcal{P}_{\text{pf}}$ and $\mathcal{P}_{\text{dec}}$ for prefill and decode pools, respectively. We summarize a pool by the tuple

$$\theta_p = \big(\pi(p),\, s_p,\, t_p,\, z_p,\, b_{\text{prec},p},\, g_p,\, \eta_p,\, m_p^{\text{gpu}},\, \mathcal{M}_p,\, r_p,\, c_p,\, \rho_p(t)\big)$$

where $b_{\text{prec},p}$ is the KV precision in bytes per element, m_p^{gpu} per-GPU HBM bytes, $\mathcal{M}_p$ supported models, r_p region, c_p cloud, and $\rho_p(t)$ the spot price at time t in \$/GPU-hour. We write the per-GPU-second price as $\tilde{\rho}_p(t) = \rho_p(t)/3600$.

Compatibility. A prefill pool p can hand off to a decode pool q iff: (i) $\mathcal{M}_p \cap \mathcal{M}_q \neq \emptyset$; (ii) $b_{\text{prec},p} = b_{\text{prec},q}$ or a known transcoder exists; and (iii) the layouts admit a reshape between (t_p, z_p) and (t_q, z_q). We write $q \in \mathcal{D}_{\text{compat}}(p)$. When $t_p \neq t_q$ the KV tensor is re-sharded en route; the reshape latency is measured offline and folded into the fixed overhead term ξ_p of the cost model below.

A directed edge $(p, q) \in \mathcal{E}$ between every p and every compatible q has *goodput* $W_{pq}(t)$ in *bytes/s* (i.e., the nominal link rate in Gb/s divided by 8 and multiplied by a measured compression/TLS efficiency factor $c_{pq}^{\text{eff}} \in (0, 1]$), one-way latency $\ell_{pq}(t)$, and egress cost Γ_{pq} in \$/byte. Intra-region edges are RDMA (200–400 Gb/s, sub-100 μs). Inter-region same-cloud is 10–40 Gb/s. Cross-cloud is 2–12 Gb/s after WAN egress, TLS, and compression overhead.

Request Model. A request n arrives at time t_n with prompt tokens x_n of length L_n, requested output tokens L_n^{out}, model M_n, and SLO class σ_n. We decompose $x_n = u_n \cdot v_n$ where u_n is the longest cached prefix in the (distributed, possibly stale) radix index $\mathcal{T}(t)$, and v_n the residual suffix of length $L_n - |u_n|$. Let $\mathcal{T}^{-1}(u_n) \subseteq \mathcal{P}_{\text{pf}}$ be the set of prefill pools believed to hold u_n (the *owner set*). We write $u_n^{(q)}$ for whatever prefix of x_n a specific pool q already holds (possibly empty).

KV-Cache Size Model (GQA-aware). A model is described by its layer count N_{layers}, number of *key/value* heads n_{kv} (which, under grouped-query attention [61], is smaller than the number of query heads), and head dimension d_{head}. The KV cache stores both keys and values, so the cache for a sequence of ℓ tokens is

$$S(\ell) = \underbrace{2}_{K,V} N_{\text{layers}}\, n_{\text{kv}}\, d_{\text{head}}\, \ell\, b_{\text{prec}} \quad \text{bytes}, \tag{1}$$

with per-layer size $S_\ell(\ell) = S(\ell)/N_{\text{layers}} = 2\, n_{\text{kv}}\, d_{\text{head}}\, \ell\, b_{\text{prec}}$. Crucially, (1) uses $n_{\text{kv}} d_{\text{head}}$, *not* the model hidden dimension $d_{\text{model}} = n_q d_{\text{head}}$ (with $n_q \gg n_{\text{kv}}$ for GQA models). For Llama-3-70B (N_{layers}=80, n_{kv}=8, d_{head}=128) at FP8 (b_{prec}=1), (1) gives 160 KiB/token, hence 320 MiB at ℓ=2048 and 2.5 GiB at ℓ=16384, matching the figures used in Section 1.1. Appendix F (Table F.2) tabulates the exact parameters and resulting sizes for all three evaluated models. Under tensor parallelism each replica's GPUs hold $S(\ell)/t_p$ bytes locally, but the aggregate that must traverse an inter-pod edge is the full $S(\ell)$ (re-sharded if $t_p \neq t_q$), transmitted over the aggregate goodput W_{pq}.

Cost Model. *Prefill compute.* The prefill *latency* on pool p for prompt of length L with reused prefix of length $|u|$ is

$$T_p^{\text{pf}}(L, |u|) = \alpha_p \left(L^2 - |u|^2\right) + \beta_p\,(L - |u|) + \xi_p, \tag{2}$$

in seconds, where α_p [s/tok^2], β_p [s/tok], and the fixed launch/reshape overhead ξ_p [s] are SKU-and-parallelism-specific coefficients fitted offline (Appendix E, Table E.1). The linear term captures the projection/FFN work on the $L - |u|$ freshly processed tokens. The quadratic term captures attention's $O(L^2)$ scaling with the cached-prefix correction: when a prefix of length $|u|$ is cached, only the $L - |u|$ suffix tokens are freshly attended, but *each suffix token attends to the full preceding context*, so the recomputed attention work scales as $\sum_{i=|u|+1}^{L} i \approx \frac{1}{2}(L^2 - |u|^2)$, i.e. as $(L^2 - |u|^2)$ and *not* as $(L - |u|)^2$; the latter would undercount by ignoring suffix-to-prefix attention. Two consequences are worth noting. First, at $|u| = 0$ (full prefill) (2) reduces to $\alpha_p L^2 + \beta_p L + \xi_p$, so the coefficients fitted from full prefills (the regime of Table E.1) are unchanged and the validation below still applies. Second, the reuse *benefit* of a cached prefix is the saved work $T_p^{\text{pf}}(L, 0) - T_p^{\text{pf}}(L, |u|) = \alpha_p |u|^2 + \beta_p |u|$, used in (7). We validate the full-prefill form to $R^2 > 0.97$ across H100 TP$\in \{4, 8\}$, A100 TP=4, MI300X TP=4, and L4 TP=8 INT8.

Recompute target. Recomputing a prefill is a compute-bound operation, so Janus dispatches it not to the memory-bound decode replica but to $r(q)$, a *prefill-class* (t_p=8, "8×H100") pool network-local to q; the recomputed KV reaches q over the local RDMA fabric (the intra-DC curve of Figure 2), folded into the overlap. This is both faster than recomputing on the TP=4 decode pool and consistent with the disaggregation premise of running prefill on prefill hardware. We assume each decode region hosts (or is RDMA-adjacent to) a prefill-class pool, which holds on our modeled fleet—every decode region in Table 2 co-locates one of the 24 prefill H100 pods; when no such pool is network-local to q, $r(q)$ resolves to the nearest prefill-class pool and its inbound hop is charged at the corresponding (possibly inter-region) transport cost rather than the intra-DC rate, so the scheduler simply sees a more expensive Recompute/Hybrid option. Accordingly, define the per-layer recompute rate

$$\Omega_q \;\equiv\; T^{\mathrm{pf}}_{r(q)}\Big(L_n,\, |u_n^{(q)}|\Big)\Big/ N_{\mathrm{layers}} \quad [\mathrm{s/layer}], \tag{3}$$

evaluated with the recompute target's coefficients (the TP=8 row of Table E.1 in our configuration, giving $\Omega_q \approx 10.1$ ms/layer at 8K with $|u_n^{(q)}|=0$).

Transport policies. The decode pool q requires the KV cache of the full prompt; pool p has just materialized it. Given (p, q), write $S \equiv S(L_n)$, $S_\ell \equiv S_\ell(L_n)$, and $W \equiv W_{pq}$. We define four policies, each with latency Λ_ϕ [s] and dollar cost Ψ_ϕ [$]; recompute terms are priced at the recompute target $r \equiv r(q)$ (network-local, so at q's regional spot rate):

- **Migrate** (M): ship the full cache. $\Lambda_M = S/W$; $\Psi_M = \Gamma_{pq} S$.
- **Recompute** (R): discard; recompute prefill on r, streaming each layer's KV to q over the local RDMA edge as it is produced, so only the final layer's intra-DC hop is unoverlapped. $\Lambda_R = T^{\mathrm{pf}}_r(L_n, |u_n^{(q)}|) + S_\ell/W_{rq}$, where the trailing hop S_ℓ/W_{rq} is sub-millisecond on a ≥200 Gb/s RDMA edge; $\Psi_R = \tilde{\rho}_r\, g_r\, T^{\mathrm{pf}}_r(L_n, |u_n^{(q)}|)$.
- **Hybrid-k** (H_k): ship the first k layers and recompute the remaining $N_{\mathrm{layers}} - k$ on r, overlapped via layer pipelining (Section 2.4). Per-layer transfer time $a = S_\ell/W$, per-layer recompute time $b = \Omega_q$. Under ideal overlap $\Lambda_H = \max(k\,a,\, (N_{\mathrm{layers}} - k)\, b)$; $\Psi_H = \Gamma_{pq}\, k\, S_\ell + \tilde{\rho}_r\, g_r\, (N_{\mathrm{layers}} - k)\, \Omega_q$.
- **Selective** (Σ): partition the KV that q lacks into two *disjoint* segments—a prefix segment u_n (of length $|u_n|$) whose KV p already holds and which is shipped, and a trailing suffix segment v_n (of length $|v_n|$) that is recomputed on r—and run the two in parallel. The suffix recompute is *not* a standalone $|v_n|$-token prefill: the final $|v_n|$ tokens attend to the full preceding context of length $L_n - |v_n|$ (whose KV r obtains over its local RDMA edge, overlapped with the WAN ship of u_n), so its cost is $T^{\mathrm{pf}}_r(L_n,\, L_n - |v_n|)$. Hence

$$\Lambda_\Sigma = \max\Big(S(|u_n|)/W,\; T^{\mathrm{pf}}_r(L_n,\, L_n - |v_n|)\Big),$$

$\Psi_\Sigma = \Gamma_{pq} S(|u_n|) + \tilde{\rho}_r g_r\, T^{\mathrm{pf}}_r(L_n,\, L_n - |v_n|)$. Because u_n and v_n are disjoint, Σ never ships and recomputes the same tokens; it is attractive precisely when q already caches most of the prompt, so both the shipped prefix and the recomputed suffix are small, and it degenerates to M (when $|v_n|$=0) or R (when $|u_n|$=0) at the boundaries.

Scheduling Decision. For each arrival n, the scheduler chooses

$$\Xi_n = (p_n, q_n, \phi_n) \in \mathcal{P}_{\mathrm{pf}} \times \mathcal{P}_{\mathrm{dec}} \times \{M, R, H_k, \Sigma\},$$

with $q_n \in \mathcal{D}_{\mathrm{compat}}(p_n)$. The end-to-end TTFT includes the client-to-prefill round trip and the queueing delay at *both* the prefill and decode pools:

$$\mathrm{TTFT}_n = \underbrace{\mathrm{RTT}_n}_{\text{client}\to p_n} + \underbrace{\delta^{\mathrm{pf}}_n(p_n)}_{\text{prefill queue}} + T^{\mathrm{pf}}_{p_n}(L_n, |u_n|) + \Lambda_{\phi_n} + \underbrace{\delta_n(q_n)}_{\text{decode queue}}, \tag{4}$$

where RTT_n [s] is the client→prefill round-trip time, $\delta^{\mathrm{pf}}_n(p_n)$ [s] the prefill-pool queueing delay, and $\delta_n(q_n)$ [s] the decode-pool queueing delay. The instantaneous dollar cost, with all rates in $/GPU-second and g the GPUs per replica, is

$$\Psi_n = \tilde{\rho}_{p_n} g_{p_n} T^{\mathrm{pf}}_{p_n} + \Psi_{\phi_n} + \tilde{\rho}_{q_n} g_{q_n} \left(\Lambda_{\phi_n} + \frac{L^{\mathrm{out}}_n}{\mu_{q_n}}\right), \tag{5}$$

where μ_{q_n} [tok/s] is the per-replica decode throughput of q_n under the request's batch.

Optimization Problem. Let Π be a scheduling policy and $\tau^\star_{\sigma_n}$ the TTFT target of SLO class σ_n, with per-class violation budget ϵ_{σ_n}. With $m^{\mathrm{KV}}_n = S(L_n)$ the per-request KV footprint and $\kappa_p \in (0, 1]$ the fraction of pool HBM usable for KV (after weights/activations), the operator's problem is

$$\begin{aligned} \min_{\Pi} \quad & \lim_{T\to\infty} \frac{1}{T} \sum_{n:\, t_n < T} \mathbb{E}[\Psi_n] \\ \text{s.t.} \quad & \Pr[\mathrm{TTFT}_n > \tau^\star_{\sigma_n}] \le \epsilon_{\sigma_n} \quad \forall \sigma, \\ & \sum_{n \in A_p(t)} m^{\mathrm{KV}}_n \le \kappa_p\, \eta_p\, g_p\, m^{\mathrm{gpu}}_p \quad \forall p, t, \\ & q_n \in \mathcal{D}_{\mathrm{compat}}(p_n), \;\; \mathrm{residency}(n) \models r_{q_n} \quad \forall n, \end{aligned} \tag{6}$$

where $A_p(t)$ are active requests at p, m^{gpu}_p is per-GPU HBM, and $\mathrm{residency}(n) \models r_{q_n}$ enforces any data-residency constraint of request n (Section 2.5).

Hardness.

Theorem 1 (Hardness). The offline version of (6) is NP-hard, even when prices and bandwidths are static and all SLO classes are identical.

Proof sketch. Reduce from the generalized assignment problem (GAP). Given a GAP instance with n items, m bins, sizes w_{ij}, capacities C_j, and profits p_{ij}, construct one decode pool per bin (HBM capacity C_j), one prefill pool holding all prefixes, and identify each item with a request whose KV footprint and routing cost realize w_{ij} and p_{ij} when assigned to pool j. To neutralize the SLO/TTFT constraint of (6) we place all requests in a single SLO class and set its target $\tau^\star$ larger than the maximum realizable TTFT in the constructed instance (finite because prices, bandwidths, sizes, and pool counts are all finite), so $\Pr[\mathrm{TTFT} > \tau^\star] = 0$ for every assignment and the probabilistic constraint is slack; feasibility is then governed solely by the capacity constraints and the objective by routing cost. A feasible min-cost assignment respecting capacity thus solves GAP, which is NP-hard. The prefix-reuse structure (a submodular benefit on the prefill side, Section 2.2) only adds coupling and does not reduce hardness. □

This rules out exact online solutions and motivates the decomposition in Section 2.2.

2.2. Algorithm Design

Janus decomposes the problem along two timescales and two axes:

Fast (per micro-batch, ∼ms). Subproblems S1 (prefill placement) and S2 (decode placement + transport policy).

Slow (epoch, 30 s). Subproblems S3 (Lyapunov queue control) and S4 (pool sizing).

S1: Submodular Prefix-Aware Prefill Placement. S1 operates on a micro-batch $\mathcal{B}$ of pending arrivals (those queued within a short window, default 5 ms, or the current admission backlog). Define the *prefix-reuse benefit* of placing request n on prefill pool p as the prefill compute time it saves:

$$F_n(p) = \begin{cases} \alpha_p |u_n|^2 + \beta_p |u_n| & p \in \mathcal{T}^{-1}(u_n), \\ 0 & \text{otherwise}, \end{cases} \tag{7}$$

which is the prefill compute saved by reusing the cached prefix (Section 2.1, $T_p^{\mathrm{pf}}(L_n, 0) - T_p^{\mathrm{pf}}(L_n, |u_n|)$) and is non-negative. The ground set is $\mathcal{X} = \{(n, p) : n \in \mathcal{B},\ p \in \mathcal{P}_{\mathrm{pf}},\ \mathcal{M}_p \ni M_n\}$. A solution $\mathcal{A} \subseteq \mathcal{X}$ assigns requests to prefill pools, and the *reuse objective* is

$$G(\mathcal{A}) = \sum_{n \in \mathcal{B}} \max_{p:\,(n,p)\in\mathcal{A}} F_n(p), \qquad G(\emptyset) = 0. \tag{8}$$

The feasible region is the intersection of (i) a *partition matroid* $\mathcal{M}_1$ enforcing at most one prefill pool per request, and (ii) per-pool capacity constraints $\sum_{n:(n,p)\in\mathcal{A}} m_n^{\mathrm{KV}} \le \kappa_p \eta_p g_p m_p^{\mathrm{gpu}}$.

Cost and queue awareness. Rather than subtracting the (modular) dollar cost inside G—which preserves submodularity but *destroys monotonicity*—we incorporate cost and queue pressure as an *admission threshold*: an element (n, p) is eligible only if its marginal reuse gain exceeds its Lyapunov-weighted penalty,

$$\Delta_G(n, p \,|\, \mathcal{A}) \;\ge\; V\, \tilde{\rho}_p\, g_p\, T_p^{\mathrm{pf}}(L_n, |u_n|) \;+\; Q_p\, \bar{\delta}_p, \tag{9}$$

where $V > 0$ is the *price weight* in *seconds per dollar*—the latency the operator is willing to trade to save one dollar. This makes the threshold dimensionally closed: the left side Δ_G is a reuse gain in seconds, and on the right $V\,[\mathrm{s/\$}] \cdot (\tilde{\rho}_p g_p T_p^{\mathrm{pf}})\,[\$]$ is in seconds, while Q_p is the dimensionless congestion price from S3 and $\bar{\delta}_p$ [s] the pool's mean queueing delay, so $Q_p \bar{\delta}_p$ is also in seconds. Greedy selection over eligible elements then maximizes the monotone-submodular G over the matroid while remaining cost- and congestion-aware; the approximation guarantee in Section 2.3 is stated for G.

Lemma 1 (Monotone submodularity of the reuse objective). G in (8) is monotone and submodular on $2^{\mathcal{X}}$.

Proof. Fix a request n and let $g_n(\mathcal{A}) = \max_{p:(n,p)\in\mathcal{A}} F_n(p)$ with $g_n(\emptyset) = 0$. For sets $\mathcal{A} \subseteq \mathcal{A}'$ and an element $e = (n, p^\star) \notin \mathcal{A}'$, the marginal gain is $\big(F_n(p^\star) - g_n(\mathcal{A})\big)^+$, which is non-negative (monotonicity) and non-increasing in the base set because $g_n(\mathcal{A}) \le g_n(\mathcal{A}')$ (diminishing returns, i.e. submodularity). The maximum of non-negative weights over a chosen set is a standard monotone-submodular (weighted-coverage) function. Since the per-request ground sets $\{(n, \cdot)\}$ are disjoint and $G = \sum_n g_n$, and a finite sum of monotone-submodular functions over disjoint ground sets is monotone submodular, the claim follows. □

Proposition 1 (Approximation of S1). Consider the reuse objective G under the partition matroid $\mathcal{M}_1$. (i) When per-pool KV capacity is slack (the common case), $\mathcal{M}_1$ is the only binding constraint and the combinatorial (lazy) greedy deployed on the fast path returns $\mathcal{A}_{\mathrm{g}}$ with $G(\mathcal{A}_{\mathrm{g}}) \ge \frac{1}{2} G^\star$ [54], while the continuous-greedy algorithm with pipage/swap rounding attains $\mathbb{E}[G(\mathcal{A}_{\mathrm{cg}})] \ge (1 - 1/e)\, G^\star$ [53], which is optimal under a matroid constraint unless P=NP. (ii) When heterogeneous per-pool KV capacity binds, the capacity constraint is a *knapsack* (the footprints m_n^{KV} differ across requests), not a matroid. We then rely on the conservative $\frac{1}{3}$ greedy guarantee for one matroid plus one knapsack; the local-search algorithm of Lee, Mirrokni, Nagarajan,

and Sviridenko [62] improves the achievable factor to $1/(2+\epsilon)$ in this constraint class. We state $\frac{1}{3}$ as the figure we rely on, and note that any sharper constant for this class only strengthens the downstream bounds.

Scope of the guarantee under the admission threshold. Proposition 1's greedy guarantee is stated for greedy on a *fixed* ground set, whereas the eligibility test (9) restricts the ground set. We therefore evaluate eligibility *once, at batch-formation time*, against the fixed marginal $\Delta_G(n, p \mid \emptyset)$, yielding a fixed eligible ground set $\mathcal{X}_{\text{elig}} \subseteq \mathcal{X}$ to which the $\frac{1}{2}$ (resp. $1 - 1/e$) bound applies, now relative to $G^\star$ over $\mathcal{X}_{\text{elig}}$. (The alternative of re-evaluating eligibility *during* greedy, as the base set grows, is a heuristic we do not claim a bound for.) *Fallback.* A request none of whose placements is eligible—or for which no owner pool exists—is not dropped: it is assigned to the feasible prefill pool minimizing the right-hand side of (9) (i.e. the cheapest compatible pool, incurring a full prefill), guaranteeing every request is placed. On our traces the fast-path greedy and the continuous-greedy reference produce reuse objectives within 3.4% of each other, and capacity binds on fewer than 4% of micro-batches at our operating utilization; the headline $\frac{1}{2}$ figure therefore describes the regime in which the system almost always runs, and the $\frac{1}{3}$ worst case applies only under sustained $> 95\%$ HBM pressure. Discretizing admission into equal KV slots would restore a true second matroid (and the $\frac{1}{2}$ two-matroid-intersection bound) at the cost of some packing efficiency; we do not impose it, preferring the knapsack with the stated guarantee.

S2: KV-Aware Decode and Transport Selection. Given prefill placement p_n, S2 selects (q_n, ϕ_n) to minimize

$$J_n(q, \phi) = \Lambda_\phi + V\,\Psi_\phi + Q_q(t)\,\delta_n(q) \tag{10}$$

over $q \in \mathcal{D}_{\text{compat}}(p_n)$ (further restricted to residency-compatible regions) and $\phi \in \{M, R, H_k, \Sigma\}$. The objective is expressed in the seconds currency: Λ_ϕ [s] is the transport latency, $V\,\Psi_\phi$ converts the dollar cost Ψ_ϕ [\$] via the price weight V [s/\$] of (9), and $Q_q(t)$ is the dimensionless virtual-queue congestion price from S3 multiplying the marginal queueing delay $\delta_n(q)$ [s]. Larger V therefore prioritizes dollar cost (consistent with the Pareto sweep of Section 3). The hybrid-k branch admits a closed-form optimum.

Theorem 2 (Hybrid layer-split optimum). With per-layer transfer time $a = S_\ell/W$ and per-layer recompute time $b = \Omega_q$, the real-valued latency-minimizing layer split for H_k on edge (p, q) is

$$k^\star_{\mathbb{R}} = \frac{N_{\text{layers}}\,b}{a+b} = \frac{N_{\text{layers}}\,\Omega_q\,W}{S_\ell + \Omega_q\,W}, \tag{11}$$

and the deployed integer split is obtained by evaluating Λ_H at the two neighbouring integers,

$$k^\star = \operatorname*{arg\,min}_{k \in \{\lfloor k^\star_{\mathbb{R}} \rfloor, \lceil k^\star_{\mathbb{R}} \rceil\} \cap [0, N_{\text{layers}}]} \max\Big(ka,\ (N_{\text{layers}} - k)b\Big). \tag{12}$$

At the real optimum the two lines cross exactly: $\Lambda_H(k^\star_{\mathbb{R}}) = k^\star_{\mathbb{R}}\,a = (N_{\text{layers}} - k^\star_{\mathbb{R}})\,b \le \min(\Lambda_M, \Lambda_R)$. For the integer split the equality no longer holds, so $\Lambda_H(k^\star) = \max(k^\star a, (N_{\text{layers}} - k^\star)b) \le \Lambda_H(k^\star_{\mathbb{R}}) + \max(a, b)$. Accounting additionally for one stage of pipeline fill/drain, the realizable latency satisfies $\Lambda_H \le \max(k^\star a, (N_{\text{layers}} - k^\star)b) + \max(a, b)$.

Proof. $\Lambda_H(k) = \max(ka, (N_{\text{layers}} - k)b)$ is the pointwise maximum of an increasing line (ka) and a decreasing line ($(N_{\text{layers}} - k)b$); over real k its minimum lies at their crossing $ka = (N_{\text{layers}} - k)b$, giving $k^\star_{\mathbb{R}}$ in (11). Substituting $a = S_\ell/W$ yields the closed form. Since Λ_H is unimodal (decreasing then increasing) in k, its minimizer over the integers lies in $\{\lfloor k^\star_{\mathbb{R}} \rfloor, \lceil k^\star_{\mathbb{R}} \rceil\}$, which is exactly the set searched in (12); evaluating both is therefore *exact* over the integers, and moving from $k^\star_{\mathbb{R}}$ to either neighbour changes Λ_H by at most the larger slope, $\max(a, b)$. (Taking $\lfloor k^\star_{\mathbb{R}} \rfloor$ alone would be feasible but not integer-optimal; the two-point search costs one extra comparison.) The bounds follow since $k^\star_{\mathbb{R}} \le N_{\text{layers}} \Rightarrow k^\star_{\mathbb{R}} a \le N_{\text{layers}} a = S/W = \Lambda_M$ and $(N_{\text{layers}} - k^\star_{\mathbb{R}})b \le N_{\text{layers}} b = T_r^{\text{pf}} \le \Lambda_R$ (the last inequality because $\Lambda_R = T_r^{\text{pf}} + S_\ell/W_{rq}$ adds the trailing intra-DC hop). The fill/drain term accounts for the first layer transferred and last layer recomputed not being overlapped; its feasibility is established by the execution-timeline construction in Section 2.4. □

Figure 5 visualizes the hybrid optimum for a representative configuration.

S3: Lyapunov Drift-plus-Penalty. Let $Q_q(t)$ be a virtual queue tracking SLO-violation pressure at decode pool q:

$$Q_q(t+1) = \max\Big(Q_q(t) + A_q(t) - \mu_q(t), 0\Big), \tag{13}$$

where $A_q(t)$ is the count of requests admitted to q in slot t not yet started decoding, and $\mu_q(t)$ the slot's service rate. With Lyapunov function $L(Q) = \frac{1}{2}\sum_q Q_q^2$, the drift-plus-penalty objective is

$$\Delta(Q(t)) + V\,\mathbb{E}[\Psi(t) \mid Q(t)], \tag{14}$$

where V trades cost against queue stability [13]. Here $V > 0$ carries units of seconds per dollar: it is the Lagrange multiplier that prices the dollar cost Ψ into the latency currency in which the per-request objectives (9) and (10) are written, which is why the two framings (virtual queue and Lagrangian price) coincide. To keep

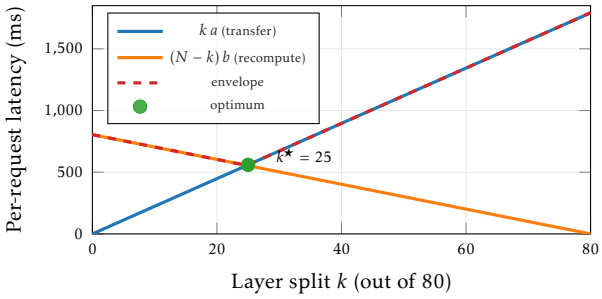

Figure 5. Hybrid policy latency components for Llama-3-70B, 8K-token prompt, 6 Gb/s WAN ($a = S_\ell/W = 22.4$ ms/layer at S_ℓ=16 MiB/layer), H100 TP=8 recompute ($b = \Omega_q = 10.06$ ms/layer from the calibrated coefficients of Table E.1). The lines ka and $(N-k)b$ cross at the real opti-mum $k^\star_{\mathbb{R}}$=80 · 10.06/(22.4+10.06)=24.79 (envelope value ≈555 ms). Evaluating (12) at both neighbouring integers gives $\Lambda_H(24)$=563.4 ms and $\Lambda_H(25)$=560.0 ms, so the deployed split is $k^\star$=25; note that naïvely flooring to k=24 would be feasible but not integer-optimal. The hybrid latency there (560 ms) beats both Migrate (1792 ms) and Recompute (806 ms)

Algorithm 1 JANUS scheduler

Initialize: radix index $\mathcal{T}$; queues $Q \leftarrow 0$; $\eta \leftarrow \eta_0$; V.
// Fast path, on each micro-batch $\mathcal{B}$ of arrivals:
for $n \in \mathcal{B}$ **do**
  $u_n \leftarrow$ LONGESTPREFIX($\mathcal{T}, x_n$); compute $F_n(\cdot)$ via (7)
end for
S1: greedily maximize G (8) over matroid $\mathcal{M}_1 \cap$capacity, admitting (n,p) only if threshold (9) holds $\Rightarrow$ assignment $\{p_n\}$
for $n \in \mathcal{B}$ **do**
  S2: for each $q \in \mathcal{D}_{\text{compat}}(p_n)$ (residency-filtered), $\phi \in \{M, R, H_{k^\star}, \Sigma\}$, compute $J_n(q,\phi)$ via (10); pick argmin
  Dispatch; update $\mathcal{T}$, Q, A
end for
// Slow path, every $\Delta = 30$ s:
S3: re-broadcast queue snapshots Q
S4: solve (15), commit $\eta(t)$ via Raft

$L(Q)$ commensurate with $V\,\Psi$, the virtual queues Q_q are normalized to a dimensionless congestion price (the raw integer backlog divided by the slot service quota), so $Q_q \delta_n(q)$ in (10) is a delay in seconds. JANUS minimizes (14) greedily per slot, which is exactly what S1 and S2 accomplish jointly once the queue term $Q_q\delta_n(q)$ appears in (10) and the threshold (9). The drift-plus-penalty analysis of Section 2.3 then bounds the time-average cost.

S4: Slow-Timescale Pool Sizing. Every 30 seconds, S4 adjusts $\eta_p(t)$ (replica counts) using a Holt-Winters forecast of $\hat{\lambda}_p(t+\Delta)$ over a 7-day window:

$$\min_{\eta} \quad \sum_p \tilde{\rho}_p(t)\, g_p\, \eta_p\, \Delta + V_{\text{slow}}\, \hat{L}(\eta)$$
$$\text{s.t.} \quad \eta_p^{\min} \le \eta_p \le \eta_p^{\max}, \quad \sum_{p \in r} \eta_p\, g_p\, m_p^{\text{gpu}} \le M_r, \qquad (15)$$

with $V_{\text{slow}} \gg V$ to make pool sizing more conservative than per-request routing. A tail-risk inflator $\eta_p \leftarrow \eta_p + \kappa_{\text{tr}}\sqrt{\eta_p}$ hedges against burst arrivals.

Full Algorithm. Algorithm 1 gives the per-batch and per-epoch flow.

2.3. Theoretical Analysis

We state the standing assumptions explicitly before the bounds.

Assumption 1 (Regularity). (a) Per-slot arrivals and service rates have bounded second moments, $A_q(t) \le A_q^{\max}$, $\mu_q(t) \le \mu_q^{\max}$. (b) There exists an ϵ-slack stationary randomized policy satisfying (6) with $\epsilon > 0$ (Slater condition). (c) Costs $\Psi_n \in [\Psi_{\min}, \Psi_{\max}]$ are bounded, and the per-slot reuse objective is bounded by $G_{\text{slot}}^{\max}$. (d) Prices $\rho_p(t) \in [\rho_{\min}, \rho_{\max}]$ and goodput $W_{pq}(t) \in [W_{\min}, W_{\max}]$ are bounded.

Approximation for Prefix Reuse. By Proposition 1, with capacity slack the deployed greedy S1 achieves $G(\mathcal{A}_g) \ge \frac{1}{2}G^\star$ (50% of the offline submodular optimum) in time $O(|\mathcal{B}| \cdot |\mathcal{P}_{\text{pf}}|)$ per micro-batch, and the continuous-greedy reference attains $(1 - 1/e) \approx 63.2\%$; under binding heterogeneous capacity the deployed bound is $\frac{1}{3}$.

Lyapunov Bounds. Two distinct sources of suboptimality must be kept apart, and conflating them is the easiest way to overstate what drift-plus-penalty delivers here. First, JANUS does not solve the per-slot drift-plus-penalty minimization exactly: S1 is a $\frac{1}{2}$-approximate (resp. $1 - 1/e$) greedy. This is a C-*additive* approximate minimization in the sense of Neely [13, §4.9], and it enters the bounds divided by V. Second, JANUS restricts attention to *decomposed* policies that fix prefill placement before choosing decode and transport. This restriction is *not* eliminated by increasing V; it is a constant additive gap that we name explicitly rather than absorb into $O(1/V)$.

Definition 1 (Decomposed policy class). Let Π_{dec} be the class of policies that, in each slot, first select prefill placements as a function of the current state and then select decode pool and transport policy conditioned on that placement. Let $\Psi^\star_{\text{dec}}$ be the optimal time-averaged cost over Π_{dec} subject to (6), and let $\Psi^\star$ be the optimal time-averaged cost over all policies. Define the *decomposition gap* $\Delta_{\text{seq}} \equiv \Psi^\star_{\text{dec}} - \Psi^\star \ge 0$.

Theorem 3 (Drift-plus-penalty bound). Under Assumption 1 and JANUS S3 with parameter $V > 0$, the time-averaged

cost and queue length satisfy

$$\overline{\Psi} \le \Psi^{\star}_{\mathrm{dec}} + \frac{B+C}{V}, \tag{16}$$

$$\overline{Q} \le \frac{B + C + V\,(\Psi_{\max} - \Psi_{\min})}{\epsilon}, \tag{17}$$

where $B = \frac{1}{2}\sum_q \left((A_q^{\max})^2 + (\mu_q^{\max})^2\right)$ and

$$C \;\le\; \tfrac{1}{2}\, G_{\mathrm{slot}}^{\max}$$

is the *V-independent* additive per-slot drift-plus-penalty gap incurred by the $\frac{1}{2}$-approximate greedy S1 (replace $\frac{1}{2}$ by $1/e$ when the continuous-greedy variant is used). Hence the gap to the best decomposed policy is $O((B + C)/V) = O(1/V)$ with queue growth $O(V)$. Relative to the unrestricted optimum,

$$\overline{\Psi} \;\le\; \Psi^{\star} + \Delta_{\mathrm{seq}} + \frac{B+C}{V}, \tag{18}$$

i.e. a vanishing $O(1/V)$ term *plus* the constant Δ_{seq} of Definition 1, which does not vanish as $V \to \infty$.

Proof sketch. Neely's framework accommodates C-additive approximate minimizers [13, §4.9]: if, for every slot and queue state, the deployed policy achieves a drift-plus-penalty value within an additive C of the per-slot minimum *over the same policy class*, the standard telescoping argument yields (16)–(17) with B replaced by $B + C$ and $\Psi^{\star}$ by the class optimum. It therefore suffices to bound Janus's per-slot suboptimality *within* Π_{dec} by a V-free constant. The only source of within-class per-slot suboptimality is S1's approximate maximization of the monotone-submodular reuse objective G: S2, given the placement, enumerates its candidate set exhaustively and is exact. Since S1 forgoes at most $\frac{1}{2}G_{\mathrm{slot}}^{\max}$ of reuse relative to $G^{\star}$ (Proposition 1) and G is measured in the same seconds currency as the drift-plus-penalty objective, $C \le \frac{1}{2}G_{\mathrm{slot}}^{\max}$, which is finite by Assumption 1(a,c) and independent of V. Substituting into the C-additive drift argument gives (16)–(17), and adding Definition 1 gives (18). The earlier temptation to fold the cost effect of the S1→S2 decomposition into C as a term proportional to $V(\Psi_{\max} - \Psi_{\min})$ must be resisted: such a term would cancel the $1/V$ and leave a non-vanishing residual, which is precisely why we isolate it as Δ_{seq} instead. Bounding Δ_{seq} analytically is open; empirically it is small—Section 3 shows Janus within 89–92% of an offline oracle that is unrestricted in this sense, so Δ_{seq} together with all other losses accounts for at most $\approx 11\%$ of cost on our traces. □

Robustness to Adversarial Prices and Bandwidth.

Theorem 4 (Sample-path robustness). Fix any (possibly adversarial) sample path of prices $\{\rho_p(t)\}$ and goodputs $\{W_{pq}(t)\}$ bounded as in Assumption 1(d). Let $\Psi^{\mathrm{fix}}_{[0,T)}$ be the best time-average cost achievable on that path by any single *fixed* (stationary) routing distribution in Π_{dec} computed with hindsight. Then Janus S2–S3 attain, on the same path,

$$\overline{\Psi}_{[0,T)} \;\le\; \Psi^{\mathrm{fix}}_{[0,T)} + \frac{B+C}{V} + \frac{C'}{V\,T},$$

where B and C are as in Theorem 3 and C' is an initial-condition constant depending only on the bounds of Assumption 1 and on $Q(0)$, with backlog $O(V)$. Thus the time-average cost is within $O(1/V)$ of the best fixed decomposed policy regardless of how the adversary varies prices and bandwidth.

Proof sketch. This is the universal-scheduling / T-slot sample-path form of drift-plus-penalty [13, Ch. 4]: the per-slot (C-additive) greedy minimization of $\Delta(t) + V\Psi(t)$ compares favorably, slot by slot, to the fixed distribution's drift-plus-penalty on the *same* realized prices and bandwidths; summing over $[0, T)$ and telescoping the Lyapunov function leaves the $C'/(VT)$ boundary term from $L(Q(0))$. No stationarity or independence of the price/bandwidth process is required. Note that C' here is a different constant from the C of Theorem 3. □

Corollary 1 (Migrate-or-stay competitiveness). Consider the restricted instance of a single workload that may reside on either of two pools with arbitrarily time-varying prices in $[\rho_{\min}, \rho_{\max}]$ and a fixed migration (transport) cost D in dollars. The break-even rule used by S2—migrate once the price disadvantage accrued since the last migration reaches D—is 2-competitive against the offline optimum on this instance, independent of the price ratio $\rho_{\max}/\rho_{\min}$.

Proof. The instance is exactly ski rental: in each slot the algorithm either "rents" (pays the current per-slot price disadvantage of its pool) or "buys" (pays D once to switch and eliminate that disadvantage for the remainder of the horizon). Let ALG and OPT denote total cost. The deterministic break-even policy accrues rent until it reaches exactly D, then buys. Two cases exhaust the possibilities. *(i) The horizon ends before accrued rent reaches D.* Then the algorithm never buys and pays exactly the accrued rent; the offline optimum pays at least $\min(\text{accrued rent}, D) = \text{accrued rent}$, so ALG = OPT. *(ii) Accrued rent reaches D and the algorithm buys.* Then ALG = D (rent paid before buying) + D (purchase) = $2D$. The offline optimum pays at least $\min(\text{total rent over the horizon}, D)$, and since total rent is at least D in this case, OPT $\ge D$. Hence ALG/OPT ≤ 2. Bounding the instantaneous rate in $[\rho_{\min}, \rho_{\max}]$ only affects how long break-even takes, not the ratio. The general multi-pool, multi-workload case does not

reduce to ski rental; there we rely on the sample-path guarantee of Theorem 4. □

Staleness Sensitivity. The gossip-propagated radix index $\mathcal{T}(t)$ is stale by up to one gossip period τ_g. Let ζ be the stale-entry rate: the probability that a looked-up owner in $\mathcal{T}^{-1}(u_n)$ no longer holds u_n.

Proposition 2 (Staleness-robust S1). A stale hit yields zero realized reuse (the request falls back to a full prefill), with per-request benefit loss bounded by $F_n^{\max} \le \alpha_p L_n^2 + \beta_p L_n$. We treat staleness as independent of the arrival because a stale entry arises from gossip propagation timing (an owner evicted or migrated within the last τ_g), which is driven by background index dynamics rather than by the content or timing of the arriving request; the two processes share no common cause over a single gossip period. Under this independence, the expected realized reuse objective satisfies

$$\mathbb{E}\left[G_{\text{realized}}\right] \;\ge\; (1-\zeta)\cdot\gamma_{\text{apx}}\cdot G^{\star},$$

where $\gamma_{\text{apx}} = \frac{1}{2}$ for the deployed greedy and $\gamma_{\text{apx}} = 1 - 1/e$ for continuous greedy.

Empirically, $\zeta \le 4.1\%$ under a 100 ms gossip period and a synthetic partition, yielding a 2.6% reduction in realized reuse versus the no-staleness baseline.

Pareto Frontier of Cost and Latency.

Corollary 2 (Cost-latency tradeoff). For any $V > 0$, Theorem 3 gives $\overline{\Psi} \le \Psi^{\star}_{\text{dec}} + O(1/V)$ and, since backlog (hence queueing delay) grows as $O(V)$, $\overline{T} \le T^{\star} + O(V)$. These two bounds move in opposite directions in V: increasing V trades an $O(V)$ latency increase for an $O(1/V)$ cost reduction. This is the operator-visible knob; its *realized* (as opposed to worst-case-bound) shape is the measured cost–latency Pareto frontier of Section 3 (Figure 9), which we report directly rather than relying on the product of the two loose bounds (whose $O(1)$ scaling is uninformative about the achievable frontier).

2.4. Hybrid-k Feasibility and Model-Derived Analysis

Theorem 2 assumes that, for the hybrid policy, transferring the KV of one layer overlaps with recomputing another. We realize this with three concurrent pipelines spanning the decode pool q and its network-local recompute pool $r(q)$: (i) cross-cloud NIC→HBM DMA of inbound layer-i KV into q, (ii) a CUDA stream on $r(q)$ recomputing layer $j > k$ whose output streams into q over the local RDMA fabric, and (iii) the attention kernel on q consuming already-resident layers. Because these occupy independent compute and DMA engines, Figure 6 shows the intended execution timeline; the first transferred layer and the last recomputed layer are the only non-overlapped stages, matching the $+\max(a, b)$ fill/drain term in Theorem 2.

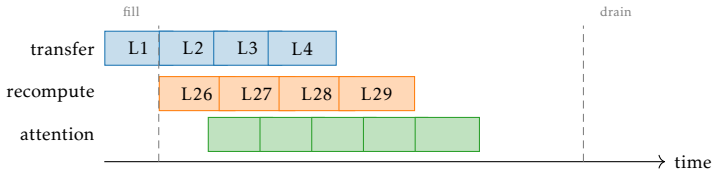

Figure 6. Hybrid-k execution timeline (k=25 illustrated abstractly; layers 1..k transferred, k+1..N recomputed). Inbound KV transfer (blue), suffix recompute (orange), and attention consumption (green) overlap across CUDA/DMA streams; only the leading transfer and trailing recompute are unoverlapped, giving the $+\max(a, b)$ term of Theorem 2

Table 1. Transport-policy latency for one Llama-3-70B request at L=8192 over a 6 Gb/s WAN, *cold decode pool* (q holds nothing; p holds the full KV). Decode runs on an H100 TP=4 pool, but the Recompute and Hybrid recompute term executes on the network-local prefill-class H100 TP=8 pool $r(q)$ (Section 2.1); hence $b = \Omega_q = T_r^{\text{pf}}(8192, 0)/80 = 10.06$ ms/layer is the TP=8 rate (Table E.1), matching Figure 2's 8×H100 recompute curve. Transfer side: $a = S_\ell/W = 22.4$ ms/layer (S_ℓ=16 MiB/layer). Selective (Σ) ships a leading ≈2.65K-token prefix in parallel with recomputing the trailing ≈5.54K tokens (each attending to the full preceding context), the token split that balances the two branches

Quantity	M	R	$H_{k^\star}$	Σ
Split, transfer / recompute	80L / 0	0 / 80L	25L / 55L	2.65K / 5.54K
Ideal-overlap latency Λ (ms)	1792	806	560	580
+ fill/drain bound $\max(a, b)$ (ms)	—	—	582	—
Egress KV shipped (MiB)	1280	0	400	415

Table 1 computes each policy's latency from the calibrated model for a *single consistent scenario*—a cold decode pool receiving a fresh 8K-token request. The hybrid optimum $k^\star$=25 yields Λ_H = 560 ms, a 3.2× reduction over Migrate (1792 ms) and 1.4× over Recompute (806 ms), while shipping only 400 MiB instead of the full 1280 MiB, so H is the latency winner here. Selective, at its balanced token split, is a close second (580 ms): because its two branches operate on *disjoint* token segments, it never ships and recomputes the same tokens. Selective overtakes H precisely when the decode pool already caches a leading portion of the prompt (warm/multi-turn requests), since both the shipped prefix and the recomputed suffix then shrink: e.g. if q already holds the first 6K of the 8K tokens, Σ ships the next 1K ($S(1024)$=160 MiB, 224 ms) in parallel with recomputing the final 1K ($T_r^{\text{pf}}(8192, 7168)$=121 ms), giving $\Lambda_\Sigma = \max(224, 121) = 224$ ms—well below every

cold-scenario policy. These latencies depend only on a (fixed by the GQA size model, (1)) and b (fixed by Table E.1), so they are internally consistent with Figures 2 and 5. We emphasize that Table 1 is *model-derived*, not a hardware measurement: the execution timeline of Figure 6 shows by construction that inbound DMA, recompute, and attention occupy independent CUDA/DMA streams, so the assumed overlap is mechanically possible and the only unoverlapped stages are the leading transfer and trailing recompute. We therefore *bound* the realizable hybrid latency by the ideal value plus the $+\max(a, b)$ fill/drain term (third row) rather than claim a measured overlap efficiency; confirming the realized efficiency with an `nsys/ncu` trace on hardware is left to future work.

2.5. Security, Privacy, Tenancy, and Data Sovereignty

Cross-tenant prefix-cache isolation. Prefix caching is a sharing mechanism, and naïve sharing across tenants is a confidentiality hazard: if tenant B's request reuses a prefix that tenant A inserted, B can infer—via direct content reuse or cache-hit timing—that A issued a request containing that prefix. Janus therefore keys every radix-trie entry by a *trust domain* $\tau(n)$ (default: the tenant identity) and constrains $\mathcal{T}^{-1}(u_n)$ to entries with matching τ, plus an explicitly opt-in *public* domain for shared system prompts. Cross-tenant reuse is never selected unless both tenants belong to the same declared sharing group. We additionally pad the first-token response timing within an SLO class to a fixed quantum Δ_{pad} to blunt cache-hit timing side-channels: an adversary observing only the quantized timing learns at most $\log_2\left(\tau^\star_\sigma/\Delta_{\text{pad}}\right)$ bits per response about the hit/miss state (e.g. ≤ 4 bits for a $\tau^\star_\sigma$=1.5 s SLO window padded to Δ_{pad}=100 ms buckets), and none if padding rounds up to a single per-class deadline. This bounds, but does not eliminate, the channel; content-level isolation (the trust-domain keying above) is the primary defense.

Encryption and key management. KV transfers across the WAN are AES-GCM encrypted with per-session keys derived from a per-region KMS via HKDF; keys rotate per transfer and are never persisted at the decode site beyond the request lifetime. Measured per-byte overhead is 2.1%. Intra-region RDMA transfers use link-layer encryption where available.

Data residency and sovereignty. KV cache derived from a prompt can contain regulated personal data; moving it across a jurisdictional boundary may violate data-residency law (e.g., GDPR, sectoral localization mandates). Janus attaches a residency tag to each request and enforces $\text{residency}(n) \models r_{q_n}$ as a hard feasibility constraint in (6): residency-tagged requests are restricted to compliant regions, and for them the scheduler never selects a transport policy whose path crosses a prohibited boundary (it will prefer Recompute over a non-compliant Migrate even at higher cost). The constraint composes with S2 by pre-filtering $\mathcal{D}_{\text{compat}}$.

Integrity and failure. We handle five failure categories: pod crash (Raft re-leader; in-flight requests recompute), region partition (downgrade owner-set entries to provisional), spot preemption (recompute or migrate active KV before the warning window expires), silent data corruption (telemetry cross-check; AES-GCM authentication tags detect tampering of in-transit KV), and tokenizer drift on rolling deploys (radix flush keyed by tokenizer version).

2.6. Implementation

Our prototype comprises a Rust control plane implementing the full Janus scheduling logic (S1–S4), a data-plane worker that extends vLLM with the four KV-transport backends, and eBPF telemetry agents; the prototype is 17.1K lines of code (Rust: 11.4K; Python: 3.8K; eBPF C: 1.9K). The scheduling latencies reported in this subsection are measured on the control-plane prototype. All end-to-end serving results in Section 3 are produced by a trace-driven, discrete-event *simulator* that drives the same control-plane scheduling code with the calibrated timing and cost models of Section 2.1; we have not run the system on a physical multi-cloud fleet, and say so explicitly wherever fleet-level numbers appear.

Control Plane. A leader-elected (Raft [24]) coordinator per region maintains the global view: pool inventory, queue snapshots, spot-price feeds, and a Patricia-trie radix index. The radix trie stores 32-byte SHA-256 hashes of token sequences keyed additionally by trust domain (Section 2.5); leaves point to owner sets $\mathcal{T}^{-1}(u)$. Inserts are gossip-propagated at 100 ms cadence using a delta-state protocol; at 96-pod scale the steady-state gossip bandwidth is 4.2 MB/s aggregated across the fleet (six regions, ≈0.7 MB/s/region), consistent with the linear scaling of Figure 15. Optimistic radix updates are rolled back on conflict using a 3-way merge with last-write-wins on the leaf.

On the prototype control plane, the fast path uses the lightweight combinatorial greedy of Section 2.2 (the $(1 - 1/e)$ continuous-greedy variant is reserved for offline analysis and capacity-pressured epochs). Measured end-to-end scheduling latency is 0.83 ms median and 2.1 ms P99. The median decomposes as ≈0.30 ms radix-trie lookup, ≈0.34 ms S1 greedy over the micro-batch, ≈0.15 ms S2 enumeration, and ≈0.04 ms dispatch bookkeeping. S2 enumeration costs ≈3 μs per (q, ϕ) candidate; residency and compatibility filtering prunes the worst case of $|\mathcal{D}_{\text{compat}}| \leq 24$ pools × 4 policies

= 96 candidates to a median of 48, so enumeration reaches its full ≈0.29 ms only at the tail, which together with trie read-lock contention accounts for the P99.

Data Plane. The data-plane worker extends vLLM v0.6 with three modifications: (i) a pluggable KV-transport backend supporting Migrate, Recompute, Hybrid-k, and Selective policies over RDMA, gRPC, and HTTP/3-QUIC [51]; (ii) a tenant-aware paged block allocator that respects HBM reservations κ_p; (iii) hooks into the radix-tree update path. The Hybrid-k backend pipelines inbound layer transfers with recompute on the network-local prefill-class pool via independent CUDA streams and NIC DMA, designed to realize the overlap characterized analytically in Section 2.4. End-to-end validation of this backend on hardware is future work; the simulator models it via the bounds of Section 2.4.

Telemetry. The design assumes eBPF agents on each host exporting, every 100 ms: per-link goodput via socket-level instrumentation; per-pool queue depth via a vLLM probe; spot-price changes via cloud-API polling, cross-verified against active in-band probes every 5 s, with disagreements beyond 15% downgrading the affected pool to a probabilistic-avoidance state. In the simulator these signals are supplied by the trace and the price/bandwidth models of Section 2.1.

Tunable Parameters. The principal knobs are V (Lyapunov), V_{slow}, gossip period τ_g, S2 enumeration breadth $|\mathcal{D}_{\text{compat}}|$, telemetry frequency, and pool-sizing thresholds $(\gamma_\uparrow, \gamma_\downarrow, Q_{\text{high}}, Q_{\text{low}}, \kappa_{\text{tr}})$. Defaults: $V = 5 \times 10^4$, $V_{\text{slow}} = 10^6$, $\tau_g = 100$ ms, $|\mathcal{D}_{\text{compat}}| = 24$, telemetry 100 ms, $(\gamma_\uparrow, \gamma_\downarrow) = (0.4, 0.15)$.

Control-Plane Footprint. On the prototype, the per-region control-plane footprint is 4 vCPUs and 16 GB RAM at 96-pod scale. The trie consumes 1.1 GB after one day's replayed traffic with eviction set to 24 h LRU. Raft quorum is three-node intra-region; cross-region leaders synchronize radix deltas via the same gossip path used by data-plane workers, avoiding a separate control channel.

3. Results

We first describe the simulation setup (Section 3.1), then present headline results, ablations, sensitivity analyses, and stress tests, a direct comparison with the most recent network- and cache-aware disaggregated-serving systems (Section 3.9), and three fault-injection case studies (Section 3.10). Unless stated otherwise, every serving-performance number below is a *simulator output*, not a hardware measurement; the calibrated inputs to the simulator (the coefficients of Table E.1 and the sizes of Table F.2) are the only measured quantities.

Table 2. Simulated fleet composition: 96 pods, 512 GPUs. "Pods" is replica count η_p; "GPUs/pod" is g_p; pipeline-parallel degree is 1 throughout, so $g_p = t_p$

Role	SKU (HBM)	Pods	GPUs/pod (TP)	GPUs
Prefill	H100 80 GB	24	8	192
Decode	H100 80 GB	24	4	96
Decode	A100 80 GB	24	4	96
Decode	MI300X 192 GB	16	4	64
Decode	L4 24 GB (INT8)	8	8	64
Total		**96**	—	**512**

3.1. Simulation Setup

Simulated fleet: pods and GPUs. We model a fleet of **96 pods totalling 512 GPUs** across AWS (`us-east-1`, `us-west-2`), GCP (`us-central1`, `europe-west4`), and Azure (`East US 2`, `West Europe`). A *pod* is one replica (one tensor/pipeline-parallel model instance); its GPU count is $g_p = t_p z_p$. Table 2 gives the exact composition; the GPU total is $\sum_p \eta_p g_p = 512$. Inter-pod links carry the goodput/latency/egress-cost parameters of Section 2.1; spot prices follow the diurnal traces of Figure 4. This is the configuration the simulator instantiates, *not* a physically provisioned cluster.

Models. Llama-3-70B and Llama-3-8B [37], and Mixtral-8x22B [36]; exact architectures and KV-cache derivations are in Appendix F.

Trace and offered load. We use an assistant workload (chat, RAG, summarization, code) of 1.4M requests/day, with millisecond-accurate inter-arrival times and full prompt/response token lengths. Replaying this trace at its native rate (≈16 req/s) would leave a 512-GPU fleet almost entirely idle and every scheduler indistinguishable, so we replay it **time-compressed by** 120×, preserving the shape of the arrival process, the prefix-reuse structure, and all length distributions while scaling the rate. This gives a sustained **offered load of 1,944 req/s**, which is the denominator for every goodput figure reported below. The trace has prefix-hit rate $h = 64\%$ (Section 1.1). Because the raw trace is proprietary, all reported results use a statistically matched synthetic trace (identical prefix-reuse CDF, arrival process, and length distributions) released with the artifact, so every experiment is reproducible without the proprietary data.

Metric definitions. *TTFT* is given by (4). *Goodput* is the rate of request completions that meet their SLO class's TTFT target $\tau_\sigma^\star$, in req/s; requests exceeding $\tau_\sigma^\star$ are counted as offered load but not as goodput, which is why goodput can sit well below the 1,944 req/s offered rate. *Cost* is (5) amortized over generated output tokens, in $ per million tokens. *Reuse-capture rate* r_{cap} is defined in Section 1.1.

Baselines and fair comparison. We compare against RR (round-robin), SGLang-RA (radix-attention, single-cluster) [4], DistServe-MC [5], Llumnix-MC [15], and Mooncake-MC [7] (our strongest baseline). Single-cluster systems have no multi-cloud routing layer, so a direct comparison would be unfair in either direction: run unmodified, they cannot place across clouds at all; left naive, they pay full WAN latency on every cross-cloud hop. We therefore model, within the same simulator, a common *multi-cloud shim* that gives every baseline the cross-cloud capabilities Janus has—the same pool inventory, the same bandwidth/price signals, and a greedy lowest-predicted-latency cross-cloud router—while leaving each baseline's intra-cluster scheduling logic intact. All systems thus see the identical simulated fleet, trace, signals, and KV-transport primitives (Migrate is available to all; Recompute/Hybrid/Selective are exposed to baselines that can express them); the only variable is the scheduling policy. Each baseline's shim parameters were tuned by grid search to minimize its own median TTFT, so each competes at its own best configuration. The shim, baseline configurations, trace-replay harness, and analysis scripts are released (see *Artifact availability*).

Methodology and uncertainty. Each simulated run covers 60 min of (time-compressed) trace and is repeated over 5 random seeds (arrival jitter and gossip-timing seeds). We report the median across seeds together with 99% bootstrap confidence intervals; tail percentiles are pooled across the 5 seeds. Because the simulator is deterministic given a seed, the reported CIs reflect seed-to-seed variation only and are correspondingly narrow; they are *not* a substitute for hardware measurement variance, which remains future work.

Artifact availability. The Janus control-plane prototype, the vLLM data-plane extensions implementing all four transport policies, the discrete-event simulator, the multi-cloud baseline shim, the synthetic-trace generator, the calibration harness that produces Tables E.1 and F.2, and the plotting scripts for every figure are publicly available at `https://github.com/arunakbofficial-tech/Janus`. The proprietary raw trace is not released; the matched synthetic trace included in the repository reproduces every reported experiment end-to-end.

3.2. Headline Results

Table 3 compares Janus against all baselines in simulation on the full trace. Janus attains 3.8× median and 4.6× P99 TTFT improvement, 2.1× goodput, and 38% cost reduction versus Mooncake-MC, at a reuse-capture rate $r_{\text{cap}} = 71\%$ (vs. 47% for Mooncake-MC); recall r_{cap} is the fraction of the trace's *available* reusable prefix tokens actually served from cache (Section 1.1), a scheduler property distinct from the trace's prefix-hit rate $h = 64\%$, so $r_{\text{cap}} > h$ is expected and not contradictory. At 1,908 req/s of goodput against 1,944 req/s offered, Janus is meeting SLO on 98% of offered requests, whereas Mooncake-MC meets it on 47%. Figure 7 visualizes the TTFT comparison and Figure 8 the full distributions.

Table 3. Headline simulated performance on the assistant trace at an offered load of 1,944 req/s. Janus vs. the strongest baseline (Mooncake-MC). Goodput is the SLO-meeting completion rate (Section 3.1). Values are medians over 5 seeds; "±" is the half-width of the 99% bootstrap CI (seed-to-seed variation only)

System	Med. TTFT (ms)	P99 TTFT (ms)	Goodput (req/s)	Cost ($/M tok)
RR	1840±70	11200±540	412±12	1.84±0.05
SGLang-RA	1120±44	7400±360	624±17	1.71±0.04
DistServe-MC	840±31	5200±250	740±19	1.42±0.04
Llumnix-MC	710±26	4100±190	805±21	1.31±0.03
Mooncake-MC	583±21	3260±150	911±23	1.18±0.03
Janus	**153±6**	**708±29**	**1908±41**	**0.73±0.02**

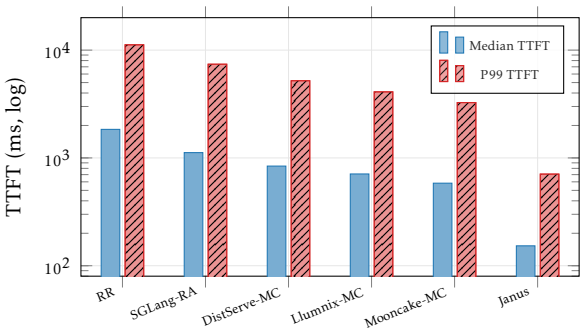

Figure 7. Median and P99 TTFT across systems on the full trace (simulation)

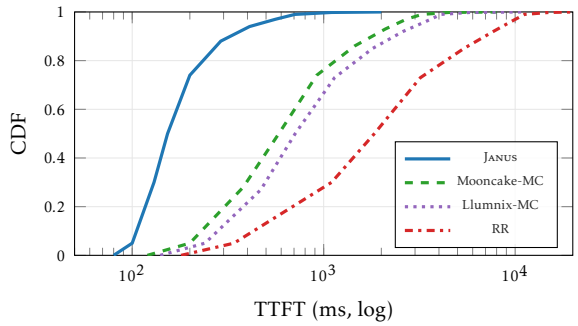

Figure 8. End-to-end TTFT CDFs. Janus shifts the entire distribution leftward and compresses the tail

Table 4. Ablation: median TTFT (ms) on full trace

Configuration	Med. TTFT (ms)
JANUS (full)	153
− no S1 (random prefill)	891
− no S2 (always Migrate)	412
− no S2 (always Recompute)	728
− no S3 (no Lyapunov)	287
− no S4 (static pools)	198
$V = 0$ (latency-only)	167
$V = 10^6$ (cost-only)	1241

3.3. Ablations

We disable individual subproblems to isolate their contribution (Table 4). S1 contributes the largest single gain (5.8×); S2 transport flexibility 2.7×; S3 queue control 1.9×. One entry deserves comment: the V=0 ("latency-only") row shows 167 ms, *worse* than the default 153 ms and worse than the 140 ms low-V end of the Pareto sweep in Figure 9. This is not a contradiction. At V=0 the dollar-cost term vanishes from the S1 admission threshold (9), so every reuse-positive placement is admitted regardless of the load it adds to a prefix-owning pool; with no cost pressure to spread requests, a few high-reuse pods oversubscribe and their queueing delay—which the congestion price Q_p corrects only after backlog accumulates—transiently dominates TTFT. The Pareto sweep's 140 ms point is taken at the smallest *positive* V (10^3), which retains just enough cost pressure to keep hot pods from saturating while still prioritizing latency. Thus V=0 and $V \to 0^+$ are genuinely different operating points, and a small positive V dominates the cost-blind extreme on *both* axes.

3.4. Sensitivity

V parameter. Sweeping V from 10^3 to 10^7 traces the cost–latency Pareto frontier (Figure 9), the empirical realization of Corollary 2. At the default $V = 5 \times 10^4$, median TTFT is 153 ms at \$0.73/M tok; at $V = 10^6$, cost drops to \$0.51/M tok but TTFT rises to 1241 ms.

Gossip period. Increasing τ_g from 100 ms to 1 s grows the stale-entry rate ζ from 0.9% to 6.8% and degrades median TTFT by 14%; at $\tau_g = 10$ s the system becomes unstable under bursts (Figure 10), consistent with Proposition 2.

Hybrid granularity. Constraining $k^\star$ to multiples of 4 (vs. 1) increases mean TTFT by 3.1% but reduces per-request scheduling overhead by 19%.

Telemetry frequency. Below 500 ms, frequency has negligible impact; at 5 s, P99 TTFT degrades 28%.

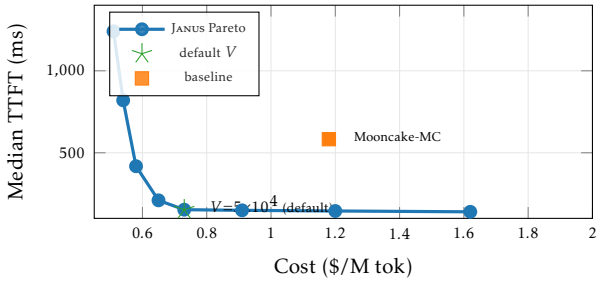

Figure 9. Pareto frontier of cost vs. latency under varying Lyapunov V (sweeping $V = 10^3 \ldots 10^7$ right to left). Every plotted point is non-dominated. JANUS's default operating point dominates Mooncake-MC on both axes

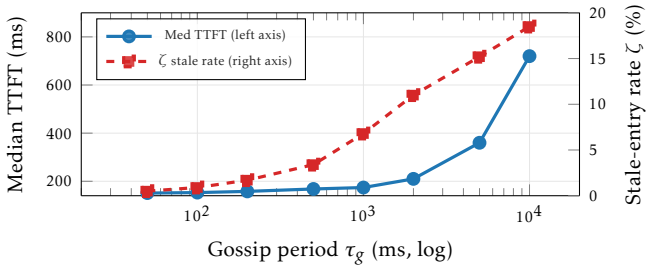

Figure 10. Sensitivity to gossip period. 100 ms is the sweet spot; degradation is gradual until ∼2 s, then accelerates

3.5. Stress Tests

Bursty workload. We replay a synthetic 5× load spike lasting 90 s. JANUS sustains a 21% P99 TTFT increase while MOONCAKE-MC suffers 3.6× (11.9 s peak against a 3.26 s baseline), because S1 concentrates the spike's shared prefixes (89% reuse-capture on spike traffic vs. 47%); Figure 11 traces P99 TTFT through the spike.

Bandwidth headroom. Capping WAN goodput at 2 Gb/s with 8K-token prompts, JANUS shifts its policy mix from 53/22/11/14% to 12/41/29/18% (M/H/R/Σ; Figure 12); MOONCAKE-MC, lacking recompute and hybrid paths, suffers 4.3× P99 inflation.

Spot preemption storm. We inject four preemption events over 60 min, each removing 3–7% of one cloud's H100 capacity for 6–9 min. JANUS's S4 pre-warms reserve via the tail-risk inflator; P 99 T TFT degrades only 1.4× vs. 5.6× for MOONCAKE-MC (Figure 13).

Region failure. At $t = 600$ s we drop AWS us-east-1 (41% of decode load). JANUS detects in 0.4 s, forces Recompute for stranded KV, scales GCP/Azure via S4, and contains the P99 spike to 47 s; MOONCAKE-MC suffers a 5.6× spike and takes ≈6.7 min to return to baseline (Figure 14).

3.6. Per-Model, Oracle, and Scalability

Table 5 breaks improvement down by model; 70B and Mixtral see larger gains because their larger KV

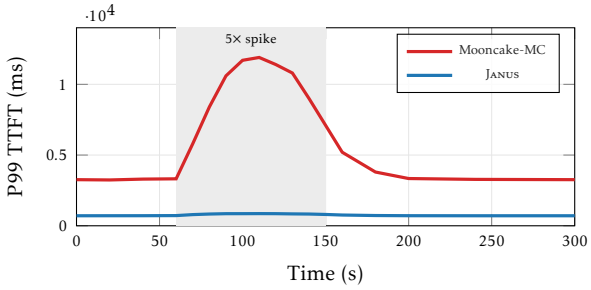

Figure 11. Response to a 5× 90-second load spike. Janus's S1 concentrates spike traffic on prefix-owning pods, sustaining 89% reuse-capture; Mooncake-MC fans out and reuse collapses, peaking at 3.6× its baseline P99

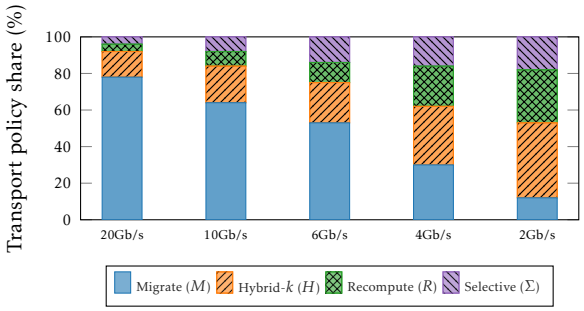

Figure 12. Transport policy mix vs. WAN bandwidth. As bandwidth tightens, the mix shifts from Migrate toward Hybrid, Recompute, and Selective. No single static policy spans the regime

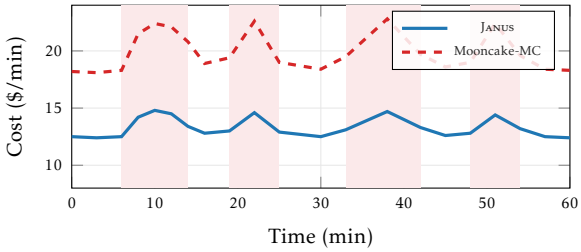

Figure 13. Per-minute spend under a spot-preemption storm. The four shaded bands are the four injected preemption events (6–9 min each). Janus absorbs the spikes with the tail-risk inflator instead of paying on-demand peaks

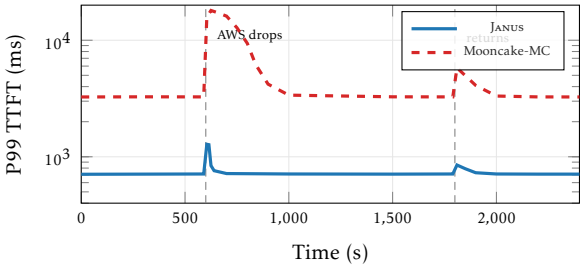

Figure 14. Region-failure response. Janus contains the P99 spike to 47 s; Mooncake-MC peaks at 5.6× baseline and needs ≈6.7 min (to $t \approx 1000$ s) to recover

Table 5. TTFT improvement over Mooncake-MC by model

Model	Med. TTFT ×	P99 TTFT ×
Llama-3-8B	2.4	3.1
Llama-3-70B	3.8	4.6
Mixtral-8x22B	4.2	5.4

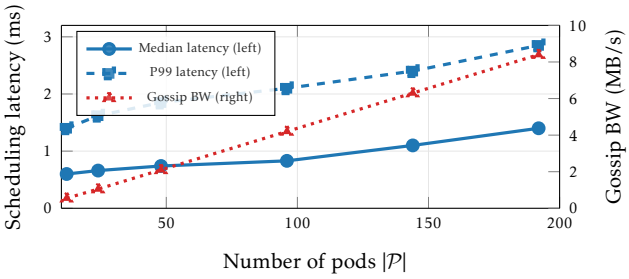

Figure 15. Scalability: sub-linear scheduling latency in $|\mathcal{P}|$ (solid: median, dashed: P99), linear gossip bandwidth

Table 6. Multi-tenant performance under weights 5:2:1

Tenant	SLO target	Janus	WRR baseline
A (chat)	1.5 s P99 TTFT	**1.21 s**	2.70 s ×
B (RAG)	8 s P99 TTFT	**5.4 s**	6.8 s
C (summary)	200 tok/s	**217 tok/s**	184 tok/s ×

caches make transport-policy choice more impactful. Against an offline LP-relaxation oracle with full future knowledge and no decomposition restriction, Janus achieves 89.4% of oracle cost and 92.1% of oracle TTFT—consistent with the composition of the $\frac{1}{2}/(1-1/e)$ submodular bound (Proposition 1), the $O(1/V)$ Lyapunov gap, and the decomposition gap Δ_{seq} (Theorem 3), and bounding the sum of all three at ≈11% on our traces. Scaling pods from 12 to 192, scheduling latency grows sub-linearly (0.6 → 1.4 ms) and gossip bandwidth linearly (Figure 15).

3.7. Multi-Tenant Fairness, Reuse, and Egress

With three tenants (chat 500 req/s, RAG 80 req/s, summarization 12 req/s) under weights 5:2:1, Janus keeps all tenants within SLO while a weighted round-robin baseline violates chat's 1.5 s P99 SLO at 2.7 s under burst (Table 6). Janus sustains a 71% reuse-capture rate over time (1.5× Mooncake-MC, 9× RR; Figure 16) and reduces inter-cloud egress 3.2–4.0× through judicious recompute/hybrid/selective selection (Figure 17).

3.8. Long-Context, Energy, and Forecasting

On a 32K–128K-token long-context workload (KV cache 2–10 GB/request), Migrate becomes infeasible across most cross-cloud edges; in simulation Janus selects

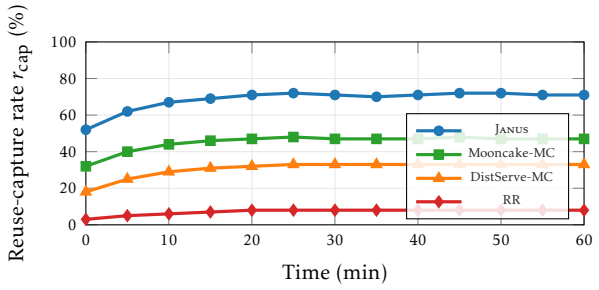

Figure 16. Reuse-capture rate r_{cap} over time. Janus achieves and sustains 71%

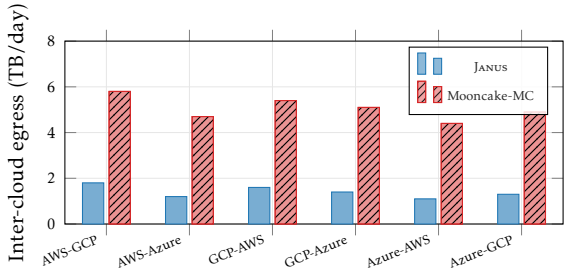

Figure 17. Daily inter-cloud KV egress by direction. Janus reduces egress by 3.2–4.0×

Hybrid in 64% of cases and Selective in 22%, achieving median TTFT 2.31 s vs. 11.4 s for Mooncake-MC (4.9×).

Energy (first-order analytical estimate). We do not have wall-plug measurements, so we give an order-of-magnitude model rather than a measured figure. Transporting a byte costs on the order of 0.4–0.8 J/MB (intra-cloud RDMA vs. encrypted cross-cloud, from published NIC/switch power envelopes). *Recomputing* a prefill, by contrast, runs an 8×H100 pod (≈5.6 kW at 700 W/GPU) for T_r^{pf} seconds: for a 1024-token Llama-3-70B prefill, $T_r^{\text{pf}}(1024, 0) \approx 0.087$ s (Table E.1), so the recompute energy is $\approx 5.6\,\text{kW} \times 0.087\,\text{s} \approx 4.9 \times 10^2$ J, i.e. on the order of hundreds of joules—consistent with the work performed, since a 1024-token forward pass is $\approx 2N_{\text{params}}L \approx 1.4 \times 10^{14}$ FLOPs and cannot cost only tens of joules at any realistic pod throughput. The KV for that prefill is $S(1024) = 160$ MiB, so migrating it costs $\approx 160 \times 0.78 \approx 1.3 \times 10^2$ J. Hence, for this size, *Migrate is* ≈ 4× *more energy-efficient than Recompute*, and the advantage grows with length because recompute energy scales super-linearly (attention $O(L^2)$) while transfer energy scales linearly. The energy-optimal policy is therefore to migrate whenever bandwidth permits; Janus chooses Recompute/Hybrid to meet *latency and dollar* objectives under bandwidth scarcity, trading energy for those. We deliberately make no percentage energy-savings claim versus [49] absent a measured power trace; quantifying end-to-end energy on hardware is future work.

Forecasting. On the replayed trace, the Holt-Winters forecaster used by S4 captured 91% of 15-min spot-price variance (an LSTM added only 1.8% at 11× inference cost), and per-request decision logs of (9)–(10) let us reconstruct, in the simulator, the placement rationale for any request—useful for diagnosing induced radix-staleness anomalies.

3.9. Comparison with Recent Network- and Cache-Aware Disaggregated Serving

The closest contemporary systems target either cross-datacenter KV transfer or load-aware KV scheduling, but none treats the *transport policy itself* as an online decision variable with joint submodular-plus-Lyapunov guarantees across heterogeneous clouds. Table 7 positions Janus against them along the axes that matter for multi-cloud disaggregation. MemServe [55] introduces an elastic memory pool and a global prompt tree unifying context caching with disaggregation, but within one datacenter. P/D-Serve [56] scales prefill/decode disaggregation across tens of thousands of NPUs with fine-grained organization, again intra-operator. KVDirect [57] optimizes the KV-transfer datapath itself (pull-based, tensor-centric) for distributed inference. FlowKV [58] adds load-aware scheduling and low-latency KV transfer. LMCache [59] provides a reusable KV-connector layer that persists and shares caches. The most directly comparable is *Prefill-as-a-Service / cross-datacenter KVCache* [60], from the Mooncake lineage, which performs PD disaggregation across clusters by streaming KVCache between datacenters. Janus differs in three respects: (i) the KV-transport *policy* (Migrate/Recompute/Hybrid-k/Selective) is a first-class scheduling variable chosen per request rather than a fixed streaming datapath; (ii) the formulation spans *heterogeneous* GPU SKUs and *multiple* clouds with spot-price arbitrage, not a single operator's homogeneous fleet; and (iii) we provide provable guarantees (monotone-submodular reuse with $\frac{1}{2}/(1 - 1/e)$ bounds, $O(1/V)$ Lyapunov cost-optimality within the decomposed class, and universal-scheduling robustness) rather than purely empirical results.

3.10. Fault-Injection Case Studies

To complement the aggregate experiments, we script three fault-injection scenarios in the simulator, each exercising a different component. These are constructed illustrations, not operational incident reports; the numbers are simulator outputs under the injected fault.

Case A: Prefix-Sharing RAG Storm. A retrieval-augmented workload issues queries sharing a 12 KB system prompt and a 4 KB tool-schema preamble; we inject a burst of 3,400 near-identical requests over 12 minutes. Janus's S1 concentrates the load on the two

Table 7. Feature comparison with recent disaggregated-serving systems. PD: prefill/decode disaggregation; XC: cross-cloud/cross-DC; Het.: heterogeneous SKUs; Policy: transport-policy choice as a decision variable; Spot: spot-price arbitrage; Guar.: provable guarantees

System	PD	XC	Het.	Policy	Spot	Guar.
Mooncake [7]	✓	—	—	—	—	—
DistServe [5]	✓	—	—	—	—	—
MemServe [55]	✓	—	—	—	—	—
P/D-Serve [56]	✓	—	—	—	—	—
KVDirect [57]	✓	✓	—	—	—	—
FlowKV [58]	✓	✓	—	partial	—	—
LMCache [59]	✓	✓	—	—	—	—
Prefill-aaS [60]	✓	✓	—	—	—	—
Janus	✓	✓	✓	✓	✓	✓

prefill pods holding the shared prefix, raising their per-pod reuse-capture to 94%; the Mooncake-MC fan-out dilutes it to 51%. The simulator's decision logs confirm the marginal reuse gain in (9) dominates the cost threshold during the spike, as Lemma 1 and Proposition 1 predict. Spike goodput is 2.7× higher under Janus.

Case B: Bandwidth Collapse. We inject a cross-cloud link degradation cutting AWS↔GCP goodput from 9.4 to 1.6 Gb/s for 38 minutes. The modeled telemetry surfaces the drop within one 100 ms window; S2 shifts the AWS→GCP policy mix from 62/24/10/4% to 8/44/34/14% (M/H/R/Σ). Throughput falls 19%, P99 TTFT rises 1.3×, and no SLO-class violations occur. Mooncake-MC (no Recompute/Hybrid path) suffers an 82% throughput collapse and 7.4× P99 inflation, because every request keeps attempting Migrate on the throttled link until timeout.

Case C: Quota Exhaustion. We inject an exhaustion of the GCP H100 quota in `us-central1`. Because S4's tail-risk inflator had pre-warmed 6 reserve H100 replicas in AWS `us-west-2` in the preceding epoch, Janus absorbs the lost GCP capacity with a 14% transient cost increase and no TTFT regression; when the quota is restored 19 minutes later, S4 de-allocates the reserve. The scheduler emits an explanatory event listing affected pods, the absorbing reserve, and the projected cost overshoot.

4. Discussion

4.1. Related Work

Disaggregated LLM serving. DistServe [5], Splitwise [6], Mooncake [7], and Llumnix [15] pioneered prefill/decode separation, all assuming a single cluster. A second wave scales or distributes disaggregation: MemServe [55] unifies context caching and disaggregation through an elastic memory pool and global prompt tree; P/D-Serve [56] organizes prefill/decode at tens-of-thousands-of-device scale. Janus extends disaggregation to heterogeneous multi-cloud fleets and makes the KV-transport policy a first-class, per-request scheduling decision.

Cross-datacenter and network-aware KV transfer. The transport datapath has become its own research thread: KVDirect [57] builds a pull-based, tensor-centric KV-transfer mechanism for distributed inference; FlowKV [58] couples low-latency KV transfer with load-aware scheduling; LMCache [59] offers a reusable KV-connector layer that persists and shares caches across engines. Closest to our setting, the Prefill-as-a-Service / cross-datacenter KVCache line [60] streams KVCache between datacenters to disaggregate prefill and decode across clusters. These works optimize *how* to move the cache; Janus instead decides *whether* to move, recompute, hybridize, or selectively ship it—per request, under price and bandwidth volatility—and composes that decision with prefix-reuse placement under provable guarantees (Section 3.9).

Prefix-cache-aware routing. SGLang [4], vLLM [11], and llm-d [3] exploit prefix sharing within a single radix index, and CachedAttention [40] persists caches across multi-turn conversations. We generalize to gossip-replicated multi-cloud indices and give a monotone-submodular reuse objective with $\frac{1}{2}/(1-1/e)$ guarantees (Proposition 1).

KV-cache management. CacheGen [39] compresses and streams the cache; InfiniGen [41] and H2O [42] evict adaptively for long contexts. Grouped-query attention [61] shrinks the cache architecturally—a factor we model exactly in (1). These reduce or reshape the cache but assume a single locus; Janus decides where it should physically reside.

Orthogonal inference optimizations. Speculative decoding [38], offloaded inference [43, 44], and weight quantization [45, 46] reduce per-request work within a pool, complementary to Janus's routing.

Cross-cloud and elastic serving. Helix [21] optimizes placement on heterogeneous single-cluster GPUs; LoongServe [48] reshapes parallelism for long contexts; ServerlessLLM [47] attacks cold-start latency, a bottleneck we observe in S4 warmup. SkyPilot [19] and Sky Computing [18] address stateless workloads and do not handle KV state.

Queue-based scheduling and classical theory. Continuous batching [11] and iteration-level scheduling in Orca [16] operate intra-pod. Our drift-plus-penalty machinery is classical [13]; the contribution is composing it with submodular prefix placement [12, 53, 54], a combination we have not seen analyzed. Submodular maximization (from the classical greedy analysis of Nemhauser–Wolsey–Fisher [12]) underpins cache and content placement [20]; the matroid-constrained guarantees we invoke are due to

Fisher–Nemhauser–Wolsey [54] (combinatorial greedy, $\frac{1}{2}$), Calinescu–Chekuri–Pál–Vondrák [53] (continuous greedy, $1 - 1/e$), and, for the matroid-plus-knapsack case, Lee–Mirrokni–Nagarajan–Sviridenko [62] (local search). Erasure-coded resilience for prediction serving [17] is complementary to our failure handling (Section 2.5) rather than a scheduling policy.

Cluster schedulers and geo-distributed ML. Borg [25], the Kubernetes lineage [52], Tetris [26], Hawk [27], and Sparrow [28] pack stateless jobs; Gaia [22] optimizes geo-distributed *training*. Early calls for real-time ML systems [29] and distributed caches [30] foreshadowed stateful serving but predate the KV-cache regime; LLM serving's per-request, bursty KV state is a different regime. Spot-arbitrage work [18, 19] targets stateless or checkpoint-restart jobs; Janus's state-aware arbitrage, using transport policy to make the cache itself a routing decision, is to our knowledge novel.

4.2. Limitations and Future Work

The most important limitation is that our end-to-end serving results are from a trace-driven simulator calibrated to single-pod microbenchmarks, not from a physical multi-cloud deployment. The simulator faithfully reproduces the calibrated prefill/transport timing and cost models but does not capture second-order hardware effects—real CUDA/DMA-stream contention (which bounds the assumed hybrid overlap), NIC and PCIe queueing, encryption jitter, tokenizer/version skew, and control-plane/data-plane coupling under load. Validating the headline numbers on a physical fleet, and in particular measuring the realized hybrid overlap efficiency with nsys/ncu traces, is the primary future work; we expect hardware TTFT and cost to be somewhat worse than the idealized simulator, though the qualitative policy-selection behavior (Figure 12) should be robust because it follows directly from the bandwidth/recompute crossover of Figure 2.

A second limitation is metric scope: we evaluate TTFT, goodput, cost, egress, and reuse capture, but not *time per output token* (TPOT). TPOT is affected by cross-cloud decode placement and, in particular, by any mid-stream KV migration, which would amortize a relocation stall across the remaining output tokens. Janus as evaluated never relocates a decode mid-generation, so TPOT is determined by the decode pool's steady per-token rate $1/\mu_q$; extending the scheduler to mid-stream migration, and evaluating the resulting TPOT/TTFT tradeoff, is future work.

Third, the analysis carries an honest residual: Theorem 3 closes the gap to the best *decomposed* policy at rate $O(1/V)$, but the decomposition gap Δ_{seq} of Definition 1 is a constant that no choice of V removes. We bound the sum of all approximation sources empirically (within 89–92% of an unrestricted offline oracle) but do not bound Δ_{seq} analytically; doing so, or replacing the sequential composition with a jointly-optimized relaxation, is open.

Janus also assumes a gossip-replicated radix index; under heavy partitions it grows stale (up to 4.1% stale-prefix rate in a synthetic partition; Proposition 2), which a CRDT-based merge [50] could bound explicitly. The fast-path greedy carries a $\frac{1}{2}$ worst-case bound (and $\frac{1}{3}$ under capacity pressure above 95% utilization), though it tracks the $(1 - 1/e)$ continuous-greedy reference within 3.4% in simulation; an adaptive switch to continuous greedy under pressure is future work. We treat $\rho_p(t)$ as exogenous, but a large operator's routing could shift demand and prices; a game-theoretic extension modelling strategic provider pricing is open. We assume model weights are pre-replicated; cold-pool bootstrap (loading a 140 GB checkpoint) is modeled at 35–90 s and dominates S4 scale-up, motivating fast-loading techniques [23, 47]. Finally, per-request scheduling at 0.83 ms median (measured on the prototype) is non-negligible against the smallest prefills (~30 ms); an approximate fast path that bypasses S2 enumeration for short prompts could halve this overhead.

4.3. Conclusion

We presented Janus, to our knowledge the first scheduler for joint prefill/decode disaggregation across heterogeneous multi-cloud LLM fleets. By formulating per-request scheduling as a constrained graph problem with stateful edges and decomposing it into a monotone-submodular prefix-aware placement and a Lyapunov drift-plus-penalty controller—with the KV-transport policy elevated to a first-class decision variable—Janus attains provable guarantees ($\frac{1}{2}/(1 - 1/e)$ reuse approximation, $O((B+C)/V)$ cost-optimality within the decomposed policy class with $O(V)$ queue bound, an exact integer hybrid layer-split rule with a closed-form relaxation, and universal-scheduling robustness) together with substantial gains *in a calibrated trace-driven simulation*: 3.8× median and 4.6× P99 TTFT reduction, 2.1× goodput, a 71% reuse-capture rate, and 38% cost reduction over the strongest baseline, with graceful degradation under injected spot-preemption, bandwidth-collapse, and region-failure faults. Physical multi-cloud validation of these simulated results is the natural next step. We believe joint state-and-route scheduling—with the KV-transport policy as a first-class decision—will be foundational to multi-cloud and edge-adjacent LLM serving.

E. Calibration Coefficients

Table E.1 lists the prefill-cost coefficients $(\alpha_p, \beta_p, \xi_p)$ obtained by fitting the full-prefill form of (2) (i.e. $|u|$=0,

so $\alpha_p L^2 + \beta_p L + \xi_p$) to 2,400 *measured* single-pod prefills per SKU (prompt lengths log-uniform in [128, 16384]). These microbenchmarks are the only hardware measurements in the paper; they are the single source of truth for every recompute and per-layer time elsewhere, and for the two-argument model (2) they fix the $|u|$=0 behavior exactly (the cached-prefix correction $-\alpha_p|u|^2$ is architectural, not separately fitted). As a check, the H100 TP=8 row gives $T^{\text{pf}}(8192, 0) = 2.3\times10^{-9}\,(8192)^2 + 7.9\times10^{-5}\,(8192) + 0.004 = 0.805$ s (per-layer $b = 10.06$ ms) and $T^{\text{pf}}(16384, 0) = 1.92$ s, matching the recompute curve of Figure 2, the ∼1.9 s recompute quoted in Section 1.1, and the b used in Figure 5 and Table 1.

Table E.1. Prefill-cost coefficients for (2) (Llama-3-70B, FP8 KV). ξ_p is the fixed kernel-launch + KV-reshape overhead

SKU	α_p (s/tok^2)	β_p (s/tok)	ξ_p (ms)	R^2
H100 (TP=4)	4.0×10^{-9}	1.17×10^{-4}	3.1	0.991
H100 (TP=8)	2.3×10^{-9}	7.90×10^{-5}	4.0	0.989
A100 (TP=4)	8.1×10^{-9}	2.20×10^{-4}	3.6	0.987
MI300X (TP=4)	4.5×10^{-9}	1.40×10^{-4}	4.4	0.984
L4 (TP=8, INT8)	2.3×10^{-8}	5.06×10^{-4}	7.9	0.976

F. KV-Cache Size Derivation

Table F.2 derives the per-token and full-context KV-cache sizes used throughout the paper directly from each model's published architecture via (1), $S(\ell) = 2\, N_{\text{layers}}\, n_{\text{kv}}\, d_{\text{head}}\, \ell\, b_{\text{prec}}$. All three evaluated models use grouped-query attention, so the key/value head count n_{kv} is much smaller than the query-head count; using d_{model} in place of $n_{\text{kv}} d_{\text{head}}$ would overcount the cache by the GQA group factor n_q/n_{kv} (e.g. 8× for Llama-3-70B). The per-token size is $2\, N_{\text{layers}}\, n_{\text{kv}}\, d_{\text{head}}\, b_{\text{prec}}$ bytes; we report it at FP8 ($b_{\text{prec}} = 1$). Under tensor parallelism of degree t_p each GPU stores $1/t_p$ of this locally, but the full $S(\ell)$ is what traverses an inter-pod edge (re-sharded if $t_p \neq t_q$).

Table F.2. Exact KV-cache derivation per evaluated model via (1) (FP8 KV, b_{prec}=1 byte). Per-token = $2N_{\text{layers}}n_{\text{kv}}d_{\text{head}}b_{\text{prec}}$

Model	N_{layers}	n_{kv}	d_{head}	/token	@16K
Llama-3-8B	32	8	128	64 KiB	1.0 GiB
Llama-3-70B	80	8	128	160 KiB	2.5 GiB
Mixtral-8x22B	56	8	128	112 KiB	1.75 GiB

For Llama-3-70B the per-token size is $2 \cdot 80 \cdot 8 \cdot 128 \cdot 1 = 163{,}840$ bytes = 160 KiB, hence 320 MiB at ℓ=2048 and 2.5 GiB at ℓ=16384, matching Section 1.1.

G. Decode-Selection Pseudocode (S2)

Algorithm 2 expands S2. It uses the GQA-aware sizes of (1) (S total, S_ℓ per-layer), the recompute latency Λ_R evaluated on the network-local recompute target $r = r(q)$ (Section 2.1, priced at $\tilde{\rho}_r, g_r$), and the two-point integer optimum $k^\star$ of (12).

Algorithm 2 S2: KV-aware decode + transport selection

Input: prefill p, request n, queues Q, goodput W, GPU-sec prices $\tilde{\rho}$, GPUs/replica g, egress Γ, weight V
best $\leftarrow \infty$
for $q \in \mathcal{D}_{\text{compat}}(p)$ **compatible with** residency(n) **do**
  $r \leftarrow$ RecompTarget(q) // network-local prefill-class (TP=8) pool
  $S \leftarrow 2N_{\text{layers}} n_{\text{kv}} d_{\text{head}} L_n b_{\text{prec}}$; $S_\ell \leftarrow S/N_{\text{layers}}$
  $\Omega_q \leftarrow T_r^{\text{pf}}(L_n, |u_n^{(q)}|)/N_{\text{layers}}$ // (3)
  $a \leftarrow S_\ell/W[p,q]$; $b \leftarrow \Omega_q$
  $\Lambda_M \leftarrow S/W[p,q]$; $\Psi_M \leftarrow \Gamma[p,q]\, S$
  $\Lambda_R \leftarrow T_r^{\text{pf}}(L_n, |u_n^{(q)}|) + S_\ell/W[r,q]$; $\Psi_R \leftarrow \tilde{\rho}_r\, g_r\, T_r^{\text{pf}}(L_n, |u_n^{(q)}|)$
  $k_{\mathbb{R}} \leftarrow N_{\text{layers}}\, b/(a+b)$
  $\mathcal{K} \leftarrow \{\lfloor k_{\mathbb{R}} \rfloor, \lceil k_{\mathbb{R}} \rceil\} \cap [0, N_{\text{layers}}]$
  $k^\star \leftarrow \arg\min_{k\in\mathcal{K}} \max(ka, (N_{\text{layers}}-k)b)$ // exact over integers
  $\Lambda_H \leftarrow \max(k^\star a, (N_{\text{layers}}-k^\star)b) + \max(a,b)$
  $\Psi_H \leftarrow \Gamma[p,q]\, k^\star S_\ell + \tilde{\rho}_r g_r (N_{\text{layers}}-k^\star)\Omega_q$
  $\Lambda_\Sigma \leftarrow \max\big(S(|u_n|)/W[p,q],\ T_r^{\text{pf}}(L_n,\ L_n-|v_n|)\big)$ // suffix attends to full prefix
  $\Psi_\Sigma \leftarrow \Gamma[p,q]\, S(|u_n|) + \tilde{\rho}_r g_r\, T_r^{\text{pf}}(L_n,\ L_n-|v_n|)$
  for $\phi \in \{M, R, H_{k^\star}, \Sigma\}$ **do**
    $c \leftarrow \Lambda_\phi + V\,\Psi_\phi + Q_q\,\delta_n(q)$ // seconds; V in s/$
    if $c <$ best **then**
      best $\leftarrow c$; $(q^\star, \phi^\star) \leftarrow (q, \phi)$
    end if
  end for
end for
return $(q^\star, \phi^\star)$

H. Pool-Sizing Pseudocode (S4)

I. Notation Summary

Acknowledgements. **Funding.** This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors. **Ethics approval.** Ethical approval was not required. The study did not involve human participants, their identifiable data or tissue, or animals. All evaluation was performed in software (single-pod microbenchmarks and a trace-driven simulator); the assistant trace used to derive the synthetic workload was de-identified prior to any analysis and contains no personally identifiable information.

Algorithm 3 S4: Pool size adjustment (every $\Delta = 30$ s)

Input: pools $\mathcal{P}$, prices ρ, forecast $\hat{\lambda}$, queues Q
$\eta^\star \leftarrow \eta_{\text{current}}$
for $p \in \mathcal{P}$ **do**
  $\hat{u}_p \leftarrow \hat{\lambda}_p / \mu_p$ // forecast utilization
  if $\hat{u}_p > 0.85$ **or** $Q_p > Q_{\text{high}}$ **then**
    $\eta_p^\star \leftarrow \lceil \eta_p(1+\gamma_\uparrow) \rceil$
  else if $\hat{u}_p < 0.40$ **and** $Q_p < Q_{\text{low}}$ **then**
    $\eta_p^\star \leftarrow \max(\eta_p^{\min}, \lfloor \eta_p(1-\gamma_\downarrow) \rfloor)$
  end if
  $\eta_p^\star \leftarrow \eta_p^\star + \lceil \kappa_{\text{tr}} \sqrt{\eta_p^\star} \rceil$ // tail-risk inflator, per pool
end for
Clip $\sum_{p \in r} \eta_p^\star g_p m_p^{\text{gpu}} \le M_r$ for every region r; commit via Raft

Table I.3. Notation reference

Symbol	Meaning (units)
$\mathcal{P}_{\text{pf}}, \mathcal{P}_{\text{dec}}$	prefill / decode pool sets
$\pi(p)$	phase role of pool p
η_p	replica count of pool p
$g_p = t_p z_p$	GPUs per replica (TP × PP)
t_p, z_p	tensor- / pipeline-parallel degree
m_p^{gpu}	per-GPU HBM (bytes)
$\kappa_p \in (0,1]$	HBM fraction usable for KV
κ_{tr}	tail-risk inflator coefficient (S4)
$\gamma_\uparrow, \gamma_\downarrow$	S4 scale-up / scale-down step
$\mathcal{D}_{\text{compat}}(p)$	decode pools compatible with p
$\mathcal{T}, \mathcal{T}^{-1}(u)$	radix index, owner set of prefix u
x_n, u_n, v_n, L_n	prompt, cached prefix, suffix, length
$u_n^{(q)}$	prefix of x_n already held by pool q
N_{layers}	transformer layer count
$n_{\text{kv}}, d_{\text{head}}$	KV-head count, head dim (GQA)
b_{prec}	KV bytes per element
$S(\ell), S_\ell$	total / per-layer KV size, (1) (bytes)
T_p^{pf}	prefill latency, (2) (s)
α_p, β_p, ξ_p	prefill-cost coefficients
Ω_q	per-layer recompute time, (3) (s/layer)
$r(q), g_r, \tilde{\rho}_r$	recompute target pool, its GPUs/replica, price
r_p, c_p	region, cloud of pool p
Λ_ϕ, Ψ_ϕ	latency (s), cost ($) of transport ϕ
$W_{pq}, \Gamma_{pq}, e_{pq}^{\text{eff}}$	goodput (bytes/s), egress ($/byte), link efficiency
$\rho_p, \tilde{\rho}_p$	spot price ($/GPU-hr), ($/GPU-s)
μ_q	per-replica decode rate (tok/s)
$\delta_n(q), \bar{\delta}_p$	decode / mean prefill queue delay (s)
Q_q	virtual-queue congestion price (dimensionless)
V, V_{slow}	price weights, latency valuation (s/$)
$\tau_\sigma^\star, \epsilon_\sigma$	TTFT target and violation budget of class σ
$G(\mathcal{A})$	monotone-submodular reuse objective
γ_{apx}	S1 approximation factor ($\frac{1}{2}$ or $1-1/e$)
B, C, C'	Lyapunov drift, approx.-gap, boundary constants
Δ_{seq}	decomposition gap, Definition 1 ($)
$k^\star$	hybrid integer layer split, (12)
h	prefix-hit rate (trace property)
r_{cap}	reuse-capture rate (scheduler property)
ζ	radix stale-entry rate
τ_g	gossip period (s)
residency(n)	data-residency constraint of request n

Conflict of interest. The authors declare that they have no conflict of interest. **Data and code availability.** The JANUS control-plane prototype, the vLLM data-plane extensions, the discrete-event simulator, the multi-cloud baseline shim, the synthetic-trace generator, the calibration harness producing Tables E.1 and F.2, and all figure-plotting scripts are publicly available at https://github.com/arunakbofficial-tech/Janus. The proprietary raw trace is not released owing to commercial confidentiality; the released statistically matched synthetic trace reproduces every reported experiment end-to-end. **Author contributions.** All authors contributed to the study conception, design, analysis, and writing of the manuscript, and all authors read and approved the final manuscript.

References

[1] Agrawal A, Kedia N, Panwar A, Mohan J, Kwatra N, Gulavani B, et al. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2024.

[2] Kwon W, Li Z, Zhuang S, Sheng Y, Zheng L, Yu CH, et al. Efficient memory management for large language model serving with PagedAttention. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP); 2023.

[3] Red Hat, IBM, Google, NVIDIA, CoreWeave, and contributors. llm-d: A Kubernetes-native distributed inference framework; 2025. Available from: https://github.com/llm-d/llm-d.

[4] Zheng L, Yin L, Xie Z, Huang J, Sun C, Yu CH, et al. SGLang: Efficient execution of structured language model programs. In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS); 2024.

[5] Zhong Y, Liu S, Chen J, Hu J, Zhu Y, Liu X, et al. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2024.

[6] Patel P, Choukse E, Zhang C, Shah A, Goiri I, Maleki S, et al. Splitwise: Efficient generative LLM inference using phase splitting. In: Proceedings of the International Symposium on Computer Architecture (ISCA); 2024.

[7] Qin R, Li Z, He W, Zhang M, Wu Y, Zheng W, et al. Mooncake: A KVCache-centric disaggregated architecture for LLM serving. In: Proceedings of the USENIX Conference on File and Storage Technologies (FAST); 2025.

[8] Dao T, Fu DY, Ermon S, Rudra A, Re C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS); 2022.

[9] Hu J, Wu X, Liu D, Yang F, Yang Z, Zhou L. Benchmarking multi-cloud LLM inference under heterogeneous accelerators and pricing. In: Proceedings of the Conference on Machine Learning and Systems (MLSys); 2025.
[10] Pope R, Douglas S, Chowdhery A, Devlin J, Bradbury J, Heek J, et al. Efficiently scaling transformer inference. In: Proceedings of the Conference on Machine Learning and Systems (MLSys); 2023.
[11] Kwon W, Li Z, et al. vLLM: Easy, fast, and cheap LLM serving with PagedAttention; 2023. Available from: https://github.com/vllm-project/vllm.
[12] Nemhauser GL, Wolsey LA, Fisher ML. An analysis of approximations for maximizing submodular set functions—I. Mathematical Programming. 1978;14(1):265–294.
[13] Neely MJ. Stochastic Network Optimization with Application to Communication and Queueing Systems. Morgan & Claypool; 2010.
[14] Dao T. FlashAttention-2: Faster attention with better parallelism and work partitioning. In: Proceedings of the International Conference on Learning Representations (ICLR); 2024.
[15] Sun B, Huang Z, Zhao H, Xiao W, Zhang X, Li Y, et al. Llumnix: Dynamic scheduling for large language model serving. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2024.
[16] Yu GI, Jeong JS, Kim GW, Kim S, Chun BG. Orca: A distributed serving system for transformer-based generative models. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2022.
[17] Kosaian J, Rashmi KV, Venkataraman S. Parity models: Erasure-coded resilience for prediction serving systems. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP); 2019.
[18] Stoica I, Shenker S. From cloud computing to sky computing. In: Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS); 2021.
[19] Yang Z, Wu Z, Luo M, Chiang WL, Bhardwaj R, Kwon W, et al. SkyPilot: An intercloud broker for sky computing. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI); 2023.
[20] Dehghan M, Jiang B, Seetharam A, He T, Salonidis T, Kurose J, et al. On the complexity of optimal request routing and content caching in heterogeneous cache networks. IEEE/ACM Transactions on Networking. 2017;25(3).
[21] Mei Y, Zhuang Y, Miao X, Yang J, Jia Z, Vinayak R. Helix: Distributed serving of large language models via max-flow on heterogeneous GPUs. In: Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS); 2025.
[22] Hsieh K, Harlap A, Vijaykumar N, Konomis D, Ganger GR, Gibbons PB, et al. Gaia: Geo-distributed machine learning approaching LAN speeds. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI); 2017.
[23] Ren J, Rajbhandari S, Aminabadi RY, Ruwase O, Yang S, Zhang M, et al. ZeRO-Offload: Democratizing billion-scale model training. In: Proceedings of the USENIX Annual Technical Conference (ATC); 2021.
[24] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In: Proceedings of the USENIX Annual Technical Conference (ATC); 2014.
[25] Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J. Large-scale cluster management at Google with Borg. In: Proceedings of the European Conference on Computer Systems (EuroSys); 2015.
[26] Grandl R, Ananthanarayanan G, Kandula S, Rao S, Akella A. Multi-resource packing for cluster schedulers. In: Proceedings of the ACM SIGCOMM Conference; 2014.
[27] Delgado P, Dinu F, Kermarrec AM, Zwaenepoel W. Hawk: Hybrid datacenter scheduling. In: Proceedings of the USENIX Annual Technical Conference (ATC); 2015.
[28] Ousterhout K, Wendell P, Zaharia M, Stoica I. Sparrow: Distributed, low-latency scheduling. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP); 2013.
[29] Nishihara R, Moritz P, Wang S, Tumanov A, Paul W, Schleier-Smith J, et al. Real-time machine learning: The missing pieces. In: Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS); 2017.
[30] Nishtala R, Fugal H, Grimm S, Kwiatkowski M, Lee H, Li HC, et al. Scaling Memcache at Facebook. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI); 2013.
[31] Shoeybi M, Patwary M, Puri R, LeGresley P, Casper J, Catanzaro B. Megatron-LM: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053. 2019.
[32] Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, et al. Language models are few-shot learners. In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS); 2020.
[33] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS); 2017.
[34] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux MA, Lacroix T, et al. LLaMA: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971. 2023.
[35] Touvron H, Martin L, Stone K, Albert P, Almahairi A, Babaei Y, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288. 2023.
[36] Jiang AQ, Sablayrolles A, Roux A, Mensch A, Savary B, Bamford C, et al. Mixtral of experts. arXiv preprint arXiv:2401.04088. 2024.
[37] Dubey A, Jauhri A, Pandey A, Kadian A, Al-Dahle A, Letman A, et al. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783. 2024.
[38] Leviathan Y, Kalman M, Matias Y. Fast inference from transformers via speculative decoding. In: Proceedings of the International Conference on Machine Learning (ICML); 2023.
[39] Liu Y, Li H, Cheng Y, Ray S, Huang Y, Zhang Q, et al. CacheGen: KV cache compression and streaming for

fast large language model serving. In: Proceedings of the ACM SIGCOMM Conference; 2024.
[40] Gao B, He Z, Sharma P, Kang Q, Jevdjic D, Deng J, et al. Cost-efficient large language model serving for multi-turn conversations with CachedAttention. In: Proceedings of the USENIX Annual Technical Conference (ATC); 2024.
[41] Lee W, Lee J, Seo J, Sim J. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2024.
[42] Zhang Z, Sheng Y, Zhou T, Chen T, Zheng L, Cai R, et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS); 2023.
[43] Sheng Y, Zheng L, Yuan B, Li Z, Ryabinin M, Chen B, et al. FlexGen: High-throughput generative inference of large language models with a single GPU. In: Proceedings of the International Conference on Machine Learning (ICML); 2023.
[44] Aminabadi RY, Rajbhandari S, Awan AA, Li C, Li D, Zheng E, et al. DeepSpeed-Inference: Enabling efficient inference of transformer models at unprecedented scale. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC); 2022.
[45] Lin J, Tang J, Tang H, Yang S, Chen WM, Wang WC, et al. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In: Proceedings of the Conference on Machine Learning and Systems (MLSys); 2024.
[46] Frantar E, Ashkboos S, Hoefler T, Alistarh D. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In: Proceedings of the International Conference on Learning Representations (ICLR); 2023.
[47] Fu Y, Xue L, Huang Y, Brabete AO, Ustiugov D, Patel Y, et al. ServerlessLLM: Low-latency serverless inference for large language models. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2024.
[48] Wu B, Liu S, Zhong Y, Sun P, Liu X, Jin X. LoongServe: Efficiently serving long-context large language models with elastic sequence parallelism. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP); 2024.
[49] Stojkovic J, Zhang C, Goiri I, Torrellas J, Choukse E. DynamoLLM: Designing LLM inference clusters for performance and energy efficiency. In: Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA); 2025.
[50] Shapiro M, Preguica N, Baquero C, Zawirski M. Conflict-free replicated data types. In: Proceedings of the Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS); 2011.
[51] Iyengar J, Thomson M. QUIC: A UDP-based multiplexed and secure transport. IETF; 2021. RFC 9000.
[52] Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J. Borg, Omega, and Kubernetes. Communications of the ACM. 2016;59(5):50–57.
[53] Calinescu G, Chekuri C, Pal M, Vondrak J. Maximizing a monotone submodular function subject to a matroid constraint. SIAM Journal on Computing. 2011;40(6):1740–1766.
[54] Fisher ML, Nemhauser GL, Wolsey LA. An analysis of approximations for maximizing submodular set functions—II. Mathematical Programming Study. 1978;8:73–87.
[55] Hu C, Huang H, Hu J, Xu J, Chen X, Xie T, et al. MemServe: Context caching for disaggregated LLM serving with elastic memory pool. arXiv preprint arXiv:2406.17565. 2024.
[56] Jin Y, Wang T, Lin H, Song M, Li P, Ma Y, et al. P/D-Serve: Serving disaggregated large language model at scale. arXiv preprint arXiv:2408.08147. 2024.
[57] Chen Y, Xu Y, Jin M, Liu J, Wu C. KVDirect: Distributed disaggregated LLM inference. arXiv preprint arXiv:2501.14743. 2025.
[58] Cui Z, Zhao S, Liu Y, et al. FlowKV: Enhancing multi-turn conversational coherence and low-latency KV-cache scheduling in disaggregated inference. arXiv preprint arXiv:2504.03775. 2025.
[59] Liu Y, Yao J, Li H, Cheng Y, Du K, Lu S, et al. LMCache: A KV-cache connector layer for fast and reusable LLM serving. arXiv preprint arXiv:2510.09665. 2025.
[60] Qin R, He W, Wang Y, Li Z, Xu X, Wu Y, et al. Prefill-as-a-Service: KVCache of next-generation models could go cross-datacenter. arXiv preprint arXiv:2604.15039. 2026.
[61] Ainslie J, Lee-Thorp J, de Jong M, Zemlyanskiy Y, Lebron F, Sanghai S. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP); 2023.
[62] Lee J, Mirrokni VS, Nagarajan V, Sviridenko M. Non-monotone submodular maximization under matroid and knapsack constraints. In: Proceedings of the ACM Symposium on Theory of Computing (STOC); 2009. p. 323–332.