ARI: ADAPTIVE RULE-INTERPOLATED KV CACHE MIGRATION SCHEDULING FOR MULTI-GPU LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model serving maintains a request-level key-value (KV) cache that grows during decoding. Although PagedAttention reduces within-instance fragmentation, request growth and completion can imbalance multi-GPU placement. Existing routing and migration mechanisms leave request-level migration value under interacting memory, transfer-cost, and remaining-duration conditions insufficiently modeled. We present ARI, an adaptive rule-interpolated decision layer for runtime Live-KV migration. After a global imbalance detector fires, candidates are represented by source pressure, KV size, estimated migration cost, global imbalance, and remaining execution time. Gaussian rule matching and normalized consequent interpolation yield continuous migration priorities. A separate destination controller plans feasible moves on a virtual cluster state, stopping at the first balanced prefix along its greedy path (minimum Top-k) when one exists. The priority rule base grows only in dense, insufficiently covered regions, and observed post-migration utility updates rule confidence. ARI retains MELL's initial allocation and reuses request-level Direct-KV or Token-RePrefill execution paths. Across six model–platform settings on 4 and 8 GPUs, ARI achieves higher output-token throughput than MELL. On 4-GPU Llama-2-7b-chat, throughput increases from 436.82 to 675.31 tokens/s (54.6%) and P95 latency decreases by 30.9%. In selected settings, ARI also improves serving performance or migration efficiency over other baselines. Upon acceptance, we will publicly release all source code and experimental scripts to support reproducibility.

1 INTRODUCTION

LLM inference consists of a parallel prefill phase followed by autoregressive decoding (Vaswani et al., 2017). During decoding, each request retains the keys and values of all previous tokens so that later iterations avoid recomputing the entire prefix. The resulting KV cache grows with sequence length and can become a primary memory constraint (Kwon et al., 2023; Liu et al., 2025). Continuous batching and PagedAttention improve iteration-level utilization and reduce fragmentation inside an inference instance (Yu et al., 2022; Kwon et al., 2023). Recent ICLR work improves attention execution through better work partitioning (Dao, 2024), supports streaming with attention sinks (Xiao et al., 2024), and reduces retained KV state through adaptive compression (Ge et al., 2024). These advances do not by themselves guarantee that a placement remains balanced across replicated GPU workers. Output lengths are not known exactly at admission time (Liu et al., 2025; Srivatsa et al., 2024); requests grow at different rates and release their state at different times. Consequently, one GPU may approach its safe KV capacity while another remains underutilized, causing queueing, preemption, recomputation, rejection, or unnecessary GPU activation.

Which active request to migrate remains unresolved. Llumnix supports runtime rescheduling through live request migration (Sun et al., 2024), while MELL jointly considers dynamic KV occupancy and costly migration when placing and moving requests (Liu et al., 2025). Neither a load-driven source-target action nor a categorical allocation rule assigns each active request a continuous, context-dependent value. A fixed linear score can consume several features, but it assumes that each feature has the same additive marginal effect in every state. That assumption misses conditional interactions. A large KV cache is attractive when source pressure is high, transfer cost is low, and substantial

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

decoding remains. The same cache may have negative net value when host-staged transfer is expensive and the request is nearly complete.

The second decision is where and how many requests to migrate. Low-interruption migration mechanisms establish that live state can move across serving instances (Fu et al., 2024), but selecting the GPU with the most free memory ignores compute load, queueing, link cost, and the imbalance created after transfer. Disaggregated systems also redistribute KV state across prefill and decode resources (Zhong et al., 2024; Qin et al., 2025; He et al., 2025); their phase-placement decisions differ from selecting active requests to move among replicated workers. Likewise, fixed Top-1 can fail to remove a hotspot, whereas fixed Top-3 or Top-5 can overmigrate. The request ordering, destination, and migration-set size should therefore be planned together. The planner should stop at the first balanced prefix along its greedy path when available; this does not require a search for a globally minimum migration subset.

We introduce ARI, an adaptive rule-interpolated scheduler above MELL's initial allocator and request-level migration executor. MELL places new requests (Liu et al., 2025). A runtime monitor then tracks GPU KV usage, queue and compute load, active-request KV size, generated tokens, and recent per-token latency. Only the global imbalance detector initiates a planning round. ARI builds candidate states, interpolates a continuous priority from a sparse rule base, selects a feasible destination with a fixed destination controller, and virtually applies request-target pairs until the cluster becomes balanced. A separate backend selector chooses Direct-KV transfer or Token-RePrefill under its configured execution policy. Once execution finishes, ARI converts the realized balance improvement, memory release, migration time, and added latency into utility; the residual updates rule confidence, while repeated undercovered states may create new priority rules.

Our contributions are threefold. **First**, we combine source pressure, request KV size, migration cost, global imbalance, and remaining duration through adaptive rule interpolation to produce a continuous migration priority. **Second**, we adapt the sparse rule base using coverage and observed utility: recurrent, poorly covered states can create new rules, while migration feedback updates rule confidence. **Third**, we couple request ranking with destination-aware planning that accepts feasible, imbalance-reducing moves and stops at the first balanced prefix along its greedy path when one exists.

2 Related Work

LLM serving and KV-cache management. Orca introduced iteration-level scheduling for generative models (Yu et al., 2022), and vLLM's PagedAttention reduced internal KV-cache fragmentation (Kwon et al., 2023). DistServe separated prefill and decode resources to optimize goodput (Zhong et al., 2024). Splitwise similarly separates prompt computation from token generation (Patel et al., 2024), while Mooncake organizes a disaggregated KV cache around the two phases (Qin et al., 2025). BanaServe rebalances disaggregated serving using layer-level weight migration, attention-level KV migration, and a shared KV store (He et al., 2025). Its migration units and prefill/decode resource allocation differ from ARI's request-level placement among replicas. FlashAttention-2 improves GPU attention execution (Dao, 2024); StreamingLLM retains initial attention-sink tokens together with a recent context window (Xiao et al., 2024). FastGen selects cache-retention policies from attention profiling (Ge et al., 2024). SGLang reuses prefix KV states through RadixAttention (Zheng et al., 2024b), and Preble schedules distributed prompt sharing with KV reuse (Srivatsa et al., 2024). DuoAttention allocates different cache spans to retrieval and streaming heads (Xiao et al., 2025), whereas SqueezeAttention manages KV budgets across layers and sequence positions (Wang et al., 2025).

These work optimizes attention, retained cache contents, reuse, batching, or phase placement. Our focus is moving active requests with existing KV state after cross-instance imbalance emerges; integration with compressed caches requires separate implementation and evaluation. Llumnix provides dynamic runtime rescheduling and an efficient live-migration path (Sun et al., 2024); ServerlessLLM reduces model loading and migration interruption in serverless environments (Fu et al., 2024). MELL is the closest KV-centric system: it manages online multi-GPU placement and chooses between KV transfer and token-based reconstruction while limiting migration (Liu et al., 2025). ARI does not redesign these transfer operations. It adds continuous candidate valuation, online rule adaptation, destination-aware planning, and a state-dependent minimum Top-k above MELL's initial placement and request-level execution boundary. FastServe's preemptive queueing (Wu et al.,

2023) and L4's length-aware inter-instance rescheduling (Yuan et al., 2025) provide additional scheduling baselines with different primary objectives.

Kernel interpolation and online rule adaptation. Recent surveys describe inference from sparse rule bases (Li et al., 2021) and kernel-based representations of similarity (Ghojogh et al., 2021). ARI uses Gaussian matching to blend scalar rule consequents, rather than reproducing a particular fuzzy-set transformation procedure. Each rule stores a state center, feature scale, priority consequent, and confidence. Interpolation falls back to zero when total weighted match is negligible. A local sample-density statistic and coverage test restrict growth to recurrent, undercovered feedback regions. The migration-specific representation and observed-utility update define the proposed scheduling mechanism; the background surveys do not establish its empirical effectiveness. The output ranks candidates after a separate imbalance trigger. Confidence adaptation is a heuristic, not a guarantee of calibrated utility prediction or convergence.

3 METHOD

3.1 PROBLEM FORMULATION

Consider N replicated GPU workers $\mathcal{G} = \{g_1, \ldots, g_N\}$. GPU g_j exposes a logical KV-cache pool of capacity M_j^{KV}. At time t, the active request set is $\mathcal{R}_t = \{r_1, \ldots, r_m\}$; request r_i resides on $g(r_i)$ and owns K_i bytes of live KV state. MELL performs the initial allocation. We study only the subsequent runtime request-level KV-cache migration decision: which active request should move, to which feasible GPU, and how many requests must move. KV compression, quantization, reuse, and eviction are outside the scope.

The imbalance detector uses the coefficient of variation (CV) of logical KV utilization across workers. A state is balanced when its CV is at most a configured threshold τ_I. Request occupancy is estimated from token history with block rounding, rather than equated to total device memory measured by a hardware monitor. Appendix A.1 gives the elementary definitions.

Capacity constraints are checked separately during destination selection; the balance predicate does not include entry/exit hysteresis. Per-request cooldown and concurrent-migration limits belong to runtime reservation. Token-progress boundaries can invoke planning, but a balanced snapshot returns an empty plan. Neither a high request score nor a rule-growth condition independently triggers migration. The objective is an early-stopping feasible sequence of virtual CV-reducing moves, not a guarantee of positive realized utility or globally minimum migration count.

3.2 METHOD OVERVIEW

Figure 1 shows the decision and feedback loop. After imbalance is detected, the core candidate builder considers assigned requests with a feasible target for feature extraction; it does not explicitly restrict sources to a separately classified hotspot set. The ranking module maps candidate states to priorities. The planner evaluates destinations and virtually applies improving moves until balance returns or candidates are exhausted. Runtime reservation may admit only a subset of the planned assignments. Feedback from execution updates the request-priority mechanism.

The decision layer outputs an ordered schedule; execution handles fencing, transfer or reconstruction, ownership changes, and failure recovery. A plan does not prove transfer completion, and its prefix length can differ from committed migrations.

3.3 CANDIDATE REQUEST REPRESENTATION

The normalized vector is $\mathbf{x}_i = [P_i, \widetilde{K}_i, \widetilde{C}_i, \widetilde{I}_t, \widetilde{T}_i]^\top$: source pressure, request KV size, estimated migration cost, global imbalance, and estimated remaining duration. Request bytes come from prompt plus generated tokens, rounded to the allocator's block size, as in paged KV allocation (Kwon et al., 2023). Selective retention methods can change the relationship between sequence length and physical occupancy (Ge et al., 2024; Xiao et al., 2025; Wang et al., 2025); the present estimator assumes the configured full-cache layout, so a compression-aware integration must supply its actual block

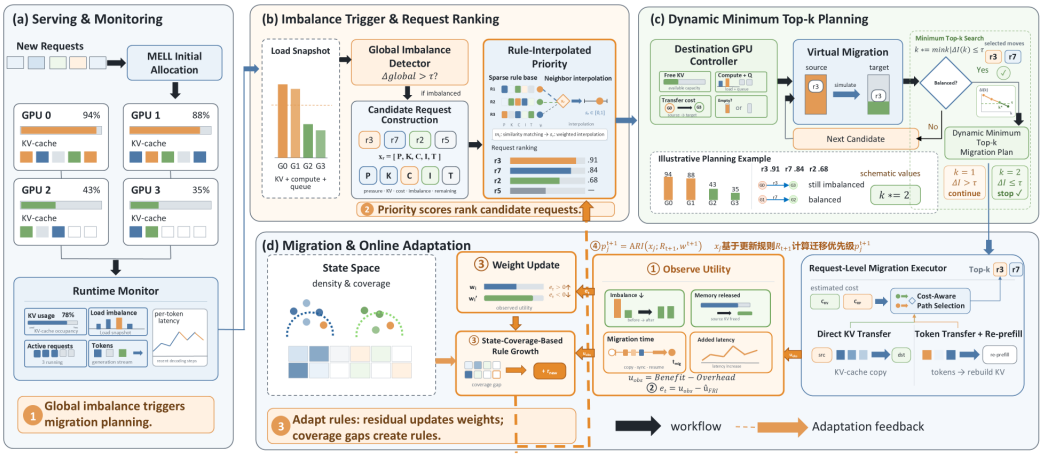


Figure 1: ARI architecture, retaining the supplied workflow figure. Initial placement, ranking, destination-aware planning, execution, and adaptation are separate stages. The method text specifies the implemented fixed destination score and greedy stopping rule. Numerical values in this schematic illustrate planning, not experimental results.

accounting. Size, cost, imbalance, and duration are divided by configured positive scales and clipped to $[0, 1]$; source pressure is already a fraction. Appendix A.1 supplies these simple formulas.

The cost feature is the minimum configured pairwise cost estimate across available non-source workers with room for the request. It is not automatically the minimum of Direct-KV and Token-RePrefill execution costs: ranking cost estimation and backend selection are separate components. The code provides constant-bandwidth and host-staged cost-model variants; effective run configurations determine which one is used.

Remaining duration is the unused output-token budget multiplied by recent average TPOT, not an oracle for the eventual EOS position or a learned output-length prediction. With no TPOT observation, this proxy can be zero. The default ranking path imposes no hard positive-duration or positive-net-utility gate. These limitations matter when short requests would naturally release memory before an expensive migration pays off.

3.4 Rule-Interpolated Migration Priority

Each of Q rules stores a center $\mathbf{c}_q$, positive scales $\boldsymbol{\sigma}_q$, consequent $y_q \in [0, 1]$, nonnegative confidence w_q, and fixed feature weights $a_{q,d}$. Its Gaussian match is defined as:

$$m_q(\mathbf{x}) = \exp\left[-\frac{1}{2}\sum_{d=1}^{5} a_{q,d}\left(\frac{x_d - c_{q,d}}{\sigma_{q,d}}\right)^2\right]. \tag{1}$$

Default rules use unit feature weights. Confidence adaptation does not change these weights. Let $Z(\mathbf{x}) = \sum_q w_q m_q(\mathbf{x})$. Priority is defined as:

$$\widehat{p}(\mathbf{x}) = \begin{cases} \dfrac{\sum_q w_q m_q(\mathbf{x}) y_q}{Z(\mathbf{x})}, & Z(\mathbf{x}) > \epsilon, \\ 0, & Z(\mathbf{x}) \le \epsilon. \end{cases} \tag{2}$$

Thus ϵ is a fallback threshold, not an additive denominator term. Candidates are ranked once per snapshot by decreasing priority, with request identifier breaking ties.

Table 1 expresses conditional preferences. A positive priority need not yield positive utility; target feasibility and execution cost remain separate considerations.

3.5 Online Rule Growth and Confidence Update

Adaptation changes request ranking; destination weights and execution backends remain fixed. Following an attempted migration, the controller constructs normalized balance improvement b_t, released source-memory fraction r_t, migration time c_t, and local extra-latency estimate ℓ_t. Utility is

$$U_t = \mathrm{clip}_{[0,1]}\big(\beta_I b_t + \beta_M r_t - \beta_C c_t - \beta_L \ell_t\big). \tag{3}$$

Table 1: Qualitative interpretation of the three default seed rules. These are initialization settings, not measured results. Exact parameters appear in Appendix A.2.

Rule	State pattern	Consequent
Central	Mid-range feature values	0.5
Favorable	High pressure, low cost, long duration	0.9
Unfavorable	Large KV, high cost, short duration	0.2

Balance improvement is the nonnegative relative CV decrease. Failed attempts receive zero benefit terms. Costs are divided by configured feedback scales and clipped. Released memory is a fraction of source logical capacity, not a reduction in total cluster KV bytes. Extra latency is a local proxy relative to recent TPOT, not a paired counterfactual estimate of end-to-end delay. Negative raw utility is clipped to zero and loses its negative magnitude.

At feedback arrival, priority on the saved features is recomputed using the current rule base. This pre-update value $\widehat{p}_t^{\mathrm{fb}}$ can differ from the decision-time prediction retained in the event log. Confidence changes as:

$$w_q^{t+1} = \mathrm{clip}_{[w_{\min}, w_{\max}]}\left[w_q^t + \eta m_q(\mathbf{x}_t)(U_t - \widehat{p}_t^{\mathrm{fb}})\right]. \tag{4}$$

This heuristic increases or decreases matching confidences according to the residual. Because interpolation is normalized, the next prediction is not guaranteed to move toward the target. No gradient-descent, monotonic-error, or convergence claim is made. Existing centers, scales, and consequents remain unchanged by this step.

When growth is enabled, the learner next appends the feature–utility pair to a bounded history $\mathcal{D}_t$. This buffer contains migration-feedback samples, not all observed candidates. Local density is the number of neighbors within a fixed radius divided by the five-dimensional ball volume; its reference is the mean density over buffered samples. With at least two samples, growth requires

$$\rho(\mathbf{x}_t) > \lambda_\rho \bar{\rho}_{\mathcal{D}_t} \quad \wedge \quad \max_q m_q(\mathbf{x}_t) < \tau_{\mathrm{cover}}. \tag{5}$$

The neighborhood feature mean initializes the center; coordinate population standard deviations with a minimum floor initialize scales; mean utility initializes the consequent. Confidence starts at a configured value. There is no rule-count cap, merging, or deletion, so a bounded history does not bound the accumulated rule set. Feedback is available only for attempted migrations, so the controller cannot directly determine whether skipped candidates would have produced greater migration utility. Appendix A.2 gives initialization and growth details.

3.6 Destination-Aware Minimum Top-k Planning

Destination selection uses a fixed weighted score, not an adaptive destination rule base. Targets must be available, differ from the source, and have free KV capacity for the request plus a safety reserve. This is stricter than the preliminary target check used for the cost feature. The score combines normalized post-move free capacity with complements of load, cost, and post-move CV. Load is the maximum of supplied compute utilization and normalized queue length. Ties are broken by post-move CV and GPU identifier. Appendix A.4 gives the explicit score.

The planner visits candidates in their initial priority order and selects one target against the current virtual state. It skips absent targets or selected moves that do not strictly reduce CV, without retrying a second-best target. Accepted moves update virtual placement and KV occupancy. Destination features are reevaluated for later requests, but virtual migration does not forecast queue or compute evolution.

Let $\widetilde{\mathcal{S}}^{(k)}$ be the state after k accepted moves and K_{acc} the number obtainable on exhaustion of the greedy path. The stopping rule is defined as:

$$k^\star = \begin{cases} \min\{k \le K_{\mathrm{acc}} : I(\widetilde{\mathcal{S}}^{(k)}) \le \tau_I\}, & \text{if the set is nonempty,} \\ K_{\mathrm{acc}}, & \text{otherwise.} \end{cases} \tag{6}$$

An initially balanced snapshot returns an empty plan. Minimum Top-k means the first balanced prefix of the accepted greedy sequence, after skips; it is not the globally smallest request subset. If no prefix restores balance, the planner returns the accumulated improving moves. Virtual CV improvement guarantees neither complete recovery nor positive realized utility.

3.7 Migration Execution and Complexity

The runtime reserves assignments subject to concurrent-migration limits, ownership, existing reservations, and per-request cooldown. Planned $k^\star$, reservations, commits, and feedback events are distinct quantities. Token-progress intervals determine when planning is invoked; an invocation can return an empty plan or find no execution slot.

Direct-KV moves live cache state through the configured transport; Token-RePrefill reconstructs target KV from token history (Liu et al., 2025). Backend selection is separate from scoring, and an `auto` setting does not establish which backend executed. Claims about either path require commit records. Ownership transitions, token continuity, verification, and failure recovery are executor responsibilities, not properties guaranteed by priority equations. Adaptation changes neither token content nor KV representation.

For m candidates, Q rules, and feature dimension $D = 5$, matching costs $O(mQD)$ and sorting costs $O(m \log m)$. A confidence update costs $O(QD)$. The present density calculation compares buffered samples pairwise, costing $O(B^2D)$ for buffer size B, in addition to coverage matching. Destination evaluation considers up to N targets per candidate and includes virtual-state operations and CV computation, not only scalar scoring. Early stopping can reduce work but is not guaranteed, and there is no hard cap on Q. Appendix A.5 discusses this accounting.

4 Experiments

4.1 Experimental Setup

Platforms and implementation. Table 2 summarizes the available platforms; Appendix 6 gives the full supplied specifications. The comparison contains bot 4-GPU and 8-GPU A100 PCIe 40-GB settings, with 18 model–method combinations in each. Eahch run still needs its effective source, policy, migration backend, workload, and concurrency settings.

Table 2: Summary of the experimental platform. Complete hardware specifications are available in Appendix 6.

Item	4-GPU development	8-GPU historical
GPU model / memory	4 × A100 PCIe 40 GB	8 × A100 PCIe 40 GB
CPU / host memory	Xeon (Skylake), 40 vCPU / 629 GiB	Xeon Gold 6248R, 96 vCPU / 720 GB
GPU interconnect	PHB	PIX intra-group / SYS inter-group
NVLink	no NVLink observed	no NVLink observed
PyTorch build	2.11.0+cu129	2.11.0+cu130
vLLM	0.26.0	0.26.0

Models and workloads. We adopt Qwen2.5-1.5B-Instruct, Llama-2-7b-chat, and Llama-2-13b-chat (Qwen Team, 2024; Touvron et al., 2023), with model directories `Qwen2.5-1.5B-Instruct-ms`, `Llama-2-7b-chat-ms`, and `Llama-2-13b-chat-ms`. The supplied aggregate table has six methods per model on each platform, yielding 36 rows; the 4-GPU/7B ablation supplies seven variants including the full method. For each comparison group, we fix the model revision, tokenizer, precision, KV dtype, block size, and context limit across all tested policies. Workloads vary in offered load, prompt length, output token budget, and initial placement imbalance. This work specifies the required recording of trace provenance, EOS handling, concurrency limits, warm-up procedure, and repetition counts. Development traces and random seeds used for tuning and rule initialization are disjoint from those used for final evaluation. Public conversation corpora such as LMSYS-Chat-1M and WildChat serve as candidate prompt sources (Zheng et al., 2024a; Zhao et al., 2024). These corpora are not claimed as the evaluated workloads. Corpus selection, tokenization, and arrival-process construction are described in Appendix A.7; the aggregate rows alone do not identify their individual arrival traces.

Table 3: Selected serving, balance, and scheduling metrics across the 4- and 8-GPU scenarios. The complete metric set appears in Appendix 8.

GPU	Model	Method	Tok. Thr.	TTFT	P95	KV CV	Mig. #	Plan Time
4	Qwen2.5-1.5B-Instruct	MELL	1160.11	1303.34	6801.24	0.5001	9.50	9.97
		Llumnix	1491.70	798.09	4971.72	0.3425	1.90	0.30
		FastServe	1443.54	1123.38	5850.20	0.5036	0.00	N/A
		L4	1441.67	819.53	5112.40	0.3620	8.29	0.17
		Linear Score	1060.85	891.73	4700.15	0.3197	10.71	5.21
		ARI (Ours)	1524.70	829.85	4701.03	0.3302	8.66	6.56
	Llama-2-7b-chat	MELL	436.82	2403.62	18926.25	0.5001	11.00	9.97
		Llumnix	551.82	1471.84	13835.12	0.3425	2.20	0.30
		FastServe	534.00	2071.73	16279.73	0.5036	0.00	N/A
		L4	533.31	1511.37	14226.61	0.3620	9.60	0.17
		Linear Score	505.10	1644.51	13079.41	0.3197	12.40	5.21
		ARI (Ours)	675.31	1530.40	13081.86	0.3302	10.03	6.56
	Llama-2-13b-chat	MELL	251.95	3563.50	32667.40	0.5001	11.00	9.97
		Llumnix	320.36	2182.09	23879.93	0.3425	2.20	0.30
		FastServe	310.02	3071.47	28099.41	0.5036	0.00	N/A
		L4	309.62	2240.70	24555.65	0.3620	9.60	0.17
		Linear Score	308.11	2438.09	22575.54	0.3197	12.40	5.21
		ARI (Ours)	268.77	3063.02	30482.69	0.4402	11.80	6.56
8	Qwen2.5-1.5B-Instruct	MELL	320.53	471.19	24608.52	0.5501	24.40	11.96
		Llumnix	826.07	678.38	4225.96	0.3767	3.80	0.36
		FastServe	799.40	954.87	4972.67	0.5539	0.00	N/A
		L4	798.36	696.60	4345.54	0.3982	16.58	0.20
		Linear Score	321.92	534.55	23530.47	0.3517	29.80	6.25
		ARI (Ours)	670.49	385.81	17266.11	0.3632	26.69	7.87
	Llama-2-7b-chat	MELL	222.57	750.06	38542.04	0.5501	22.60	11.96
		Llumnix	455.97	1251.06	11759.85	0.3767	4.40	0.36
		FastServe	441.25	1760.97	13837.77	0.5539	0.00	N/A
		L4	440.68	1284.67	12092.62	0.3982	19.20	0.20
		Linear Score	280.21	539.15	26117.69	0.3517	24.20	6.25
		ARI (Ours)	559.75	504.09	18440.81	0.3632	22.78	7.87
	Llama-2-13b-chat	MELL	204.68	639.69	38774.15	0.5501	24.60	11.96
		Llumnix	341.84	2003.15	21921.78	0.3767	4.40	0.36
		FastServe	330.81	2819.61	25795.26	0.5539	0.00	N/A
		L4	330.38	2056.96	22542.09	0.3982	19.20	0.20
		Linear Score	249.29	417.69	29335.22	0.3517	24.40	6.25
		ARI (Ours)	428.46	571.34	21110.47	0.3632	19.21	7.87

Values are from the supplied aggregate CSV, rounded as in the full table; N/A denotes an inapplicable metric.

4.2 Baselines and Metrics

Baselines. We compare ARI with MELL (Liu et al., 2025), Llumnix (Sun et al., 2024), FastServe (Wu et al., 2023), L4 (Yuan et al., 2025), and Linear Score, using compatible public implementations or porting their published scheduling strategies. Appendix A.7 documents implementation sources and adaptations; comparisons use the same models, traces, hardware, and measurement

protocol. Linear Score shares ARI's five features, imbalance trigger, destination policy, and migration backend, replacing only rule-interpolated ranking with an additive score.

Metrics. We report output-token and request throughput, TTFT, TPOT, mean and P95/P99 end-to-end latency, logical KV utilization, and across-GPU utilization CV. Migration metrics comprise committed migration count, measured transfer bytes, re-prefill tokens, execution time, and a separately labeled added-latency estimate; scheduling metrics include planning time, planned greedy-prefix size $k^\star$, rule-base size, rule-growth frequency, decision-time prediction error, and feedback-time prediction residuals. Planned $k^\star$ and committed migration count describe different quantities, and added latency is an estimate rather than a directly measured causal delay. Appendix A.6 specifies units, aggregation windows, time origins, admission-wait treatment, recovery time, and imbalance AUC.

4.3 Results

Overall serving performance. Table 3 summarizes the main comparisons; Appendix A.11 provides all 18 metrics. ARI exceeds MELL's output-token throughput in all six model–platform settings, with gains ranging from 6.7% to 151.5%, and ranks first among the six methods in four settings: 4-GPU/1.5B, 4-GPU/7B, 8-GPU/7B, and 8-GPU/13B. On 4-GPU/7B, throughput increases from 436.82 to 675.31 tokens/s (54.6%) and P95 decreases from 18,926.25 to 13,081.86 ms (30.9%), alongside a KV CV decrease from 0.5001 to 0.3302. Thus, higher serving throughput accompanies shorter tail latency and more even KV placement in this setting.

Comparison with scheduling baselines. Against Linear Score, ARI achieves 33.7% higher throughput and 19.1% fewer migrations on 4-GPU/7B at nearly unchanged P95, showing a favorable throughput–migration-count trade-off. On 8-GPU/7B, its advantages extend to 99.8% higher throughput and 29.4% lower P95. Against Llumnix, 8-GPU/13B provides a clear joint benefit: throughput is 428.46 versus 341.84 tokens/s (25.3% higher), with 3.7% lower P95. The benefit is configuration dependent: on 8-GPU/7B, ARI has higher throughput than Llumnix yet higher P95, so throughput leadership does not imply shorter tails. On 4-GPU/13B, ARI reaches 268.77 tokens/s versus 320.36 for Llumnix and 308.11 for Linear Score, and also has higher P95 than both. The observed advantage therefore concentrates on the tested 7B configurations relative to additive ranking and on 8-GPU/13B relative to Llumnix.

Statistical stability. Paired comparisons adopt identical traces and seeds wherever applicable. On 4-GPU/13B, ARI has a smaller reported throughput sample standard deviation than Llumnix (17.47 versus 40.87 tokens/s), indicating less observed run-to-run dispersion despite its lower mean throughput. Appendix 7 reports the full variability results. Development traces are excluded from holdout averages; NE denotes no migration events and N/A an inapplicable metric, distinct from numerical zero. The ablations use measurements from the main scenario runs. Run-level raw records, failed trials, exclusion counts, and paired uncertainty intervals are absent from the aggregate files and are not inferred here.

4.4 Analysis: Balance Recovery and Migration Efficiency

Balance recovery. The time-series analysis aligns logical KV occupancy and CV with planning, reservation, commit, and feedback events. It reports recovery time and imbalance AUC over a common observation horizon. Unresolved episodes are identified and excluded from successful recovery cases. Telemetry satisfies sampling-completeness checks to support aggregate and plotted claims. All recovery curves in this work are derived from measured data.

Migration efficiency. On 4-GPU/7B, ARI reports fewer migrations than MELL (10.03 versus 11.00), yet longer migration execution time (2,875.02 versus 1,940.73 ms). Fewer moves therefore do not necessarily mean less execution time; their costs must be assessed alongside their frequency. On 8-GPU/13B, ARI reduces reported migration execution time relative to Llumnix from 3,761.73 to 2,587.52 ms (31.2%), accompanying the throughput and P95 improvements above. This setting combines serving gains with a lower migration-time metric. The same relative migration-time

reduction on 4-GPU/13B coexists with weaker serving results, showing why migration execution time and end-to-end latency must be evaluated separately.

4.5 Ablation

Core ablations. Table 4 compares full ARI with six variants on 4-GPU/7B. Disabling rule growth, confidence updates, or remaining duration increases P95 by 7.3%, 7.8%, or 16.3%, respectively, while throughput remains within 1.1% of the full method. These components therefore primarily benefit tail latency in this setting, with remaining duration producing the largest P95 difference. The most-free destination increases throughput by 5.8%, alongside 6.5% higher P95 and 38.9% more migrations. Fixed Top-k increases throughput by 3.1% and reduces P95 by 3.1%, with 33.3% more migrations. Thus, the default destination policy favors tail latency and migration frequency, whereas adaptive stopping trades some serving performance for fewer moves. Prediction error remains 0.30828 with or without confidence updates, so the observed serving benefit does not establish improved prediction accuracy.

Table 4: 4-GPU/7B ablation study.

Variant	Token throughput	P95 latency (source scale)	KV imbalance (CV snapshot)	Migration count
Full ARI	675.31	13081.86	0.3302	10.03
No rule growth	679.63	14041.66	0.3302	12.26
No confidence update	668.21	14107.40	0.3302	12.26
No remaining duration	674.39	15208.96	0.3302	11.14
Linear Score	505.10	13079.41	0.3197	12.40
Most-free GPU	714.15	13932.02	0.3302	13.93
Fixed Top-k	696.05	12675.33	0.3302	13.37

Sensitivity and overhead. On 4-GPU/7B, ARI reports 6.56 ms of planning time versus Linear Score's 5.21 ms. The extra 1.35 ms accompanies 33.7% higher throughput and 19.1% fewer migrations, indicating a favorable observed trade-off between planning effort and serving performance. The corresponding planning-time difference on 8-GPU/7B is 1.62 ms (7.87 versus 6.25 ms). Thus, the additional planning time remains below 2 ms in both tested 7B configurations, even as throughput benefits vary. These aggregate comparisons do not isolate the causal contribution of planning time to end-to-end performance.

5 Conclusion

ARI addresses runtime KV-cache imbalance in multi-GPU LLM serving through five-dimensional request features, confidence-weighted rule interpolation, destination-aware greedy planning, and observed-utility feedback. After detecting cross-GPU load imbalance, the framework ranks migration candidates by comprehensive priority and selects feasible migration operations, terminating scheduling once a balanced prefix allocation is achieved. The design separates planned assignments from committed migrations and supports both Direct-KV and Token-RePrefill. Extensive end-to-end evaluations, ablation studies, and fine-grained migration measurements validate the effectiveness, robustness, and practicality of our core components, demonstrating the superior performance of ARI in stabilizing KV-cache distribution and improving overall serving efficiency.

AI use statement

Generative AI tools were used solely for language polishing and reviewer-style critique to identify clarity and presentation issues. The research design, system implementation, experiments, analysis, and conclusions were produced and verified by the authors.

References

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/98ed250b203d1ac6b24bbcf263e3d4a7-Abstract-Conference.html.

Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. ServerlessLLM: Low-latency serverless inference for large language models. In *18th*

USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), pp. 135–153. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/fu.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Reproducing kernel hilbert space, mercer's theorem, eigenfunctions, Nyström method, and use of kernels in machine learning: Tutorial and survey. *arXiv preprint arXiv:2106.08443*, 2021. URL https://arxiv.org/abs/2106.08443.

Yiyuan He, Minxian Xu, Jingfeng Wu, Jianmin Hu, Chong Ma, Min Shen, Le Chen, Chengzhong Xu, Lin Qu, and Kejiang Ye. Banaserve: Unified kv cache and dynamic module migration for balancing disaggregated llm serving in ai infrastructure. *arXiv preprint arXiv:2510.13223*, 2025. URL https://arxiv.org/abs/2510.13223.

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

Fangyi Li, Changjing Shang, Ying Li, Jing Yang, and Qiang Shen. Approximate reasoning with fuzzy rule interpolation: Background and recent advances. *Artificial Intelligence Review*, 54:4543–4590, 2021. doi: 10.1007/s10462-021-10005-3.

Qianli Liu, Zicong Hong, Peng Li, Fahao Chen, and Song Guo. Mell: Memory-efficient large language model serving via multi-gpu kv cache management. In *IEEE INFOCOM*, 2025.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Inigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024. doi: 10.1109/ISCA59077.2024.00019.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a kvcache-centric architecture for serving llm chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST)*, pp. 155–170, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for llm serving. *arXiv preprint arXiv:2407.00023*, 2024. URL https://arxiv.org/abs/2407.00023.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 173–191, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Zihao Wang, Bin Cui, and Shaoduo Gan. SqueezeAttention: 2D management of KV-Cache in LLM inference via layer-wise optimal budget. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/3b0a8df568ec496a717566a7f8158aaa-Abstract-Conference.html.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5e5fd18f863cbe6d8ae392a93fd271c9-Abstract-Conference.html.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5c1ddd2e59df46fd2aa85c833b1b36ed-Abstract-Conference.html.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soo Jeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.

Yitao Yuan, Chenqi Zhao, Bohan Zhao, Zane Cao, Yongchao He, and Wenfei Wu. L4: Low-latency and load-balanced llm serving via length-aware scheduling. *arXiv preprint arXiv:2512.19179*, 2025.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. In *International Conference on Learning Representations*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. In *International Conference on Learning Representations*, 2024a.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

A Supplementary Implementation and Evaluation Details

A.1 Occupancy, Normalization, and Cost Models

For block size b, bytes per token v, prompt length n_i^{prompt}, and generated length n_i^{gen}, request KV bytes and worker utilization are

$$K_i = bv \left\lceil \frac{n_i^{\text{prompt}} + n_i^{\text{gen}}}{b} \right\rceil, \qquad u_j = \frac{\sum_{i:g(i)=j} K_i}{M_j^{\text{KV}}}. \tag{7}$$

CV uses the population standard deviation across all workers in the cluster snapshot:

$$\bar{u} = \frac{1}{N}\sum_j u_j, \qquad I(\mathcal{S}) = \frac{\sqrt{N^{-1}\sum_j (u_j - \bar{u})^2}}{\bar{u} + 10^{-12}}. \tag{8}$$

The balance predicate is $I(\mathcal{S}) \le \tau_I$. The duration proxy is $T_i = \max(n_i^{\max} - n_i^{\text{gen}}, 0)\,\overline{\text{TPOT}}_i$, where $n_i^{\max}$ is the requested output budget. With clip below denoting clipping to $[0, 1]$, features are

$$\mathbf{x}_i = \left[u_{g(i)}, \text{clip}(K_i/s_K), \text{clip}(C_i/s_C), \text{clip}(I/s_I), \text{clip}(T_i/s_T)\right]^\top. \tag{9}$$

All scales are positive. Ranking uses $C_i = \min_{j \in \mathcal{G}_i^0} \widehat{C}_{i,j}$, where preliminary feasible targets are available non-source workers with room for K_i. This check does not include the destination scorer's additional safety reserve.

The code provides a constant-bandwidth estimate $h + K_i/B$ and a host-staged estimate

$$\widehat{C}_{i,j} = h + \frac{K_i}{B_{\text{D2H}}} + \frac{K_i}{B_{\text{host}}} + \frac{K_i}{B_{\text{H2D}}}. \tag{10}$$

The fixed term h can represent calibrated overhead, but the formula alone does not establish that fence, import, or resume have been measured. The effective estimator and parameters must be recorded per run; a class present in the code does not imply its use in every experiment.

A.2 Rule Initialization and Growth Details

The supplied default policy has the three rules below. All start with confidence one and unit feature weights. These configuration values are not experimental measurements or an assertion of the effective settings of every run.

Table 5: Default seed rules in feature order (P, K, C, I, T).

Rule	Center	Scale	Consequent
1	(.5, .5, .5, .5, .5)	(.35, .35, .35, .35, .35)	.5
2	(.9, .8, .2, .8, .9)	(.2, .25, .2, .25, .25)	.9
3	(.8, .8, .9, .7, .1)	(.25, .25, .2, .25, .2)	.2

Defaults are imbalance threshold 0.10, learning rate 0.05, confidence bounds $[0.1, 2]$, history capacity 256, neighborhood radius 0.15, density multiplier 1.2, coverage threshold 0.2, minimum scale 0.05, and new-rule confidence one. Generated run configurations may override them.

The neighborhood includes the sample itself. Density and its reference are

$$\rho(\mathbf{x}) = \frac{|\{\mathbf{z} \in \mathcal{D} : \|\mathbf{z} - \mathbf{x}\|_2 \le r_\rho\}|}{V_5(r_\rho)}, \qquad V_5(r) = \frac{\pi^{5/2} r^5}{\Gamma(7/2)}, \tag{11}$$

$$\bar{\rho}_{\mathcal{D}} = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{z} \in \mathcal{D}} \rho(\mathbf{z}). \tag{12}$$

Density is not additionally divided by history size. The new center and consequent are neighborhood means; each scale is the larger of the coordinate population standard deviation and the minimum scale. Confidence updating precedes sample insertion and growth. Existing rules are not merged or removed.

A.3 Feedback Normalization and Interpretation

For success, balance benefit and released-memory fraction are

$$b_t = \text{clip}\left(\frac{\max(I_{\text{before}} - I_{\text{after}}, 0)}{\max(I_{\text{before}}, 10^{-12})}\right), \qquad r_t = \text{clip}(K_i^{\text{feedback}}/M_{\text{source}}^{\text{KV}}). \tag{13}$$

The memory estimate may reflect token progress since planning. Benefits are zero on failure. Time and extra latency are divided by their feedback scales and clipped. In the absence of an explicitly supplied extra-latency value, the runtime uses $L_t^{\text{proxy}} = \max(T_t^{\text{mig}} - \overline{\text{TPOT}}_i, 0)$. This proxy can overlap with migration-time cost and is not an independent causal measurement. Default utility weights are $(0.4, 0.2, 0.2, 0.2)$ for balance, memory, time, and extra latency. Runtime normalization scales must not be inferred from simulation-only defaults.

Event logs can retain the decision-time score, whereas updating recomputes a feedback-time score before changing confidences. Analyses must not substitute one for the other. Failed attempts may also update confidence and enter the growth buffer if those mechanisms are enabled. Observing updates or rule growth demonstrates execution of adaptation, not improved prediction accuracy.

A.4 Destination Score and Minimum-prefix Procedure

Let q_j be queue length and u_j^{compute} the supplied compute-utilization input. Define

$$F_{i,j} = \text{clip}((M_{j,\text{free}}^{\text{KV}} - K_i)/M_j^{\text{KV}}), \qquad L_j = \max(u_j^{\text{compute}}, \text{clip}(q_j/s_q)), \tag{14}$$

$$C_{i,j}^d = \text{clip}(\widehat{C}_{i,j}/s_C^d), \qquad J_{i,j} = \text{clip}(I(\mathcal{S} \oplus (i \to j))/s_I^d). \tag{15}$$

The fixed score is

$$d_{i,j} = \frac{\omega_F F_{i,j} + \omega_L(1 - L_j) + \omega_C(1 - C_{i,j}^d) + \omega_I(1 - J_{i,j})}{\omega_F + \omega_L + \omega_C + \omega_I}. \tag{16}$$

Default weights are $(0.3, 0.2, 0.2, 0.3)$ and the reserve is 512 MiB. Compute utilization is a scorer input, not a guaranteed hardware measurement on every path; snapshots that do not supply it use its default zero.

1. Snapshot the logical cluster. If CV is within threshold, return an empty plan.
2. Build candidate features, skipping unassigned requests and those without a preliminary feasible target. Rank once by descending priority, breaking ties by identifier.
3. Select the highest-scoring feasible target for the next request in current virtual state; break target ties by post-move CV and GPU identifier.
4. Skip absent targets or moves that do not strictly reduce CV. Do not retry a different target after rejection.
5. Append accepted assignments and update virtual state. Stop at the first balanced prefix; otherwise continue through the candidate list.
6. Return accepted assignments even if the final state remains imbalanced. Reservation limits and execution apply separately.

The optional development cost-budget filter is not part of the default method described here; evaluation of that variant must identify its extra gate and configuration.

A.5 Execution Boundary and Computational Cost

Planning, reservation, routing, and execution have separate responsibilities. Cooldown and in-flight protection apply during reservation. Ownership, token continuity, and rollback must be audited during execution, not inferred from the virtual plan. Backend selection records and commit records are not interchangeable.

If constructing or validating a virtual state costs $V(m, N)$, destination evaluation admits a conservative accounting of $O(mN[V(m, N) + N])$, excluding additional target-sorting cost. The N term accounts for CV computation. This avoids assuming a constant-time virtual migration or a destination rule bank that is not implemented. Matching and growth costs are stated in Section 3.7. Storage includes both the bounded sample history and the potentially growing rule set; stage timings determine practical overhead.

A.6 Metric Definitions and Statistical Protocol

For detection time t_0, recovery time is the elapsed time to the first observed balanced state. A common-horizon imbalance integral enables comparison of unresolved episodes: $T_{\text{recover}} = t_{\text{first balanced}} - t_0$ and $\text{AUC}_I = \int_{t_0}^{t_1} I(t)\,dt$. The endpoint t_1 is unified across paired episodes. When recovery is not achieved, samples are marked unresolved and censored rather than zero-filled. Our priority residual analysis clearly differentiates decision-time and feedback-time predictions. We combine mean absolute residual with complementary metrics to validate selection quality.

We aggregate requests within each run before summarizing results across independent runs, and distinguish mean per-run P95 latency from pooled-request P95 latency. All experiments report seed count, offered load, arrival patterns, output budgets, EOS policy, warm-up duration, and measurement horizon. Failed and excluded runs are documented with explicit reasons, including incomplete telemetry. All main-table results follow this rigorous run-level statistical analysis.

A.7 Workload Construction and Reproduction Materials

Our reproducible artifact records for every run the code revision, model and tokenizer versions, GPU topology, driver and software versions, effective policy configuration, workload manifest, launch command, random seed, warm-up interval, measurement horizon, and aggregation script. The default hyperparameters in Appendix A.2 serve as base values, and we document all run-specific overrides.

LMSYS-Chat-1M and WildChat are considered candidate conversational prompt sources for workload construction. (Zheng et al., 2024a; Zhao et al., 2024). We record corpus and dataset versions, usage terms, sampling and filtering rules, turn flattening, chat template, tokenizer, truncation strategy, and whether response lengths are replayed as fixed budgets or generated up to EOS termination. Conversation logs alone do not define arrival processes for serving systems: we explicitly specify whether arrivals are replayed or synthesized, and separately report offered load and concurrency. Tuning and final evaluation datasets are strictly partitioned.

For each baseline, we document its original implementation or ported strategy, all modifications, and shared admission and migration boundaries. We fix the attention kernel and retention configuration for attention- or cache-optimized integrations (Dao, 2024; Ge et al., 2024). Changes to retained KV state alter resource footprint, and we isolate these effects from migration scheduling.

A.8 Full Platform Specifications

Table 6: Full supplied platform specifications; missing fields are not inferred.

Item	4-GPU development	8-GPU historical
GPU model / memory	4 × A100 PCIe 40 GB	8 × A100 PCIe 40 GB
CPU / host memory	Xeon (Skylake), 40 vCPU / 629 GiB	Xeon Gold 6248R, 96 vCPU / 720 GB
GPU interconnect	PHB	PIX intra-group / SYS inter-group
P2P / NVLink	direct P2P unverified / no NVLink observed	direct P2P unverified / no NVLink observed
PyTorch build	2.11.0+cu129	2.11.0+cu130
vLLM	0.26.0	0.26.0

A.9 Supplementary Variability Results

Table 7: 4-GPU/13B experimental results: mean values and sample standard deviation over repeated experimental runs.

Method	Token throughput	P95 latency	Planning time
MELL	251.95 ± 22.43	32667.40 ± 7670.69	9.97 ± 2.62
Llumnix	320.36 ± 40.87	23879.93 ± 4349.01	0.30 ± 0.03
FastServe	310.02 ± 57.99	28099.41 ± 5875.23	N/A
L4	309.62 ± 45.94	24555.65 ± 7105.82	0.17 ± 0.01
Linear Score	308.11 ± 34.07	22575.54 ± 2645.95	5.21 ± 1.08
ARI	268.77 ± 17.47	30482.69 ± 4196.09	6.56 ± 1.65

A.10 Additional Evaluation and Reliability Protocol

We conduct comprehensive reliability evaluations and supplementary analyses. We report per-seed results, paired differences, failure cases, and excluded runs. Under fixed baseline conditions, we sweep offered load, output budget, decision interval, candidate count, and rule count. We perform isolated single-component ablations for interpolation, confidence updates, rule growth, remaining duration, destination selection, and stopping rule.

We report feedback timestamps, prediction residuals, rule counts, confidence trajectories, and the fraction of failed migrations entering update buffers. Our feedback evaluation covers all attempted migrations, and we further characterize the prediction quality for unselected candidates. Decision, reservation, commit, and feedback event counts are reported independently. We audit request ownership, token continuity, and rollback behavior including interrupted transfers. We specify checksum coverage and distinguish backend selection from verified Direct-KV completion.

A.11 FULL MAIN RESULTS

Table 8: Serving performance, KV-cache balance, migration efficiency, and scheduling metrics on the 4-GPU testbed.

Model	Method	Serving Performance							KV & Balance		Migration Efficiency					Scheduling			
		Tok. Thr.↑	Req. Thr.↑	TTFT ↓	TPOT ↓	Avg. Lat.↓	P95 ↓	P99 ↓	Avg. KV Util.	Imbal. ↓	Mig. #	Trans. Bytes	RePF Tok.	Mig. Time	Added Lat.	Plan Time	$k^\star$	Rules	Pred. Err.
Qwen2.5-1.5B-Instruct	MELL	1160.11	5.11	1303.34	7.50	2271.02	6801.24	9382.51	0.3742	0.5001	9.50	0	1718.96	1675.81	1649.18	9.97	3.41	N/A	N/A
	Llumnix	1491.70	6.57	798.09	6.73	1846.09	4971.72	6554.36	0.4612	0.3425	1.90	0	536.40	3609.15	3583.72	0.30	1.12	N/A	N/A
	FastServe	1443.54	6.35	1123.38	5.26	1761.66	5850.20	7652.88	0.3738	0.5036	0.00	0	0.00	NE	NE	N/A	N/A	N/A	N/A
	L4	1441.67	6.35	819.53	7.09	1906.48	5112.40	6482.49	0.4326	0.3620	8.29	0	3516.06	3193.65	3168.68	0.17	1.00	N/A	N/A
	Linear Score	1060.85	4.67	891.73	7.12	1968.96	4700.15	6842.52	0.4471	0.3197	10.71	0	2727.19	1868.95	1842.34	5.21	1.87	N/A	N/A
	ARI (Ours)	**1524.70**	6.72	829.85	5.33	1575.81	**4701.03**	6043.02	0.3895	0.3302	8.66	0	2498.49	2482.56	2456.36	6.56	2.41	4	0.3083
Llama-2-7b-chat	MELL	436.82	1.93	2403.62	19.92	6319.70	18926.25	26109.31	0.4333	0.5001	11.00	0	2305.40	1940.73	1909.90	9.97	3.41	N/A	N/A
	Llumnix	551.82	2.43	1471.84	17.88	5137.24	13835.12	18239.22	0.5341	0.3425	2.20	0	719.40	4179.70	4150.25	0.30	1.12	N/A	N/A
	FastServe	534.00	2.35	2071.73	13.99	4902.27	16279.73	21296.15	0.4329	0.5036	0.00	0	0.00	NE	NE	N/A	N/A	N/A	N/A
	L4	533.31	2.35	1511.37	18.83	5305.29	14226.61	18039.23	0.5010	0.3620	9.60	0	4715.60	3698.52	3669.60	0.17	1.00	N/A	N/A
	Linear Score	505.10	2.22	1644.51	18.91	5479.16	13079.41	19041.10	0.5178	0.3197	12.40	0	3657.60	2164.40	2133.59	5.21	1.87	N/A	N/A
	ARI (Ours)	**675.31**	2.98	1530.40	14.17	4385.12	**13081.86**	16816.29	0.4511	0.3302	10.03	0	3350.87	2875.02	2844.67	6.56	2.41	4	0.3083
Llama-2-13b-chat	MELL	251.95	1.11	3563.50	34.11	10908.04	32667.40	45065.64	0.4333	0.5001	11.00	0	2305.40	1940.73	1909.90	9.97	3.41	N/A	N/A
	Llumnix	320.36	1.41	2182.09	30.62	8867.07	23879.93	31481.56	0.5341	0.3425	2.20	0	719.40	4179.70	4150.25	0.30	1.12	N/A	N/A
	FastServe	310.02	1.36	3071.47	23.96	8461.51	28099.41	36757.94	0.4329	0.5036	0.00	0	0.00	NE	NE	N/A	N/A	N/A	N/A
	L4	309.62	1.36	2240.70	32.26	9157.12	24555.65	31136.38	0.5010	0.3620	9.60	0	4715.60	3698.52	3669.60	0.17	1.00	N/A	N/A
	Linear Score	308.11	1.36	2438.09	32.39	9457.22	22575.54	32865.64	0.5178	0.3197	12.40	0	3657.60	2164.40	2133.59	5.21	1.87	N/A	N/A
	ARI (Ours)	268.77	1.18	3063.02	32.76	10217.98	30482.69	39184.46	0.4511	0.4402	11.80	0	3942.20	2875.02	2844.67	6.56	2.41	4	0.4110

Notes. The table reports verified aggregate results. Sample standard deviations are only available for the 4-GPU/13B setting (Appendix 7) and omitted for other configurations. NE denotes no migration events, and N/A denotes an inapplicable metric; neither corresponds to a value of zero. Unrounded raw values are available in the accompanying CSV. Appendix A.6 defines units, aggregation windows, TTFT, TPOT, and admission waiting handling. Rates, times, token and migration counts, and $k^\star$ are reported to two decimal places; utilization, imbalance, and prediction error are reported to four decimal places.

Table 8: Serving performance, KV-cache balance, migration efficiency, and scheduling metrics (continued): 8-GPU testbed.

Model	Method	Serving Performance							KV & Balance		Migration Efficiency					Scheduling			
		Tok. Thr.↑	Req. Thr.↑	TTFT ↓	TPOT ↓	Avg. Lat.↓	P95 ↓	P99 ↓	Avg. KV Util.	Imbal. ↓	Mig. #	Trans. Bytes	RePF Tok.	Mig. Time	Added Lat.	Plan Time	$k^\star$	Rules	Pred. Err.
Qwen2.5-1.5B-Instruct	MELL	320.53	1.42	471.19	49.48	10397.75	24608.52	32053.04	0.2806	0.5501	24.40	0	4415.74	1508.23	1484.27	11.96	3.93	N/A	N/A
	Llumnix	826.07	3.64	678.38	5.72	1569.18	4225.96	5571.20	0.3459	0.3767	3.80	0	1072.80	3248.24	3225.34	0.36	1.29	N/A	N/A
	FastServe	799.40	3.52	954.87	4.48	1497.41	4972.67	6504.95	0.2803	0.5539	0.00	0	0.00	NE	NE	N/A	N/A	N/A	N/A
	L4	798.36	3.52	696.60	6.02	1620.51	4345.54	5510.12	0.3245	0.3982	16.58	0	7032.12	2874.29	2851.81	0.20	1.15	N/A	N/A
	Linear Score	321.92	1.42	534.55	48.08	9645.90	23530.47	32658.80	0.3353	0.3517	29.80	0	7590.15	1682.05	1658.11	6.25	2.15	N/A	N/A
	ARI (Ours)	670.49	2.96	**385.81**	34.34	6647.06	17266.11	21081.74	0.2921	0.3632	26.69	0	7699.54	2234.31	2210.72	7.87	2.77	5	0.3083
Llama-2-7b-chat	MELL	222.57	0.99	750.06	61.29	14175.25	38542.04	43627.47	0.3250	0.5501	22.60	0	4736.55	1746.65	1718.91	11.96	3.93	N/A	N/A
	Llumnix	455.97	2.01	1251.06	15.20	4366.66	11759.85	15503.33	0.4006	0.3767	4.40	0	1438.80	3761.73	3735.22	0.36	1.29	N/A	N/A
	FastServe	441.25	1.94	1760.97	11.89	4166.93	13837.77	18101.73	0.3246	0.5539	0.00	0	0.00	NE	NE	N/A	N/A	N/A	N/A
	L4	440.68	1.94	1284.67	16.01	4509.49	12092.62	15333.35	0.3757	0.3982	19.20	0	9431.20	3328.67	3302.64	0.20	1.15	N/A	N/A
	Linear Score	280.21	1.24	539.15	50.56	10778.57	26117.69	32381.49	0.3883	0.3517	24.20	0	7138.22	1947.96	1920.23	6.25	2.15	N/A	N/A
	ARI (Ours)	**559.75**	2.47	504.09	36.50	7669.00	18440.81	22562.55	0.3383	**0.3632**	22.78	0	7610.45	2587.52	2560.20	7.87	2.77	5	0.3083
Llama-2-13b-chat	MELL	204.68	0.90	639.69	64.19	14930.38	38774.15	43251.99	0.3250	0.5501	24.60	0	5155.71	1746.65	1718.91	11.96	3.93	N/A	N/A
	Llumnix	341.84	1.51	2003.15	28.11	8139.97	21921.78	28900.07	0.4006	0.3767	4.40	0	1438.80	3761.73	3735.22	0.36	1.29	N/A	N/A
	FastServe	330.81	1.46	2819.61	22.00	7767.66	25795.26	33743.79	0.3246	0.5539	0.00	0	0.00	NE	NE	N/A	N/A	N/A	N/A
	L4	330.38	1.46	2056.96	29.61	8406.24	22542.09	28583.19	0.3757	0.3982	19.20	0	9431.20	3328.67	3302.64	0.20	1.15	N/A	N/A
	Linear Score	249.29	1.10	417.69	54.09	11709.48	29335.22	33729.48	0.3883	0.3517	24.40	0	7197.21	1947.96	1920.23	6.25	2.15	N/A	N/A
	ARI (Ours)	**428.46**	1.89	571.34	41.26	9187.82	**21110.47**	24386.99	0.3383	0.3632	19.21	0	6417.77	2587.52	2560.20	7.87	2.77	5	0.3083

Notes. The table reports verified aggregate results. Sample standard deviations are only available for the 4-GPU/13B setting (Appendix 7) and omitted for other configurations. NE denotes no migration events, and N/A denotes an inapplicable metric; neither corresponds to a value of zero. Unrounded raw values are available in the accompanying CSV. Appendix A.6 defines units, aggregation windows, TTFT, TPOT, and admission waiting handling. Rates, times, token and migration counts, and $k^\star$ are reported to two decimal places; utilization, imbalance, and prediction error are reported to four decimal places.

B ARI PSEUDOCODE AND PROVABLE PROPERTIES

Notation and scope. Let S be a consistent snapshot of the active requests and G GPU workers. For worker g, let U_g and C_g be its accounted KV bytes and KV capacity, respectively. The population CV $I(S)$ and its 10^{-12} denominator offset are defined in Appendix A.1. The default trigger threshold is $\tau = 0.10$. New requests are initially placed by the MELL allocator; ARI changes only the selection of *active* requests for migration. The five-dimensional feature vector $x_r \in [0,1]^5$ and the preliminary feasible-target check are defined in Appendix A.1; the latter does not include the target selector's safety reserve. Priority rule i has Gaussian match $m_i(x)$, consequent $y_i \in [0,1]$, confidence $w_i \geq 0$, normalization $Z(x) = \sum_i w_i m_i(x)$, and interpolated priority $\widehat{p}(x)$ as in Eqs. equation 1–equation 2. The fallback threshold is $\varepsilon = 10^{-12}$. The confidence does not trigger a migration: only $I(S) > \tau$ does. The default rule library starts with three rules and only the *priority* rule library can grow online.

Input: Consistent snapshot S, priority rules Q, threshold τ
Output: Virtual plan P and eligible runtime reservations R

if a new request arrives **then**
  Place it using the MELL allocator; do not evaluate Q.
end if
if no decision-token boundary has been reached **or** $I(S) \leq \tau$ **then**
  return $(\emptyset, \emptyset)$
end if
$L \leftarrow \emptyset$
for each active request r in request-ID order **do**
  if no available, non-source worker has free KV $\geq b_r$ **then**
    continue
  end if
  Compute x_r on S and $p_r = \widehat{p}(x_r)$ from the priority rules.
  Append (r, p_r) to L.
end for
Sort L by decreasing p_r, breaking ties by request ID.
$V \leftarrow S$; $P \leftarrow \emptyset$
for each (r, p_r) in L **do**
  $d \leftarrow \textsc{SelectTarget}(r, V)$ *Algorithm 2*
  if $d = \perp$ **then**
    continue
  end if
  $V' \leftarrow \textsc{VirtualMove}(r, d, V)$
  if $I(V') \geq I(V)$ **then**
    continue
  end if
  Append (r, d) to P; $V \leftarrow V'$.
  if $I(V) \leq \tau$ **then**
    break
  end if
end for
$R \leftarrow \textsc{ReserveEligible}(P, \text{concurrency limit, cooldown})$
return (P, R)

Algorithm 1: ARI decision and virtual migration plan

Under the stated one-reservation limit, R may be a proper subsequence of P, and the planner's virtual post-state is not a prediction that all planned moves have already committed. If candidates are exhausted, the algorithm may return an improving but still unbalanced plan.

Input: Request r with b_r KV bytes, virtual state V, reserve s
Output: A target worker or $\perp$

$A \leftarrow \emptyset$
for each worker g in worker-ID order **do**
  if g is the source **or** unavailable **or** $C_g - U_g < b_r + s$ **then**
    continue
  end if
  $V_g \leftarrow \textsc{VirtualMove}(r, g, V)$
  $f_g \leftarrow \mathrm{clip}_{[0,1]}((C_g - U_g - b_r)/C_g)$
  $\ell_g \leftarrow \max\{\text{compute utilization}_g, \mathrm{clip}_{[0,1]}(\text{queue length}_g/\text{queue scale})\}$
  $t_g \leftarrow \mathrm{clip}_{[0,1]}(\widehat{T}(r, g)/\text{cost scale})$
  $z_g \leftarrow \mathrm{clip}_{[0,1]}(I(V_g)/\text{imbalance scale})$
  $a_g \leftarrow 0.30 f_g + 0.20(1 - \ell_g) + 0.20(1 - t_g) + 0.30(1 - z_g)$
  Append (g, a_g, z_g) to A.
end for
if $A = \emptyset$ **then**
  return $\perp$
end if
return the g with maximum a_g; break ties by smaller z_g, then smaller worker ID.

Algorithm 2: Fixed target selection for an active request

The target rule is fixed; it is not subject to online rule growth. The optional development-only cost-budget filter is omitted here because it is disabled in the default configuration. The target cost estimate $\widehat{T}$ and the separately calibrated execution-mode selector need not be identical.

Input: Decision-time features x, observed success flag and outcome, rules Q, FIFO feedback buffer B

$h \leftarrow \mathrm{clip}_{[0,1]}(\max\{(I_{\text{before}} - I_{\text{after}})/\max(I_{\text{before}}, \varepsilon), 0\})$ if successful, otherwise $h \leftarrow 0$.
$f \leftarrow \mathrm{clip}_{[0,1]}(\text{released fraction})$ if successful, otherwise $f \leftarrow 0$.
$t \leftarrow \mathrm{clip}_{[0,1]}(\text{migration seconds}/s_t)$; $e \leftarrow \mathrm{clip}_{[0,1]}(\text{extra-latency seconds}/s_e)$.
$y \leftarrow \mathrm{clip}_{[0,1]}(0.40h + 0.20f - 0.20t - 0.20e)$.
Compute $p = \widehat{p}(x)$ and all $m_i(x)$ *at feedback time*.
for each existing priority rule i **do**
  $w_i \leftarrow \mathrm{clip}_{[0.10,2.00]}(w_i + 0.05 m_i(x)(y - p))$.
end for
Push (x, y) into FIFO B, keeping at most 256 samples.
if $|B| < 2$ **then**
  return
end if
$N_x \leftarrow \{z \in B : \|z.x - x\|_2 \le 0.15\}$.
$\rho(x) \leftarrow |N_x|/\,\mathrm{vol}_5(0.15)$; $\bar{\rho} \leftarrow |B|^{-1} \sum_{z \in B} \rho(z.x)$.
if $\rho(x) > 1.20\bar{\rho}$ **and** $\max_{i \in Q} m_i(x) < 0.20$ **then**
  Append a priority rule with center $\mathrm{mean}_{z \in N_x}\, z.x$, per-dimension scale $\max\{\mathrm{std}_{z \in N_x}\, z.x_j, 0.05\}$, consequent $\mathrm{mean}_{z \in N_x}\, z.y$, and confidence 1.
end if

Algorithm 3: Default feedback, confidence update, and priority-rule growth

The update above is the default `legacy` update, not a gradient step. The switches that disable confidence updates or rule growth independently simply skip their respective parts of Algorithm 3. Execution uses a common migration coordinator. A frozen, independently calibrated selector chooses Direct-KV only if it is available and its *predicted* savings over token re-prefill are positive and clear both configured absolute and relative margins; ties use token re-prefill. Commit, ownership transfer, and rollback are runtime state-machine actions, not consequences of the virtual planning proofs below.

Proposition 1 (bounded scores). For every valid rule library and candidate feature vector, $\widehat{p}(x) \in [0,1]$. For every target that passes Algorithm 2's feasibility filter, $a_g \in [0,1]$. The utility y is also in $[0,1]$. *Proof.* Each $m_i(x) \in (0,1]$ and $w_i \geq 0$. When $Z(x) > \varepsilon$, $\widehat{p}(x)$ is a convex combination of consequents $y_i \in [0,1]$; otherwise the implementation returns zero. Each of f_g, $1-\ell_g$, $1-t_g$, and $1-z_g$ belongs to $[0,1]$, and the four nonnegative target weights sum to one. Finally, the utility is explicitly clipped to $[0,1]$. □

Proposition 2 (snapshot feasibility and monotone virtual imbalance). Assume the input snapshot is internally consistent and $s \geq 0$. After every accepted virtual assignment in Algorithm 1, each initially placed request still has exactly one virtual owner (and any unplaced request stays unplaced), total accounted placed-request KV bytes are unchanged, no destination exceeds its modeled KV capacity, and the virtual imbalance is strictly lower than immediately before that assignment. *Proof.* The claims hold before planning by the consistency assumption. For an accepted assignment (r, g), the selector verifies that g is available, differs from the current source, and has at least $b_r + s$ free bytes. The virtual move subtracts b_r from the source, adds b_r to g, and changes only r's owner, preserving both total bytes and uniqueness. Since $s \geq 0$, the destination remains within capacity. The plan appends the move only after checking $I(V') < I(V)$. Induction establishes all claims. □

This is a property of the *virtual snapshot*; concurrent arrivals, growth, stale telemetry, actual reservation and data-plane execution require separate runtime checks and cannot be declared safe solely by this proof.

Proposition 3 (first-crossing prefix, not global optimality). Fix the ranked list and the deterministic target-selection rule. If the planner's accepted sequence is $a_1, \ldots, a_k$ and it stops upon reaching $I(V_k) \leq \tau$, then every proper accepted prefix remains above τ. The planning loop terminates after examining at most one pass over its finite candidate list. *Proof.* Algorithm 1 tests the balance predicate after each accepted move and breaks at the first true test. Earlier accepted prefixes therefore failed it. Each candidate is examined at most once, so the finite loop terminates. □

This is the smallest *accepted prefix in that fixed order*. It is not the global minimum number of migrations over other request orders or target assignments; it is not a guarantee of reaching balance when candidates are exhausted. Runtime concurrency and cooldown can further reduce the number actually reserved or committed in one decision interval.

Proposition 4 (bounded feedback state). With the default confidence interval $[0.10, 2.00]$, every updated old rule retains confidence in that interval. Each appended rule has a center in $[0,1]^5$, scales at least 0.05, consequent in $[0,1]$, and confidence 1. After F feedback calls, the buffer contains at most 256 samples and the rule count is at most $|Q_0| + F$. *Proof.* Confidence projection enforces the first bound. Neighborhood features and utilities lie in the unit interval, so their means lie there; the scale floor and initial confidence give the remaining rule properties. The FIFO capacity is fixed at 256, and each feedback call appends at most one rule. □

The density-and-coverage test is an eligibility rule, not a statistical generalization theorem: it does not by itself imply smaller prediction error or improved serving performance.

Proposition 5 (conservative predicted-mode choice). Assume $\widehat{T}_{\mathrm{R}} > 0$ and nonnegative configured margins. Let the frozen mode selector estimate token re-prefill and host-staged Direct-KV costs as

$$\widehat{T}_{\mathrm{R}} = t_{\mathrm{R}} + n/v_{\mathrm{R}}, \qquad \widehat{T}_{\mathrm{D}} = t_{\mathrm{D}} + B\left(v_{\mathrm{D2H}}^{-1} + v_{\mathrm{host}}^{-1} + v_{\mathrm{H2D}}^{-1}\right), \tag{17}$$

where n is the computed-token count and B is the block-rounded KV payload. Whenever it chooses Direct-KV, its estimated saving $\Delta = \widehat{T}_{\mathrm{R}} - \widehat{T}_{\mathrm{D}}$ is strictly positive and satisfies both configured margins $\Delta \geq \delta_{\mathrm{abs}}$ and $\Delta/\widehat{T}_{\mathrm{R}} \geq \delta_{\mathrm{rel}}$. *Proof.* The selector checks Direct-KV capability, then requires $\Delta > 0$ and both margin predicates before returning Direct-KV. All other branches return token re-prefill. □

This is a claim about frozen *estimates*; model error and execution variance prevent any theorem that Direct-KV is actually faster.

Proposition 6 (optional development update only). Under the separately described `normalized_gradient` override's acceptance test, a confidence update cannot increase the

absolute error on the *current* feedback sample. If an update is accepted, that error decreases strictly.
Proof. For $Z(x) > \varepsilon$,

$$\frac{\partial \widehat{p}(x)}{\partial w_i} = \frac{m_i(x)(y_i - \widehat{p}(x))}{Z(x)}. \tag{18}$$

Thus $(y - \widehat{p}(x))m_i(x)(y_i - \widehat{p}(x))/Z(x)$ is the negative gradient direction of the half-squared current-sample error. By the stated acceptance test, projected candidate confidences are committed together only when recomputing the prediction gives strictly smaller absolute error; failed attempts leave confidences unchanged. □

This last proposition applies only to the development override, which is disabled by default. It gives no guarantee for future samples, true utility, latency, throughput, or a comparison against another scheduler. Indeed, the default confidence update need not reduce even the current-sample error: with two equal kernel matches, consequents 0.2 and 0.8, equal initial confidences, and feedback utility 0.8, both default confidence increments are equal. Projection also preserves their equality, so the prediction remains 0.5 on repeated identical feedback.

Proposition 7 (conditional interactions beyond a fixed linear score). There are priority orderings on valid five-dimensional feature vectors that no affine score $s(x) = b + \sum_{j=1}^{5} a_j x_j$ can realize, but a finite ARI rule library can realize with strict separation. *Proof.* Fix dimensions three through five to the same constants in four feature vectors. In dimensions one and two, use the four corners $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$. Require both diagonal corners to rank strictly above both off-diagonal corners. Every affine score obeys

$$s(0, 0) + s(1, 1) = s(0, 1) + s(1, 0), \tag{19}$$

where the common fixed coordinates are suppressed. The required strict ordering would make the left-hand side strictly greater, a contradiction. For ARI, put one equal-confidence Gaussian rule at each corner, with consequents $y_{00} = y_{11} = 0.9$ and $y_{01} = y_{10} = 0.1$, and use common positive scales σ on the first two dimensions. As $\sigma \to 0^+$, the matching rule's kernel is 1 and every other corner's kernel tends to 0. Equation equation 2 therefore approaches the assigned consequent at each corner. By continuity, sufficiently small positive σ yields the required strict ordering. □

This is a representational statement, not evidence that the three default initial rules, the density/coverage growth heuristic, or any particular GPU experiment achieves such a ranking or better end-to-end performance.