Memory-Aware GPU Resource Scheduling Prediction for DLRM Inference Services with BiGRU-Transformer Models

Wong Chi Hang
Computer Science, City University of Hong Kong, Hong Kong, HK, China
chihang.wong82@yahoo.com

Abstract

Deep learning recommendation models (DLRMs) combine large sparse embeddings with latency-sensitive inference, making host-memory availability as important as accelerator capacity. This study examines per-instance memory-reservation prediction using the Alibaba cluster-trace-gpu-v2025 dataset. The trace contains 23,871 instances from 156 services, including 16,485 CPU-node (CN) instances and 7,386 heterogeneous GPU-node (HN) instances. A BiGRU-Transformer encodes the eight most recent instances of the same service and fuses that history with current CPU, GPU, RDMA, disk, role, and density specifications. It is compared with train-median, role-app median, ridge regression, random forest, gradient boosting, static MLP, GRU, and Transformer baselines under a chronological 70/15/15 split. On the 3,581-instance test period, the BiGRU-Transformer achieves 16.82 GiB MAE and 0.9811 R2, the strongest result among the neural sequence variants. RandomForest achieves the lowest overall MAE at 1.64 GiB and an R2 of 0.9979, indicating that tree partitions capture the trace's discrete reservation templates particularly well. A capacity-normalized scheduling replay converts prediction error into placement risk: RandomForest records 473.86 memory-overflow node-hours, compared with 4,656.53 for the role-app median baseline and 3,332.46 for the BiGRU-Transformer. The results show that same-service histories add predictive value to neural models, while tree ensembles provide more accurate reservations on this trace. A practical scheduler can therefore combine template-sensitive tabular prediction with sequence-based drift signals and role-aware safety margins.

Keywords: *DLRM inference serving; GPU disaggregation; memory prediction; resource scheduling; BiGRU; Transformer; Alibaba cluster trace; capacity-normalized replay.*

1. Introduction

Large-scale recommender systems have progressed from item-to-item collaborative filtering and matrix-oriented ranking toward deep architectures that combine dense features, sparse categorical embeddings, and nonlinear interaction layers [1]-[6]. In production DLRMs, embedding tables and feature processing consume substantial CPU and host-memory capacity, whereas an inference instance may need only a small fraction of the GPU resources typical of training workloads [7], [8]. Prism addresses this mismatch by separating CPU-rich CNs from heterogeneous GPU-bearing HNs and connecting the two pools through RDMA [9]. The resulting architecture improves accelerator utilization, but it also makes accurate host-memory reservation central to safe placement.

Evidence from large heterogeneous clusters shows that GPU efficiency depends on more than accelerator count. Recurring workloads, resource heterogeneity, placement constraints, and fragmentation interact across CPU, memory, storage, network, and GPU dimensions [10]-[12]. Dominant resource fairness and multi-resource packing formalize the need to consider several capacities simultaneously [13], [14], while Paragon and Quasar demonstrate that workload classification and interference-aware allocation can protect quality of service in heterogeneous data centers [15], [16]. Learned schedulers such as DeepRM and Decima further show that workload history can inform placement and sequencing decisions when the trace contains sufficient temporal structure [17], [18].

For DLRM serving, memory underestimation has a direct operational consequence. A node can have free GPU slots while lacking enough host memory for another instance, and an apparently efficient packing decision may create overflow when multiple underpredicted instances overlap. Conversely, excessive memory estimates reduce

consolidation and strand capacity. The prediction problem is therefore not only statistical: the error must be interpreted through the placement decisions that it induces.

This study uses the Alibaba cluster-trace-gpu-v2025 DLRM serving trace and treats memory_request as the target available at the scheduling interface. The released records include role, application group, CPU, GPU, RDMA, memory, disk, density constraints, and lifecycle timestamps. They do not include physical node identifiers or runtime memory counters, so the analysis predicts the stated reservation and evaluates placement in a capacity-normalized replay rather than reconstructing the original production topology.

The central research question is whether recent same-service history improves per-instance memory prediction beyond static resource specifications. The proposed BiGRU-Transformer combines a bidirectional recurrent encoder, self-attention, and a static feature branch. It is evaluated against linear, tree-based, nonparametric, and neural baselines on a future-only test interval. The study also links regression error to high-memory triage and memory-overflow node-hours, allowing model selection to reflect both predictive fidelity and scheduling safety.

2. Literature Review

2.1 DLRM Serving and Multi-Resource Scheduling

DLRM serving differs from conventional dense-model inference because sparse embedding access shifts much of the footprint toward CPU and memory subsystems [6]-[9]. Disaggregation separates these bottlenecks so that CN and HN pools can scale independently, yet it also exposes a tighter coupling between reservation quality and placement feasibility [9]. Earlier cluster studies documented recurrent workload patterns, low utilization, and fragmentation in production GPU environments [10]-[12]. These observations motivate models that preserve application identity and temporal context rather than treating every arrival as an exchangeable row.

Classical schedulers provide the resource-allocation foundation for this setting. Dominant resource fairness balances competing demands across multiple resource types [13], and multi-resource packing improves consolidation when jobs have heterogeneous vectors [14]. Paragon and Quasar incorporate workload characterization and interference sensitivity into placement [15], [16]. DeepRM and Decima introduce learned policies for resource management and cluster scheduling [17], [18]. The present problem is narrower than end-to-end policy learning: it estimates one high-impact component of the resource vector before applying a deterministic multi-resource placement rule.

2.2 Temporal Representation and Structured Regression

Gated recurrent units summarize ordered histories with fewer gates than LSTMs [19], while LSTMs remain a standard mechanism for retaining long-range state [20]. Self-attention provides direct interaction among sequence positions and can reweight earlier observations according to their relevance to the current instance [21]. A short same-service window is well suited to combining these mechanisms: recurrence encodes local order, and attention emphasizes the most informative prior reservations before fusion with current resource fields.

Reservation traces are also highly structured tabular data. Random forests can partition discrete application-resource templates and model nonlinear interactions without feature scaling [22], while gradient boosting adds weak learners sequentially to correct residual structure [23]. Including these baselines is essential because a more expressive sequence network need not outperform a tree ensemble when applications repeatedly request a small set of standard configurations.

2.3 Capacity Forecasting, Uncertainty, and Operational Risk

Recent AI-infrastructure studies extend resource forecasting beyond point accuracy. Time-series foundation models with calibrated uncertainty have been applied to heterogeneous GPU capacity forecasting [24], and cross-cloud transfer learning addresses workload and topology shift across environments [25]. Conformal demand envelopes support risk-bounded oversubscription [26], while salable-supply forecasting for disaggregated recommendation serving connects instance readiness, scheduling delay, and capacity risk [27]. Noisy-neighbor

residual modeling adds a complementary view by identifying degradation that cannot be explained by a VM's own workload alone [28].

Forecasting horizon and cold-start behavior are additional operational concerns. Multi-horizon experiments on Alibaba GPU traces compare statistical and deep models under peak-aware risk curves [29], and few-shot time-series methods target new inference tenants with limited history [30]. These studies focus mainly on aggregate demand or tenant-level forecasting. The current analysis instead predicts the reservation of each arriving DLRM instance and evaluates how errors accumulate over concurrent lifecycle intervals.

Admission and planning studies broaden the objective from accuracy to business and infrastructure outcomes. Profit-aware spot-GPU admission control balances opportunistic utilization against protection costs for high-priority workloads [31], while power-aware inventory planning combines job-level demand forecasting with infrastructure constraints [32]. Post-placement observability remains important because resource stress can surface as service faults; multi-source root-cause attribution [33] and bilingual RAG-assisted incident triage [34] illustrate how heterogeneous operational evidence can support diagnosis. Hybrid-cloud deployment work emphasizes cost and placement trade-offs across environments [35], and capacity-dashboard research translates forecast risk into visual decision artifacts for operators [36]. Together, these directions support a pipeline in which per-instance prediction, conservative placement, monitoring, and human interpretation form a connected control loop.

The gap addressed here lies between aggregate capacity forecasting and full scheduling-policy learning. The model uses only scheduling-visible fields and prior same-service instances, produces an interpretable GiB reservation, and is judged both by held-out regression and by a replay that exposes the cost of underestimation. This formulation makes it possible to compare sequence learning with strong template-sensitive tabular baselines under the same operational constraints.

3. Method

3.1 Dataset and Prediction Target

The empirical pipeline is summarized in Figure 1. The input is the disaggregated_DLRM_trace.csv file associated with the GPU-disaggregated serving study [9]. All 23,871 rows are retained and ordered by scheduling-visible start time. When creation_time is absent because an instance was already active at the beginning of observation, start_time is set to zero. When deletion_time is absent, end_time is set to the maximum observed timestamp. These rules create closed lifecycle intervals for temporal analysis and replay without inventing node identities.

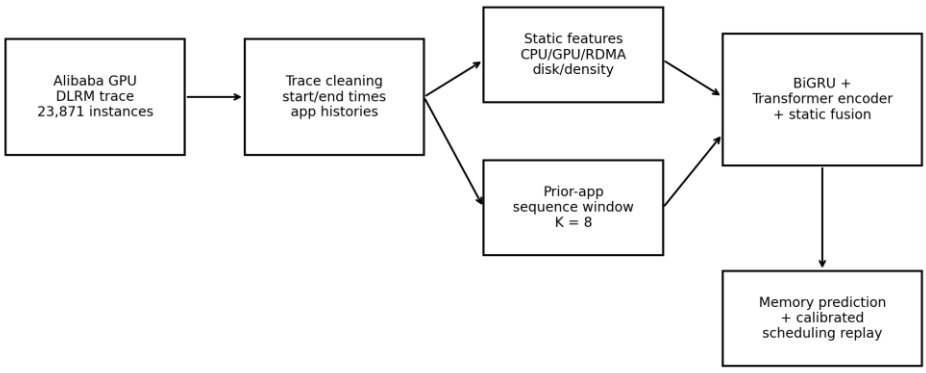

Figure 1. End-to-end pipeline for trace preprocessing, feature construction, BiGRU-Transformer prediction, and scheduling replay.

The supervised target is memory_request in GiB. Memory_limit is excluded because it duplicates the reservation target in this trace. Scheduled_time is used for descriptive delay statistics and event ordering, but not as an

admission-time feature because it is produced after the scheduling decision. Table 1 lists the source fields and shows how each field contributes to prediction, stratification, or replay.

Table 1. Dataset fields and their role in the prediction and replay pipeline.

Field	Dataset meaning	Use in this study
instance_sn	unique instance identifier	index only
role	CN or HN disaggregated role	categorical input and stratification
app_name	inference service group	categorical input and app-history sequence
cpu_request/cpu_limit	vCPU reservation and limit	numeric input
gpu_request/gpu_limit	GPU reservation and limit	numeric input and replay constraint
rdma_request/rdma_limit	RNIC bandwidth percentage	numeric input and replay constraint
memory_request	main-memory reservation in GiB	prediction target
disk_request/disk_limit	disk reservation and limit in GiB	numeric input and replay constraint
max_instance_per_node	same-app density cap; -1 means no limit	scheduling replay constraint
creation/scheduled/deletion_time	relative lifecycle timestamps in seconds	temporal split and replay intervals

3.2 Temporal Split and Feature Construction

For instance i of application a, the model receives a current-row vector x_i and a fixed-length sequence S_i containing up to eight earlier instances from the same application. The target y_i is the current memory_request. Static inputs include role, app_name, cpu_request, gpu_request, rdma_request, disk_request, disk_limit, max_instance_per_node, a creation-time-missing indicator, trace day, application count before the current row, previous memory reservation, rolling prior-memory means over three and eight instances, and elapsed time since the preceding instance of the same application. A max_instance_per_node value of -1 is represented by an explicit unlimited-density indicator rather than treated as a negative resource quantity.

The split follows chronological order: the first 70% of rows form the training set, the next 15% form validation, and the final 15% form the test period. Numeric transformations are fitted on training rows only. Every application-history feature is calculated in a single forward pass before the current row is added to the application state, preventing future reservations from entering earlier features. Table 2 summarizes the two disaggregated roles, Table 3 reports the temporal ranges and sample counts, and Table 4 describes lifecycle and scheduling-delay statistics.

Table 2. Dataset summary by disaggregated role.

Role	Instances	Apps	CPU mean	GPU sum	RDMA mean	Memory mean	Memory median	Memory p95	Disk mean
CN	16485	156	72.17	0	17.64	363.9	320	480	366.4
HN	7386	156	8.517	7386	27.86	46.83	40	120	178.5

Table 3. Chronological training, validation, and test split.

Split	Instances	Apps	Start day min	Start day max	Memory mean	Memory p95
train	16709	145	0	20.61	258.3	480
val	3581	105	20.61	24.48	272.2	480
test	3581	115	24.48	30.98	294.7	480

Table 4. Lifecycle and scheduling-delay statistics from the released timestamps.

Role	Schedulable records	Nonzero delay rate	Mean delay (s)	p90 delay (s)	p99 delay (s)	Median runtime (s)
CN	12328	0.223	78.17	55	523.4	1.656e+05
HN	4263	0.277	505.3	116	9764	6.303e+05

3.3 BiGRU-Transformer Architecture

The network contains static and sequential branches. The static branch embeds application and role identifiers, concatenates them with standardized current-row numeric features, and applies a two-layer multilayer perceptron. The sequence branch receives the previous K = 8 same-application instances with role, CPU, GPU, RDMA, disk, density cap, trace day, prior memory, and application count. A bidirectional GRU encodes ordered context in both directions within the observed window [19], [20]. The GRU outputs then pass through a single Transformer-style self-attention block with four heads [21]. Mean pooling produces a sequence context vector, which is concatenated with the static representation and mapped to a scalar memory estimate.

The loss is SmoothL1 on standardized memory. Each neural model is trained for four epochs, and the checkpoint with the lowest validation MAE is used for the held-out evaluation. The compact design keeps the comparison focused on architectural differences rather than parameter scale. Bidirectionality is applied only inside the already-observed historical window; no future instance relative to the prediction target enters the sequence.

3.4 Baselines and Training Configuration

The comparison includes eight alternatives. TrainMedian predicts the training-set median for every row. RoleAppMedian uses the training median for each application-role pair and falls back to the role median. Ridge regression uses one-hot application and role indicators with scaled numeric features. RandomForest uses 60 trees, maximum depth 12, and minimum leaf size 3 [22]. GradientBoosting uses 120 estimators, depth 3, and learning rate 0.05 [23]. Static-MLP removes the history branch, GRU retains a unidirectional recurrent branch without attention, and Transformer uses self-attention without recurrent encoding. Table 5 summarizes the inputs and hyperparameters.

Table 5. Model inputs and hyperparameters used in the empirical comparison.

Model	Input design	Hyperparameters
TrainMedian	constant median of training memory	no trainable parameters
RoleAppMedian	training median per (app, role), fallback to role median	nonparametric
Ridge	one-hot app/role plus scaled numeric features	alpha=1.0
RandomForest	numeric features plus app_id and role_id	60 trees, depth 12, min leaf 3

Model	Input design	Hyperparameters
GradientBoosting	numeric features plus app_id and role_id	120 estimators, depth 3, lr=0.05
Static-MLP	static numeric features plus app/role embeddings	SmoothL1, AdamW, 4 epochs
GRU	K=8 prior-app sequence plus static branch	unidirectional GRU
Transformer	K=8 prior-app sequence plus static branch	one self-attention encoder block
BiGRU-Transformer	BiGRU sequence encoder followed by self-attention and static fusion	K=8, hidden=32

The linear and tree baselines are implemented with scikit-learn [38], and the neural models are implemented with PyTorch [39]. Neural optimization uses AdamW, an Adam-family optimizer [37], with a fixed seed of 42. Table 6 records validation loss, training epochs, and the resulting test metrics for the four neural variants.

Table 6. Neural training log and held-out metrics.

Val. loss	Epochs used	Model	Epochs requested	MAE (GiB)	RMSE (GiB)	MdAE (GiB)	MAPE (%)	R2
0.0145	4	BiGRU-Transformer	4	16.82	27.37	10.24	16.95	0.9811
0.01483	4	Transformer	4	22.87	34	16.54	13.24	0.9709
0.01408	4	GRU	4	17.74	27.97	11.92	13.54	0.9803
0.02547	4	Static-MLP	4	27.52	42.55	18.07	17.3	0.9544

3.5 Capacity-Normalized Scheduling Replay

The scheduling replay is a feasibility audit rather than a reconstruction of the production cluster. Because physical node identifiers and hardware capacities are unavailable, the replay defines normalized bins by rounding the largest per-instance requests upward. CN bins contain 256 vCPU, 1,024 GiB memory, 1,024 GiB disk, 100 RDMA units, and no GPU. HN bins contain 32 vCPU, 8 GPU slots, 512 GiB memory, 1,024 GiB disk, and 100 RDMA units. Each instance enters at start_time and departs at end_time; departures at a timestamp are processed before arrivals at the same timestamp.

A best-fit rule reserves CPU, GPU, RDMA, disk, and predicted memory while enforcing the same-application density cap. After every event, actual memory_request is audited on each active normalized node. Whenever actual memory exceeds capacity, the excess interval contributes to memory-overflow node-hours. Peak node count measures consolidation, whereas overflow node-hours measure the safety cost of under-reservation. Reporting both prevents aggressive underprediction from appearing beneficial merely because it packs more instances onto fewer nodes.

3.6 Evaluation Metrics

Regression quality is measured with MAE, RMSE, median absolute error (MdAE), MAPE, and R2. MAE is primary because GiB error has a direct placement interpretation; RMSE exposes larger misses, and MdAE

separates typical template matching from tail behavior. The high-memory analysis converts predictions into a binary flag at the training-set 75th percentile, 320 GiB, and reports accuracy, precision, recall, F1, AUROC, and AUPRC. The replay reports peak and time-weighted node counts, memory and HN GPU utilization, overflow node-hours, replay duration, and the number of placed instances.

4. Results and Discussion

4.1 Trace Characteristics

Table 2 shows a pronounced role asymmetry. The trace contains 16,485 CN instances and 7,386 HN instances across 156 applications. CN instances average 363.9 GiB of requested memory, with a median of 320 GiB and a p95 of 480 GiB. HN instances average 46.83 GiB, with a median of 40 GiB and a p95 of 120 GiB. Figure 2 visualizes this separation: most HN reservations occupy a compact range, whereas CN reservations span a much higher memory regime. This difference supports role-stratified evaluation because a modest absolute error can be large relative to HN variance even when overall R2 remains high.

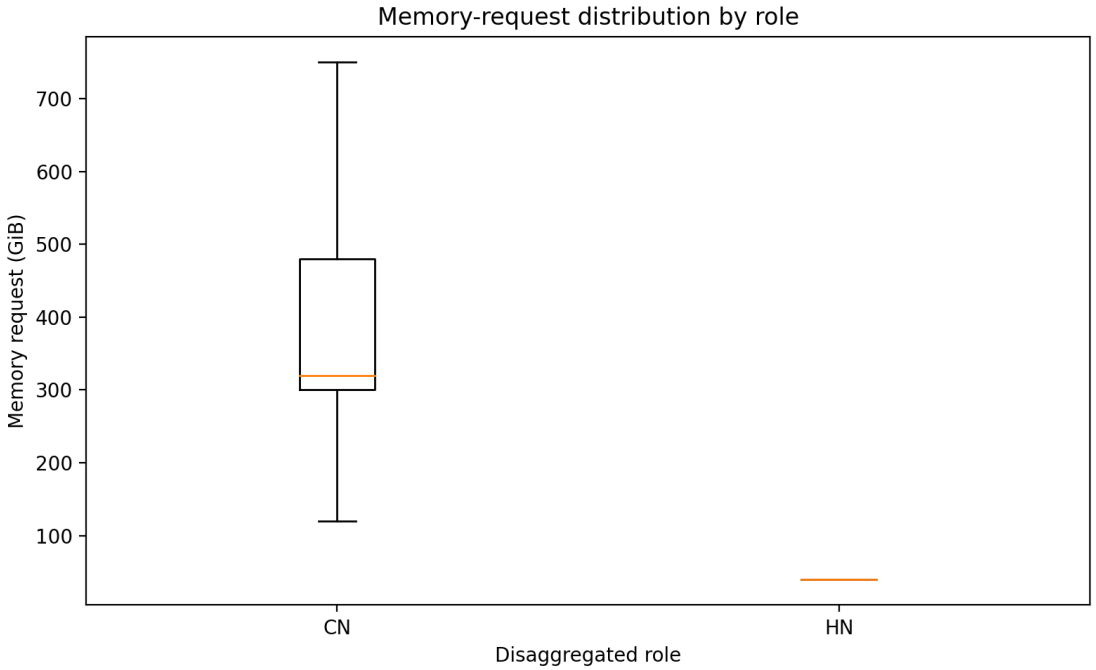

Figure 2. Memory-request distribution by CN and HN role; fliers are hidden to show the central reservation range.

The chronological split in Table 3 covers day 0 through day 30.98. Training contains 16,709 instances, validation contains 3,581, and the future test period contains 3,581 instances from 115 application groups. Table 4 shows that scheduling delay is sparse but operationally meaningful. Among rows with both creation and scheduling timestamps, the nonzero-delay rate is 22.3% for CN and 27.7% for HN. Mean HN delay reaches 505.3 seconds, and the HN p99 is 9,764 seconds, indicating that resource constraints can produce long tails even when median behavior is moderate.

Figure 3 shows daily arrivals and aggregate requested memory. The two series generally move together, but their ratio changes as role mix and reservation templates vary. The large opening-day volume reflects instances already present at the beginning of the observed trace, while later peaks capture new service activity. This nonstationarity makes a future-only split more informative than shuffled cross-validation and creates a practical role for recent application history.

ISSN: 3071-4656
DOI: 10.66372/JGER.V4I2.3

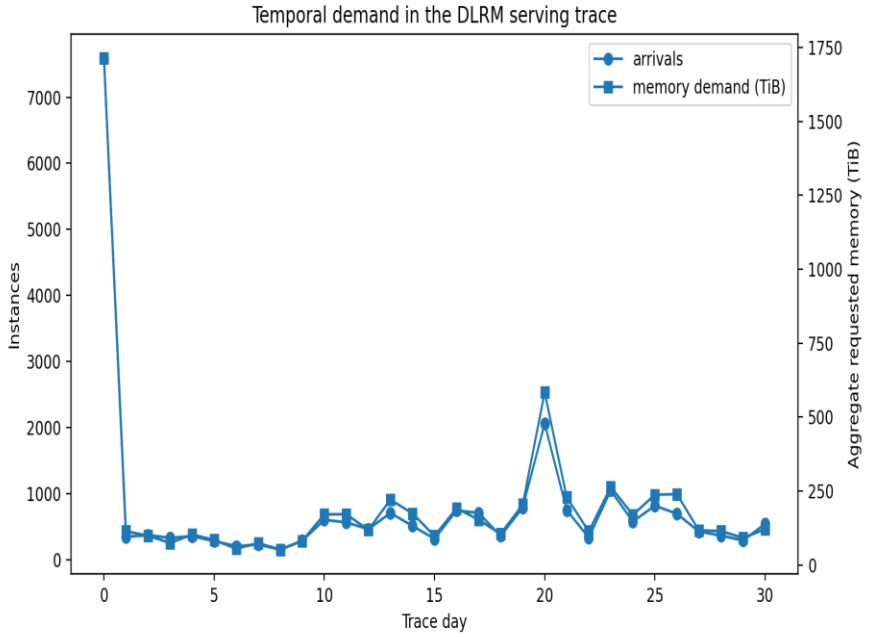

Figure 3. Temporal demand measured by daily arrivals and aggregate requested memory.

4.2 Held-Out Prediction Accuracy

Table 6 first establishes the neural comparison. BiGRU-Transformer achieves 16.82 GiB MAE, 27.37 GiB RMSE, and 0.9811 R2. GRU follows at 17.74 GiB MAE, Transformer at 22.87 GiB, and Static-MLP at 27.52 GiB. The sequence branch therefore provides a substantial improvement over static neural features, and combining recurrent encoding with attention produces the lowest neural error.

The complete comparison appears in Table 7 and Figure 4. RandomForest is the best overall predictor with 1.643 GiB MAE, 9.102 GiB RMSE, 0.0023 GiB MdAE, and 0.9979 R2. GradientBoosting ranks second at 4.475 GiB MAE, followed by Ridge at 11.57 GiB. BiGRU-Transformer remains the strongest neural sequence model but does not match the tree ensembles. The result is consistent with the trace structure: repeated application-role configurations create discrete reservation templates that recursive tree partitions can isolate with very little median error.

Table 7. Overall held-out memory-regression comparison on the temporal test split.

Model	MAE (GiB)	RMSE (GiB)	MdAE (GiB)	MAPE (%)	R2
RandomForest	1.643	9.102	0.0023	2.244	0.9979
GradientBoosting	4.475	11.8	1.712	4.031	0.9965
Ridge	11.57	18.65	7.943	11.74	0.9912
BiGRU-Transformer	16.82	27.37	10.24	16.95	0.9811
GRU	17.74	27.97	11.92	13.54	0.9803
Transformer	22.87	34	16.54	13.24	0.9709
Static-MLP	27.52	42.55	18.07	17.3	0.9544
RoleAppMedian	29.47	79.4	0	6.223	0.8411
TrainMedian	159.8	200.8	160	224.4	-0.0162

ISSN: 3071-4656
DOI: 10.66372/JGER.V4I2.3

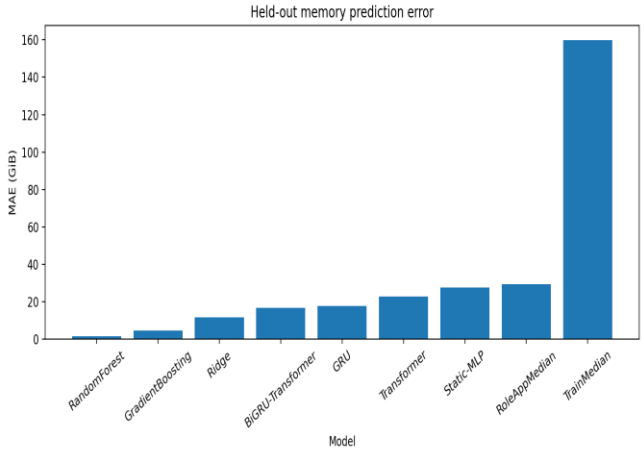

Figure 4. Held-out MAE comparison across all models.

Figure 5 plots RandomForest predictions against actual reservations. Most points lie close to the identity line, especially around the recurring 40, 120, 320, and 480 GiB templates. The remaining errors are concentrated in less frequent high-memory configurations, which explains the difference between the near-zero MdAE and the larger RMSE.

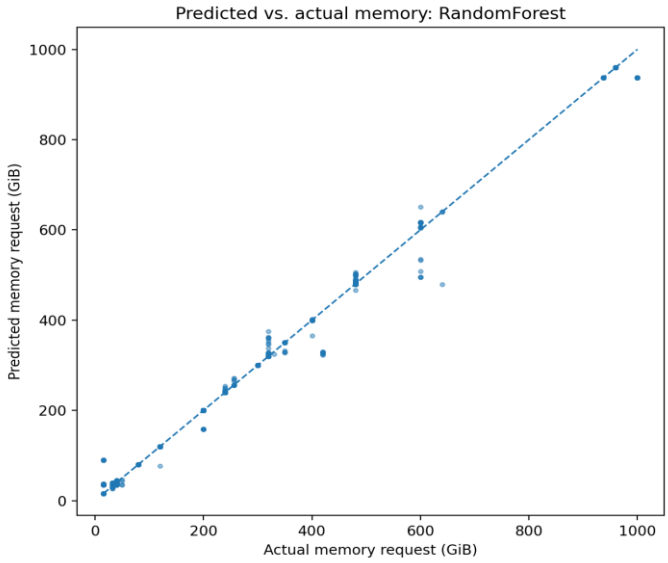

Figure 5. Predicted versus actual memory for RandomForest, the model with the lowest test MAE.

Role-level results in Table 8 explain why the overall ranking alone is insufficient. On CN instances, RandomForest records 1.545 GiB MAE and 0.9954 R2; GradientBoosting records 4.942 GiB and 0.9923. BiGRU-Transformer reaches 16.25 GiB MAE and 0.9606 R2. On HN instances, RoleAppMedian is exceptionally strong at 0.3037 GiB MAE because many application-role reservations recur exactly, while RandomForest records 1.891 GiB. BiGRU-Transformer has 18.26 GiB MAE and -0.2409 role-specific R2. The negative HN R2 coexists with a high overall R2 because CN rows dominate target magnitude and variance. For deployment, HN estimates therefore require a separate margin or calibration rule.

Table 8. Held-out memory-regression comparison stratified by CN and HN roles.

Model	Role	n	MAE (GiB)	RMSE (GiB)	MdAE (GiB)	MAPE (%)	R2
RandomForest	CN	2567	1.545	9.704	0.0023	0.2964	0.9954

ISSN: 3071-4656
DOI: 10.66372/JGER.V4I2.3

Model	Role	n	MAE (GiB)	RMSE (GiB)	MdAE (GiB)	MAPE (%)	R2
GradientBoosting	CN	2567	4.942	12.52	1.775	1.419	0.9923
Ridge	CN	2567	10.64	18.28	7.191	2.89	0.9837
BiGRU-Transformer	CN	2567	16.25	28.38	9.472	4.311	0.9606
GRU	CN	2567	19.69	30.75	13.52	5.064	0.9538
Transformer	CN	2567	27.34	38.46	20.41	6.714	0.9277
Static-MLP	CN	2567	32.21	48.38	21.58	8.105	0.8857
RoleAppMedian	CN	2567	40.99	93.76	0	8.445	0.5704
TrainMedian	CN	2567	113.8	161	120	26.44	-0.266
RoleAppMedian	HN	1014	0.3037	3.016	0	0.5983	0.9813
RandomForest	HN	1014	1.891	7.364	0.0109	7.173	0.8888
GradientBoosting	HN	1014	3.291	9.748	0.5706	10.64	0.8052
Transformer	HN	1014	11.56	18.42	7.841	29.77	0.3044
GRU	HN	1014	12.79	19.24	8.832	34.98	0.2407
Ridge	HN	1014	13.91	19.56	9.898	34.14	0.2159
Static-MLP	HN	1014	15.65	21.65	12.77	40.58	0.0392
BiGRU-Transformer	HN	1014	18.26	24.6	14.11	48.94	-0.2409
TrainMedian	HN	1014	276.3	277.1	280	725.6	-156.5

4.3 High-Memory Detection and Scheduling Replay

Table 9 evaluates the 320 GiB high-memory threshold. RandomForest reaches 0.9997 accuracy and F1, while RoleAppMedian reaches 0.9970 F1 and GradientBoosting reaches 0.9882. Among neural models, GRU has the highest F1 at 0.9240. BiGRU-Transformer obtains 0.8546 F1 but 0.9988 AUROC and 0.9989 AUPRC. The gap between ranking and threshold performance indicates that its score ordering is strong while its raw scale is conservative near the decision boundary. A validation-tuned threshold or role-specific calibration could use the representation more effectively for admission triage.

Table 9. High-memory classification using the 320 GiB training-quantile threshold.

Model	Threshold (GiB)	Accuracy	Precision	Recall	F1	AUROC	AUPRC
RandomForest	320	0.9997	1	0.9995	0.9997	1	1
RoleAppMedian	320	0.9966	0.9939	1	0.997	0.9976	0.9962

ISSN: 3071-4656
DOI: 10.66372/JGER.V4I2.3

Model	Threshold (GiB)	Accuracy	Precision	Recall	F1	AUROC	AUPRC
GradientBoosting	320	0.9872	0.9954	0.9812	0.9882	0.9991	0.9991
GRU	320	0.9218	0.9924	0.8645	0.924	0.9968	0.997
Static-MLP	320	0.9073	0.9846	0.8447	0.9093	0.9941	0.9944
Transformer	320	0.8894	0.9987	0.8	0.8884	0.9961	0.9967
Ridge	320	0.8861	1	0.7929	0.8845	1	1
BiGRU-Transformer	320	0.8601	0.998	0.7472	0.8546	0.9988	0.9989
TrainMedian	320	0.5501	0.5501	1	0.7098	0.5	0.5501

The replay in Table 10 converts these errors into placement outcomes. Oracle-ActualMemory requires 527 peak normalized nodes and produces no overflow. RoleAppMedian uses 515 peak nodes but accumulates 4,656.53 overflow node-hours, revealing that its apparent packing efficiency comes from underestimating high-memory cases. RandomForest uses 524 peak nodes and reduces overflow to 473.86 node-hours, the safest non-oracle result. BiGRU-Transformer uses 520 peak nodes and records 3,332.46 overflow node-hours. Its lower peak count relative to RandomForest is therefore not a net advantage because the additional consolidation is accompanied by substantially greater memory risk.

Table 10. Capacity-normalized scheduling replay on 3,581 held-out instances.

Scheduler	Calib. (GiB)	Peak nodes	Peak HN	Mean nodes	Mean HN	Memory util. (%)	HN GPU util. (%)	Overflow node-h	Replay h	Instances
Oracle-Actual Memory	0	527	91	331.6	39.55	73.36	41.86	0	156.4	3581
RoleAppMedian +raw	0	515	91	325.1	39.55	75	41.86	4657	156.4	3581
Random Forest +raw	0	524	91	330	39.7	73.71	41.7	473.9	156.4	3581
BiGRU-Transformer+raw	0	520	92	326.5	39.59	74.62	41.82	3332	156.4	3581

Figure 6 compares peak node counts and reinforces the need to read capacity and overflow together. The four schedulers differ by only 12 peak nodes, yet their overflow exposure ranges from zero to more than 4,600 node-hours. RandomForest offers the strongest measured compromise: it remains close to oracle capacity while sharply limiting unsafe under-reservation.

ISSN: 3071-4656
DOI: 10.66372/JGER.V4I2.3

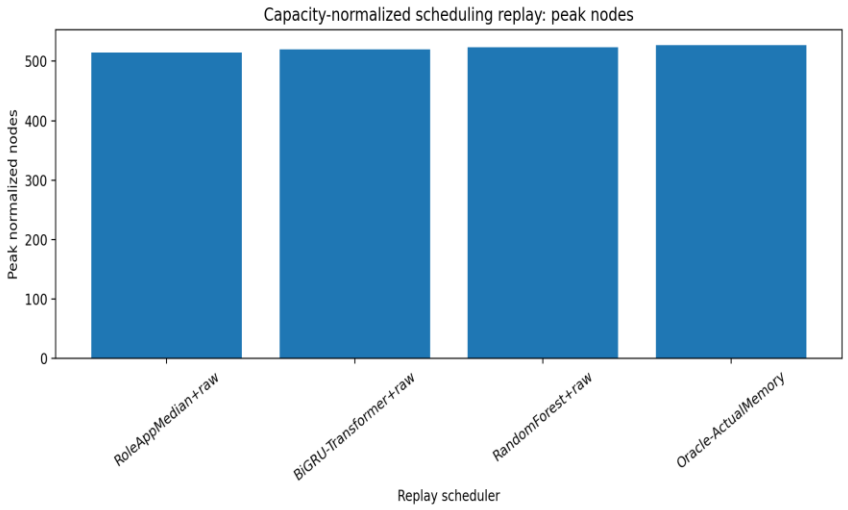

Figure 6. Peak normalized nodes in the capacity-normalized scheduling replay.

4.4 Neural Ablation and Temporal Robustness

Table 11 and Figure 7 isolate the contribution of sequence modeling. Static-MLP records 27.52 GiB MAE. Transformer lowers the error to 22.87 GiB, GRU to 17.74 GiB, and BiGRU-Transformer to 16.82 GiB. Relative to Static-MLP, the combined architecture reduces MAE by 10.70 GiB, or 38.9%. The recurrent branch contributes the largest single gain, suggesting that local ordering in recent same-service histories is particularly informative within the short K = 8 window. Self-attention adds a smaller further improvement by reweighting those historical states before fusion.

Table 11. Neural ablation study on the temporal test split.

Variant	MAE (GiB)	RMSE (GiB)	MdAE (GiB)	MAPE (%)	R2
BiGRU-Transformer	16.82	27.37	10.24	16.95	0.9811
GRU	17.74	27.97	11.92	13.54	0.9803
Transformer	22.87	34	16.54	13.24	0.9709
Static-MLP	27.52	42.55	18.07	17.3	0.9544

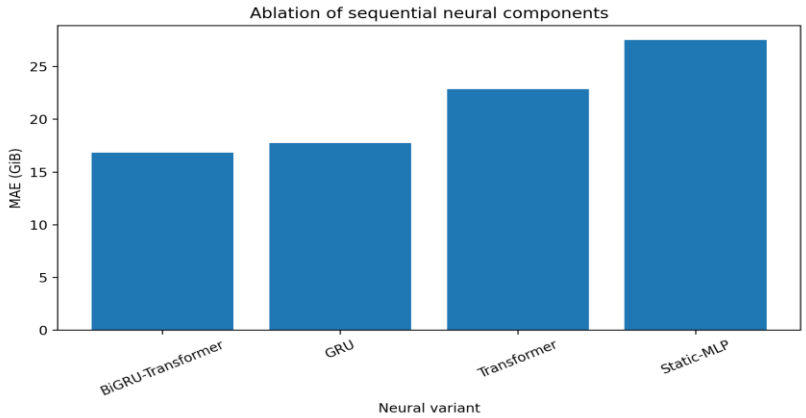

Figure 7. Neural ablation MAE for Static-MLP, GRU, Transformer, and BiGRU-Transformer.

Temporal robustness is reported in Table 12. BiGRU-Transformer achieves 16.38 GiB MAE in the early test tertile, 13.19 GiB in the middle tertile, and 20.88 GiB in the late tertile. R2 remains between 0.9749 and 0.9889,

ISSN: 3071-4656
DOI: 10.66372/JGER.V4I2.3

but the late-period increase in MAE and MAPE indicates drift in application mix or reservation patterns. This behavior supports using the sequence model as a change-sensitive signal even when a tree model remains the default point predictor for established templates.

Table 12. BiGRU-Transformer robustness across held-out test tertiles.

Test segment	n	MAE (GiB)	RMSE (GiB)	MdAE (GiB)	MAPE (%)	R2
early	1194	16.38	27.84	9.963	12.07	0.981
middle	1193	13.19	19.31	8.336	14.47	0.9889
late	1194	20.88	33.14	12.59	24.3	0.9749

4.5 Operational Implications

The results support a hybrid scheduling workflow. For known application-role templates, RandomForest provides accurate, inexpensive point estimates and the lowest non-oracle overflow exposure. In parallel, a sequence model can monitor recent same-service behavior and identify cases where current context departs from historical templates. Large disagreement between the two models can trigger a conservative reservation, a larger safety margin, or placement in a pool with additional memory headroom.

Role-aware control is equally important. CN error dominates aggregate memory risk because CN reservations are large, whereas HN error can affect accelerator admission despite smaller absolute memory values. Separate calibration for CN and HN would prevent the high overall R2 from masking weak HN scale accuracy. Prediction intervals or conformal upper bounds could further convert point estimates into explicit overflow guarantees, aligning the replay objective with risk-bounded capacity management [24], [26].

Finally, the replay illustrates why infrastructure evaluation should combine efficiency and safety. A predictor that lowers peak node count through systematic underestimation may appear attractive until actual concurrent memory is audited. Peak capacity, overflow duration, role-specific error, and threshold behavior provide complementary views of the same scheduling decision and should be reviewed together in operational dashboards [36].

5. Limitations

The target is the requested memory reservation rather than measured runtime consumption. The trace does not expose memory-utilization counters, request-level latency, or service-level objective violations, so the analysis cannot estimate dynamic headroom or link an overflow interval directly to end-user latency.

The capacity-normalized replay uses design constants because physical node identifiers and hardware inventories are absent. It provides a consistent comparison of predictors under identical constraints, but its node counts should not be interpreted as the topology or capacity of the production cluster described in [9].

Repeated reservation templates favor tree-based methods. The strong RandomForest result may not transfer to traces with continuous utilization signals, frequent unseen applications, or substantial changes in model architecture. Longer histories and cold-start methods such as those considered in [30] may become more valuable in those settings.

Neural models were intentionally compact and trained for four epochs. Role-specific heads, asymmetric losses, quantile regression, and uncertainty calibration could improve HN scale accuracy and replay safety. The current replay uses raw predictions so that the direct relationship between point error and overflow remains visible.

The observed period spans approximately one month. The temporal split tests future rows within that interval, but it does not cover seasonal demand, hardware refreshes, or major service migrations. Cross-cloud and longer-horizon studies [25], [29], [35] suggest that deployment should include periodic recalibration and monitoring for distribution shift.

6. Conclusion

Memory-aware placement is a central requirement for GPU-disaggregated DLRM serving because CPU-rich CNs and accelerator-bearing HNs are constrained by different resource profiles. On the 23,871-instance Alibaba trace, the BiGRU-Transformer uses the eight most recent same-service instances to improve neural prediction, reaching 16.82 GiB MAE and 0.9811 R2. It outperforms Static-MLP, GRU, and Transformer ablations, confirming that application history contains useful temporal information.

RandomForest is nevertheless the strongest predictor for this trace, with 1.643 GiB MAE, 0.9979 R2, and 473.86 overflow node-hours in the normalized replay. Its advantage arises from discrete, recurring reservation templates. The operational conclusion is therefore not that one architecture dominates every workload, but that template-sensitive tabular prediction and history-sensitive sequence modeling serve complementary roles. A practical scheduler can use the tree model for accurate reservations, the sequence model for drift detection, and role-aware safety margins to control memory-overflow risk.

References

[1] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," IEEE Internet Computing, vol. 7, no. 1, pp. 76-80, 2003.

[2] C. A. Gomez-Uribe and N. Hunt, "The Netflix recommender system: Algorithms, business value, and innovation," ACM Transactions on Management Information Systems, vol. 6, no. 4, pp. 1-19, 2015.

[3] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for YouTube recommendations," in Proc. ACM Conf. Recommender Systems, 2016, pp. 191-198.

[4] H.-T. Cheng et al., "Wide & deep learning for recommender systems," in Proc. Workshop on Deep Learning for Recommender Systems, 2016, pp. 7-10.

[5] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in Proc. ADKDD, 2017, pp. 1-7.

[6] M. Naumov et al., "Deep learning recommendation model for personalization and recommendation systems," arXiv:1906.00091, 2019.

[7] U. Gupta et al., "The architectural implications of Facebook's DNN-based personalized recommendation," in Proc. IEEE Int. Symp. High Performance Computer Architecture, 2020, pp. 488-501.

[8] U. Gupta et al., "DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference," in Proc. ACM/IEEE Int. Symp. Computer Architecture, 2020, pp. 982-995.

[9] C. Wang, Z. Wen, R. Zhang, P. Xu, and Y. Jiang, "GPU Memory Requirement Prediction for Deep Learning Task Based on Bidirectional Gated Recurrent Unit Optimization Transformer," in 2025 5th International Conference on Artificial Intelligence, Virtual Reality and Visualization (AIVRV), Chengdu, China, 2025, doi: 10.1109/AIVRV67401.2025.11350369.

[10] Q. Weng et al., "MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters," in Proc. USENIX NSDI, 2022, pp. 945-960.

[11] Q. Weng, L. Yang, Y. Yu, W. Wang, X. Tang, G. Yang, and L. Zhang, "Beware of fragmentation: Scheduling GPU-sharing workloads with fragmentation gradient descent," in Proc. USENIX ATC, 2023, pp. 995-1008.

[12] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces," in Proc. IEEE/ACM IWQoS, 2019, pp. 1-10.

[13] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in Proc. USENIX NSDI, 2011, pp. 323-336.

[14] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in Proc. ACM SIGCOMM, 2014, pp. 455-466.

[15] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in Proc. ACM ASPLOS, 2013, pp. 77-88.

[16] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," in Proc. ACM ASPLOS, 2014, pp. 127-144.

[17] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in Proc. ACM HotNets, 2016, pp. 50-56.

[18] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in Proc. ACM SIGCOMM, 2019, pp. 270-288.

[19] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in Proc. EMNLP, 2014, pp. 1724-1734.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[21] A. Vaswani et al., "Attention is all you need," in Proc. Advances in Neural Information Processing Systems, 2017, pp. 5998-6008.

[22] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[23] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," Annals of Statistics, vol. 29, no. 5, pp. 1189-1232, 2001.

[24] S. He, H. Tu, and I. Liu, "Safe PD capacity forecasting with time-series foundation models and calibrated uncertainty for heterogeneous GPU clusters," Journal of Advanced Computing Systems, vol. 3, no. 4, pp. 48-66, Apr. 2023, doi: 10.69987/JACS.2023.30404.

[25] S. He, X. Chang, and E. Sun, "Cross-cloud transfer learning for AI training capacity forecasting under workload and topology distribution shift," Journal of Advanced Computing Systems, vol. 4, no. 1, pp. 100-120, Jan. 2024, doi: 10.69987/JACS.2024.40108.

[26] S. Chen, S. He, and E. Sun, "Risk-bounded GPU resource oversubscription via conformal demand envelopes in production AI clusters," Journal of Advanced Computing Systems, vol. 4, no. 5, pp. 119-134, May 2024, doi: 10.69987/JACS.2024.40509.

[27] H. Tu, S. Zhao, and S. He, "LLM-augmented salable GPU supply forecasting for disaggregated recommendation serving: Predicting instance readiness, scheduling delay, and capacity risk," Journal of Advanced Computing Systems, vol. 4, no. 9, pp. 85-102, Sep. 2024, doi: 10.69987/JACS.2024.40908.

[28] J. Nie and D. Zheng, "Noisy-neighbor-aware VM degradation risk modeling with unsupervised residual fusion," Journal of Advanced Computing Systems, vol. 4, no. 4, pp. 112-123, Apr. 2024, doi: 10.69987/JACS.2024.40409.

[29] S. Zhao, J. Bai, and D. Roberson, "Multi-horizon GPU demand forecasting with workload semantics and operational risk curves: An empirical study on Alibaba Clusterdata GPU trace," Journal of Technology Informatics and Engineering, vol. 4, no. 3, pp. 544-571, Dec. 2025, doi: 10.51903/jtie.v4i3.498.

[30] S. He, C. Li, and H. Rao, "Few-shot cold-start workload forecasting for new AI inference tenants with time-series foundation models," Journal of Technology Informatics and Engineering, vol. 4, no. 1, pp. 306-324, Apr. 2025, doi: 10.51903/jtie.v4i1.546.

[31] S. Zhao, Y. Ren, and X. Chang, "Profit-aware spot GPU admission control with cost-sensitive loss and evidence-grounded policy memos for AI workload supply-demand matching," Journal of Technology Informatics and Engineering, vol. 5, no. 2, pp. 45-59, Jun. 2026, doi: 10.51903/jtie.v5i2.545.

[32] S. He, J. Nie, and C. Li, "Power-aware inventory planning for AI infrastructure using job-level forecasting and LLM workload explanations," Journal of Technology Informatics and Engineering, vol. 5, no. 1, pp. 341-359, Apr. 2026, doi: 10.51903/jtie.v5i1.548.

[33] G. Liu, S. He, and I. Liu, "LLM-augmented multi-source root cause attribution for CPU and network faults in microservices," Journal of Advanced Computing Systems, vol. 3, no. 6, pp. 39-57, Jun. 2023, doi: 10.69987/JACS.2023.30604.

[34] G. Liu, C. Li, and E. Zhang, "OpsLLM for cloud incident triage: Bilingual RAG-based root cause analysis and alert summarization for AI infrastructure operations," Journal of Advanced Computing Systems, vol. 4, no. 4, pp. 97-111, Apr. 2024, doi: 10.69987/JACS.2024.40408.

[35] Q. Xin, "Hybrid cloud architecture for efficient and cost-effective large language model deployment," Journal of Information Systems and Informatics, vol. 7, no. 3, pp. 2182-2195, Sep. 2025, doi: 10.51519/journalisi.v7i3.1170.

[36] G. Liu, S. He, and H. Wong, "LLM-compatible visual brief cards for AI infrastructure capacity dashboards: A UI/UX framework for turning forecast risk into graphic design decisions," International Journal of Graphic Design, vol. 3, no. 1, pp. 196-213, May 2025, doi: 10.51903/ijgd.v3i1.3723.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in Proc. Int. Conf. Learning Representations, 2015.

[38] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[39] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in Proc. Advances in Neural Information Processing Systems, 2019, pp. 8024-8035.