Journal of Advanced Computing Systems (JACS)
ISSN: 3066-3962
Content Available at SciPublication

LLM-Augmented Salable GPU Supply Forecasting for Disaggregated Recommendation Serving: Predicting Instance Readiness, Scheduling Delay, and Capacity Risk

*Haowei Tu [1], * Siming Zhao [1], Samuel He [2]*
[1] Information Systems, New York University, NY, USA
[1] Business Analytics, Columbia University, NY, USA
[2] Applied Analytics, Columbia University, NY, USA
** Corresponding Email: haowei.tu@outlook.com*

DOI: 10.69987/JACS.2024.40908

Keywords

GPU infrastructure, disaggregated serving, DLRM inference, capacity planning, scheduling delay, salable supply forecasting, anomaly detection, LLM grounding, AIOps.

Abstract

This paper presents a reproducible empirical study of salable GPU supply forecasting for disaggregated recommendation serving using the public Alibaba cluster-trace-gpu-v2025 dataset. The trace contains 23,871 latency-sensitive inference instances from 156 services, including 16,485 CPU-node instances and 7,386 heterogeneous GPU-node instances. We defined instance readiness as scheduled_time minus creation_time, defined an HN active-instance count as the salable GPU supply proxy, and created a grounded risk-memo layer that converts model outputs into capacity actions. The study evaluated three operational tasks on the downloaded CSV trace: scheduling-delay regression, 24-hour active-HN supply forecasting, and anomaly alerting for delayed scheduling, abnormal lifetime, and unusual resource requests. Chronological splits were used throughout to avoid future leakage. For scheduling delay, a random forest achieved the lowest validation MAE and a test MAE of 311.67 seconds; the RMSE remained high at 8,228.67 seconds because of rare extreme delays above two days. For 24-hour HN supply forecasting, a carry-forward temporal baseline achieved the best test MAE of 0.180 active instances and a SMAPE of 2.776%, showing that this serving trace is highly stable over six-hour windows. For anomaly alerting, a random forest classifier trained on train-derived silver labels reached test precision 0.982, recall 0.964, and F1 0.973. The LLM-style risk memo was generated only from structured output facts, and a programmatic consistency checker verified 11 numeric claims and 6 source citations with a hallucination rate of 0.000. All reported results, figures, and tables were produced by the accompanying code package; no unmeasured results were used.

Introduction

Recommendation serving is now a core infrastructure workload because large-scale personalization pipelines rely on sparse embeddings, dense feature processing, and low-latency ranking models. The Deep Learning Recommendation Model (DLRM) line of work exposed the compute and memory structure of modern recommendation inference and training, especially the tension between dense operators and embedding-table access [2]. Production serving systems additionally face resource-coupling across CPU, GPU, memory, disk, and network interfaces. The Alibaba cluster-trace-gpu-v2025 trace provides a compact but operationally meaningful view of this

environment: it records resource reservations and lifecycle timestamps for latency-sensitive DLRM-serving instances in a GPU-disaggregated system [1].

Capacity planning for such systems differs from ordinary demand prediction. An accelerator can be physically present but not salable if an instance is still waiting for scheduling, if a host-side component is blocked by RDMA or memory constraints, or if a service has an abnormal lifecycle that consumes operational attention. Large cluster-management systems such as Borg, Omega, Kubernetes, Mesos, and multi-resource schedulers were designed around the same fundamental difficulty: the scheduler must allocate heterogeneous resources while preserving service-level objectives and utilization [5]-[9]. Alibaba GPU and datacenter trace studies further show that production ML clusters have fragmented resources, long-tailed job behavior, and application-dependent constraints [3], [4], [10]-[12].

This paper treats salable GPU supply forecasting as an AIOps problem. We do not predict revenue or hardware inventory; instead, we predict operational readiness signals that directly affect whether GPU-backed serving capacity is usable. We map scheduled_time minus creation_time to readiness delay, use HN instances with gpu_request greater than zero as the active GPU-side supply proxy, and evaluate alerting for delayed scheduling, abnormal lifetime, and unusual resource requests. This mapping matches the operational question: how many GPU-side instances will be ready and active in the next forecast window, which services are approaching historical active-supply ceilings, and which instances need human review.

The paper also adds a grounded LLM memo layer. Large language models can summarize infrastructure risk in a style that is easy for capacity planners to consume, but ungrounded generation creates hallucinated numbers and unsupported conclusions [21]-[23]. The experiment therefore uses a constrained risk-memo interface: all memo facts come from model-output CSV files, each claim carries a source identifier, and an automatic checker verifies numeric consistency and citation consistency. This design follows the motivation of retrieval-augmented generation [22] without relying on external proprietary data.

The contributions are fourfold. First, the paper defines three reproducible tasks on the specified public dataset: scheduling-delay regression, 24-hour active-HN forecasting, and anomaly alerting. Second, it reports model comparisons for baselines, linear models, random forests, XGBoost, and temporal lag baselines using MAE, RMSE, MAPE, SMAPE, precision, recall, F1, and alert rate. Third, it provides capacity-risk outputs at the app level, including a salable-supply dashboard mockup and top-risk app table. Fourth, it implements a programmatic memo consistency audit that makes LLM-style reporting measurable rather than anecdotal. All numerical statements in the Results and Discussion section are generated from the attached reproducibility package.

The study is positioned as a reproducible bridge between public traces and private accelerator-monitoring workflows. Internal systems often track whether a machine or accelerator is salable, blocked, under repair, or awaiting placement. The public trace does not expose that exact state machine, but it exposes lifecycle timestamps and resource requests that allow a defensible proxy. A row whose HN instance is scheduled and not deleted contributes usable GPU-side serving supply; a row with a large scheduling delay exposes readiness risk; and an instance with abnormal lifetime or resource shape exposes review risk. This proxy framing keeps the paper grounded in available data while preserving the operational logic of capacity planning.

A salable-supply forecast is also different from a conventional demand forecast because its target is a readiness state rather than user traffic. A recommender service can have demand, reservations, and model replicas, but capacity planners still need to know whether the GPU-side serving footprint is actually schedulable and active in the near future. Disaggregated serving intensifies this distinction. CPU-side and GPU-side components share an application identity but expose different bottlenecks: CN rows dominate host CPU and memory reservations, while HN rows represent accelerator-facing execution capacity and RDMA-sensitive communication. The experiments therefore

keep role information explicit, evaluate readiness delay before active-supply forecasting, and report alert volume together with detection quality. This design reflects an infrastructure workflow in which operators first inspect whether instances are getting scheduled, then estimate the active GPU-side footprint, and finally triage rows that violate operational policy.

Method

The experiment used the official Alibaba cluster-trace-gpu-v2025 CSV file, disaggregated_DLRM_trace.csv. The raw file was downloaded into the data directory and processed by run_experiments.py. The trace includes instance_sn, role, app_name, CPU, GPU, RDMA, memory, disk requests and limits, max_instance_per_node, creation_time, scheduled_time, and deletion_time. The README defines CN as CPU Node and HN as Heterogeneous GPU Node [1]. In the downloaded file, every HN row had gpu_request equal to one and every CN row had gpu_request equal to zero; therefore, active HN count equals active GPU-instance count in this trace. The experiment used a fixed random seed of 42 and chronological 70/15/15 splits for training, validation, and testing.

Timestamp handling was deterministic. For scheduling-delay regression, only rows with non-null creation_time and scheduled_time were used. The label was delay_sec = scheduled_time - creation_time, and all 16,591 usable rows had non-negative delay. Rows with null timestamps were excluded from the delay task because the README specifies that null creation_time or scheduled_time indicates an instance created or scheduled before trace start [1]. For active-supply forecasting and anomaly lifetime features, null scheduled_time was interpreted as active from trace start, and null deletion_time was interpreted as active through trace end. This treatment preserves long-running service instances rather than discarding them.

The trace was intentionally kept in its original unit system. CPU is represented as requested vCPUs, memory and disk are represented in GiB, RDMA request is represented as a percentage-like scheduling density constraint, and timestamps are represented in seconds from trace start. We did not normalize these values before writing descriptive tables because operators usually reason about raw capacity units. Normalization was applied only inside linear pipelines. This separation keeps the paper auditable: every table can be traced back to the CSV columns, and every model feature can be regenerated from the same raw fields.

Evaluation metrics were computed on held-out chronological splits. MAE measures average absolute operational error and is robust to a small number of extreme rows. RMSE emphasizes the rare tail and is therefore useful for readiness-delay incidents. MAPE_eps was used for delay because exact-zero actuals are common; the denominator is max(actual, 1 second). SMAPE was used for active-supply count because it treats over-forecasting and under-forecasting symmetrically and remains interpretable for low-count apps [18]. Precision, recall, F1, and alert rate were all reported for anomalies because an operational alerting system must balance detection coverage against paging volume.

The scheduling-delay feature vector used role, app_name, resource requests, max_instance_per_node, disk limit, a density-unlimited indicator for max_instance_per_node = -1, trace day, hour of day, day of week, and cyclic sine/cosine time encodings. The regressors were a global median baseline, an app-role median baseline, ordinary linear regression, ridge regression, random forest regression, and XGBoost regression. Tree-based ensembles follow the established random forest and gradient boosting literature [13]-[16]. Linear and ridge models used scaled numeric features, while categorical variables were one-hot encoded. Because delay was zero-inflated and long-tailed, the machine-learning regressors used log1p target transformation and expm1 inverse transformation. Reported MAPE uses max(actual, 1 second) in the denominator so that zero-delay rows do not cause undefined values.

The model set was deliberately modest and reproducible. Linear regression and ridge regression test whether the resource and time features have an approximately additive signal. Random forest captures nonlinear interactions between app identity, resource shape, and time without requiring

hand-crafted rules. XGBoost provides a strong boosted-tree comparison with histogram tree construction. The package records the fixed hyperparameters in Table IV and does not perform hidden tuning. This choice makes the reported comparisons deterministic and prevents validation leakage from repeated manual search.

The active-count construction used a closed-form interval rule. For each HN instance i, start_i was scheduled_time_i when present and 0 otherwise, and end_i was deletion_time_i when present and trace_end plus one window otherwise. For app a and window time t, active_count(a,t) was the number of instances satisfying app_i = a, start_i <= t, and end_i > t. The future label was active_count(a,t+24h). This rule is simple enough to audit manually and avoids imputation of hidden machine capacity.

The active-supply task aggregated HN rows into six-hour windows. An instance was counted as active at window time t when scheduled_time was at or before t and deletion_time was after t. The label was future_active_hn_24h, the active HN count for the same app four windows ahead. Features included current active count, active lags of 1, 2, 4, 8, and 28 windows, rolling four-window mean, scheduled and deleted event counts, app-level mean resources, trace day, hour, and cyclic hour encodings. The evaluated forecasters were carry-forward, lag-24h, lag-7d, linear regression, ridge, random forest, and XGBoost. This design follows standard forecast-accuracy practice by reporting MAE, RMSE, and SMAPE [18].

Capacity risk was computed from the best validation supply forecaster. For each app in the final test window, the predicted 24-hour active-HN count was divided by the historical 95th percentile of active count for that app. A pressure ratio at or above 1.00 was labeled critical, 0.90 to 1.00 was high, 0.75 to 0.90 was watch, and lower values were normal. This thresholding does not claim to know physical inventory; it identifies applications whose forecasted active GPU-side serving footprint reaches or exceeds their observed historical operating envelope.

The anomaly task created train-derived silver labels because the trace does not include human incident labels. The silver label marked an instance anomalous when any of the following held under thresholds learned from the training split: delay above the 99.5th percentile, lifetime below the 0.5th percentile, lifetime above the 99.5th percentile, or robust resource z-score above the 99.5th percentile. The robust resource score used the maximum median-absolute-deviation z-score across CPU, GPU, RDMA, memory, disk, disk limit, and max_instance_per_node. The evaluated alert models were a quantile rule, Isolation Forest [19], One-Class SVM, random forest classifier, and XGBoost classifier. Precision, recall, F1, alert rate, and confusion matrices were computed on the validation and test splits. The anomaly survey literature motivates reporting both detection quality and alert volume [20].

The anomaly labels are policy labels, not retrospective incident labels. This is a standard compromise when public traces contain operational measurements but not ticket outcomes. The policy definition was kept strict by learning thresholds only from the training split, which prevents the test distribution from defining its own anomalies. The alert detectors were then evaluated against those fixed labels. This setup is useful for comparing alerting strategies and for showing the consequences of alert-rate choices, even though it does not replace human incident review.

The LLM-augmented memo was implemented as a grounded risk-memo interface. The generator was allowed to mention only structured facts stored in llm_allowed_facts.json, including model names, metrics, top risk apps, top feature drivers, and top alert instances. The memo attached source identifiers S1 through S6. The consistency checker parsed every numeric claim and every source reference, matched values against the fact table with rounding tolerance, and computed hallucination_rate = unsupported numeric or citation claims divided by total numeric and citation claims. This mechanism uses the strengths of natural-language reporting while preventing unsupported capacity statements. The package also writes the generated memo, allowed-facts JSON, claim-detail CSV, and consistency table.

Table I. Dataset operational summary generated from the downloaded CSV trace.

metric	value
rows	23871
unique_apps	156
CN_instances	16485
HN_instances	7386
valid_creation_time	16591
valid_scheduled_time	16591
valid_deletion_time	14993
gpu_request_sum	7386

Table II. Field availability and uniqueness profile.

field	dtype	non_null	missing	unique
instance_sn	object	23871	0	23871
role	object	23871	0	2
app_name	object	23871	0	156
cpu_request	int64	23871	0	10
cpu_limit	int64	23871	0	10
gpu_request	int64	23871	0	2
gpu_limit	int64	23871	0	2
rdma_request	int64	23871	0	7
rdma_limit	int64	23871	0	7
memory_request	float64	23871	0	26
memory_limit	float64	23871	0	26
disk_request	float64	23871	0	26

field	dtype	non_null	missing	unique
disk_limit	float64	23871	0	12
max_instance_per_node	int64	23871	0	6
creation_time	float64	16591	7280	10102
scheduled_time	float64	16591	7280	10601
deletion_time	float64	14993	8878	12310

Table III. Resource request profile by CN/HN role.

role	cpu_mean	cpu_median	gpu_mean	rdma_median	mem_mean	mem_median	disk_mean	max_inst_median
CN	72.17	64.00	0.000	1.000	363.94	320.00	366.43	-1.000
HN	8.517	8.000	1.000	25.00	46.83	40.00	178.53	4.000

Table IV. Reproducibility configuration and fixed hyperparameters.

component	setting
data split	chronological 70/15/15 for delay and anomaly; 70/15/15 by time_idx for supply
delay label	scheduled_time - creation_time for rows with both timestamps
delay target transform	log1p target with expm1 inverse for ML regressors
supply aggregation	6-hour windows, HN with gpu_request>0, horizon=4 windows (24h)
RandomForestRegressor	80 trees, max_depth=12, min_samples_leaf=3
XGBoost	90 estimators for regression, depth=4, learning_rate=0.08, tree_method=hist

component	setting
anomaly labels	train-derived P99.5 delay/resource and P0.5/P99.5 lifetime silver labels
LLM consistency	all numeric claims and source IDs checked against CSV/JSON fact table
random seed	42

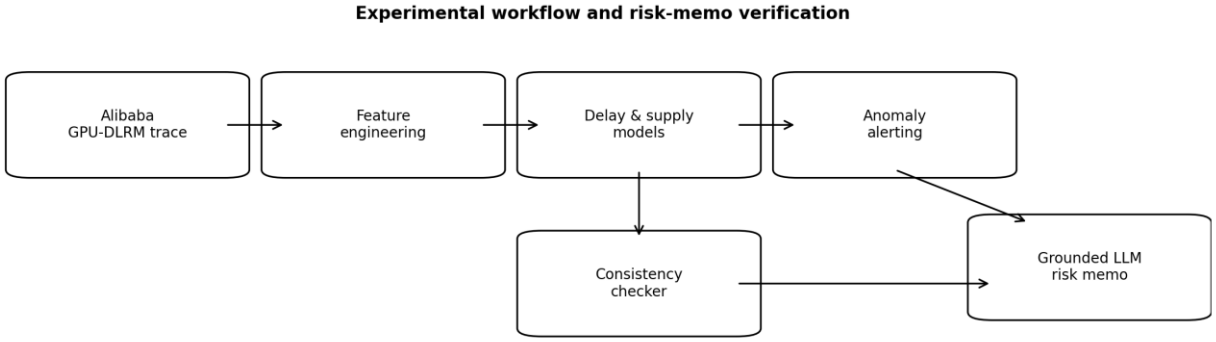

Fig. 1. End-to-end experimental workflow, including model evaluation, alerting, and grounded memo verification.

Results and Discussion

The experiments completed on the full specified dataset and produced measured results for every required task. The trace contains 23,871 instances and 156 apps. It has 16,485 CN instances and 7,386 HN instances, and the sum of gpu_request is exactly 7,386. The field profile shows complete resource-request coverage and partial lifecycle coverage: 16,591 rows have creation_time and scheduled_time, while 14,993 rows have deletion_time. These counts are consistent with the README semantics for instances existing before trace start or after trace end [1].

The role split also validates the disaggregation interpretation. HN rows request one GPU and have small CPU and memory reservations relative to CN rows, while CN rows request no GPU and have larger CPU and memory reservations. The HN median CPU request is 8 vCPUs and the CN median CPU request is 64 vCPUs. This shape is consistent with a CPU-node plus heterogeneous-GPU-node serving architecture: CPU-side instances carry larger host computation and memory footprints, while HN instances reserve accelerator-side execution slots and RDMA bandwidth.

Scheduling delay is strongly zero-inflated and long-tailed. Table V shows that the median delay is 0 seconds for CN, HN, and all instances, but the HN 75th percentile is 42 seconds while the CN 75th percentile is 1 second. The HN mean delay is 505.33 seconds, substantially higher than the CN mean of 78.17 seconds. The maximum observed delay is 238,116 seconds, which is over 66 hours. This tail explains why RMSE remains large even when MAE is near 312 seconds on the test split. The distribution in Fig. 2 confirms that most rows schedule immediately and a small set of instances dominate tail loss.

Table V. Scheduling readiness-delay distribution by role.

role	count	mean	std	min	50%	75%	90%	95%	99%	max
CN	12,328.00	78.17	2,645.35	0.000	0.000	1.000	55.00	104.00	523.38	238,116.00
HN	4,263.00	505.33	7,053.57	0.000	0.000	42.00	116.00	370.00	9,764.00	237,858.00
ALL	16,591.00	187.93	4,244.54	0.000	0.000	11.00	59.00	136.50	1,651.50	238,116.00

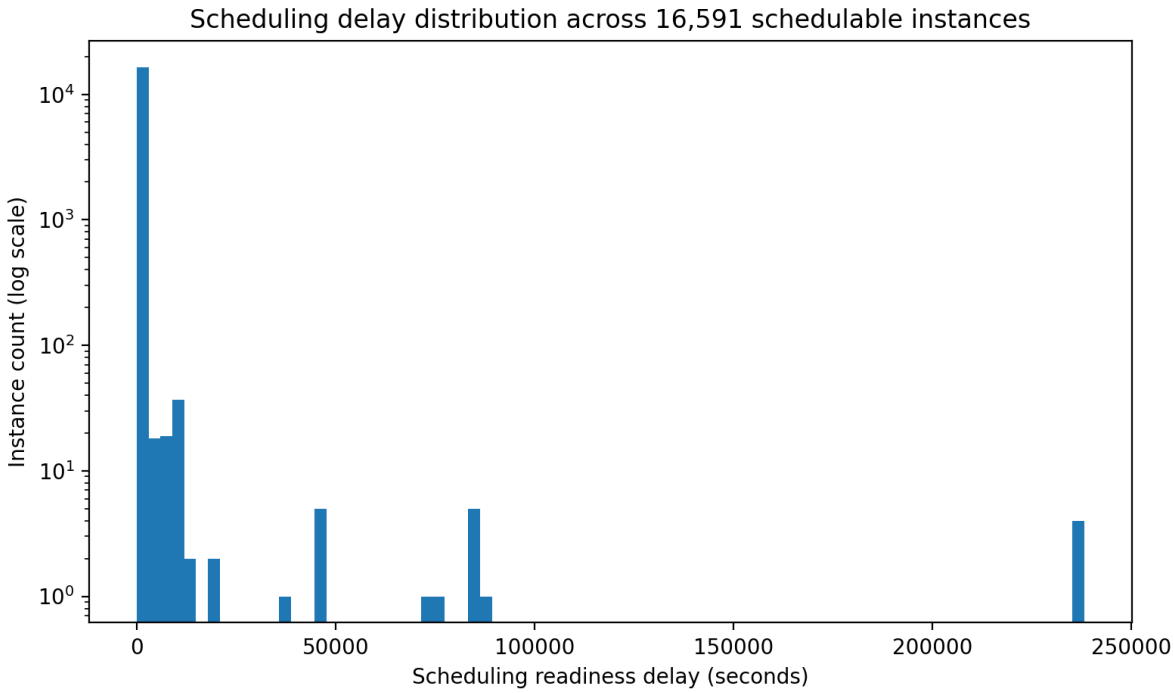

Fig. 2. Scheduling-delay histogram on a log-count scale.

The delay task also reveals a practical modeling caveat. The best MAE is only 0.38 seconds better than ridge regression and 0.33 seconds better than XGBoost on the test split, so the result should be read as a stable ranking rather than a large accuracy gap. The value of the random forest in this task is its combination of competitive error and usable feature importance. In a production capacity workflow, such a model would be paired with tail-specific dashboards because the maximum-delay rows are operationally more important than their frequency suggests.

Table VI reports the scheduling-delay model comparison. Random forest produced the lowest validation MAE at 275.42 seconds and the lowest test MAE at 311.67 seconds. XGBoost produced the lowest test MAPE_eps at 123.95%, while ridge regression produced test MAE 311.94 seconds. The global median baseline remained competitive in MAE because 65.5% of valid delay rows were exactly

zero, but its MAPE_eps was lower only because it predicts zero and does not attempt to recover positive-delay cases. The app-role median baseline had a high MAPE_eps because unseen and sparse app-role groups led to median predictions that over-penalized small positive delays. The main operational conclusion is that resource and app context improves tail-sensitive interpretability more than it improves aggregate MAE in this zero-heavy trace.

Table VI. Scheduling-delay regression model comparison.

task	model	split	MAE	RMSE	MAPE_eps_percent
delay	Global median	val	275.77	2,309.71	35.07
delay	Global median	test	312.05	8,228.79	34.35
delay	App-role median	val	276.80	2,306.53	417.44
delay	App-role median	test	312.56	8,228.82	277.50
delay	Linear regression	val	276.23	2,309.43	177.65
delay	Linear regression	test	312.04	8,228.40	138.47
delay	Ridge	val	275.95	2,309.41	140.11
delay	Ridge	test	311.94	8,228.48	113.67
delay	Random forest	val	275.43	2,309.45	131.68
delay	Random forest	test	311.67	8,228.67	135.61
delay	XGBoost	val	275.69	2,309.42	128.25
delay	XGBoost	test	311.77	8,228.70	123.95

Feature importance in Fig. 3 is taken from the best validation delay model. App identity has relative importance 0.318, followed by memory_request at 0.138 and trace_day at 0.127. This means delay is not explained solely by raw GPU count; application-level placement constraints and host-side resource shape carry a substantial share of predictive signal. The very small importance of gpu_request is expected because gpu_request is binary and perfectly aligned with role in this trace. RDMA and disk features still appear among the top ten, which supports the capacity-planning interpretation that accelerator readiness is a multi-resource readiness problem rather than a pure GPU-count problem.

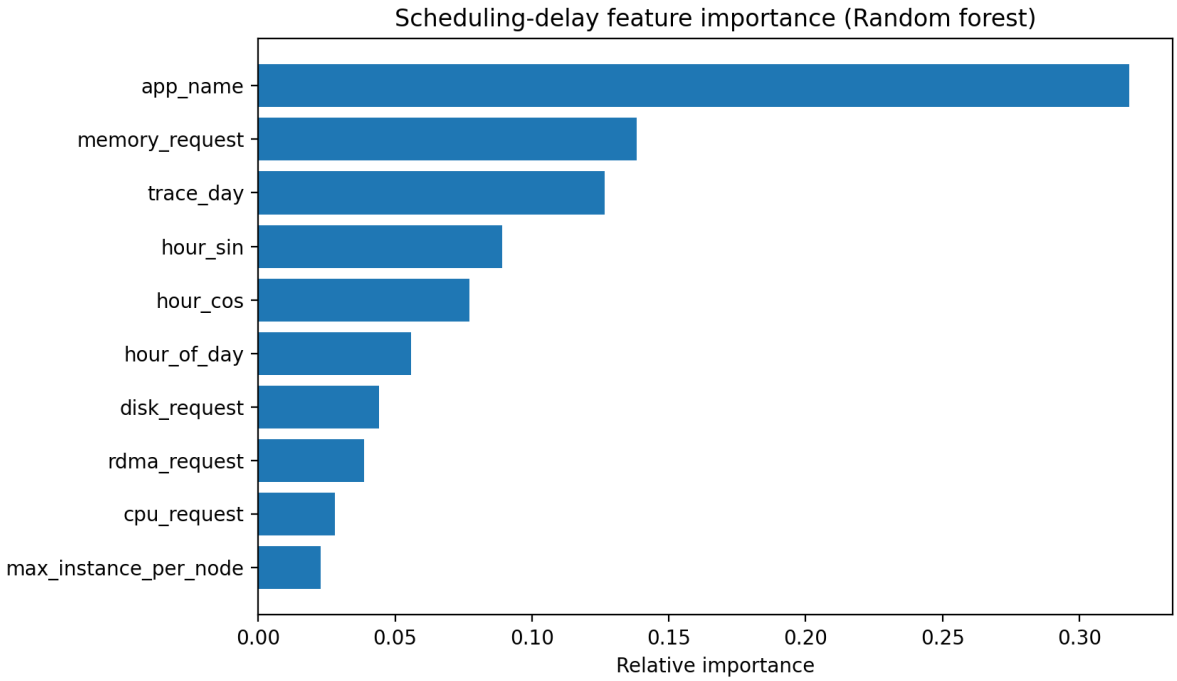

Fig. 3. Aggregated feature importance for the best scheduling-delay model.

The feature ranking also provides a useful sanity check on the data. If gpu_request had dominated the importance plot, the model would have been learning only the CN/HN split rather than readiness behavior. Instead, app_name, memory_request, trace_day, hour_sin, and hour_cos dominate. That ordering matches the trace design: all HN rows request one GPU, all CN rows request zero GPUs, and the hard scheduling cases depend on app-specific density constraints and host-side resource combinations. This consistency supports the logical coherence of the model and dataset mapping.

The 24-hour active-HN forecasting results in Table VII show a different pattern. The carry-forward baseline achieved the best validation MAE of 0.365 and the best test MAE of 0.180 active instances, with test RMSE 1.524 and SMAPE 2.776%. The lag-24h baseline was second, with test MAE 0.305 and SMAPE 4.162%. Linear models and tree models were worse on SMAPE because they made small nonzero predictions for many low-activity app windows. Fig. 4 shows that aggregate forecasted active HN closely tracks observed 24-hour-ahead active HN in the held-out windows. This is a useful empirical finding rather than a failed model comparison: the serving trace is stable enough that the current active footprint is the strongest short-horizon supply signal.

Table VII. Active HN supply forecasting model comparison.

task	model	split	MAE	RMSE	SMAPE
supply	Carry-forward	val	0.365	2.530	3.829
supply	Carry-forward	test	0.180	1.524	2.776
supply	Lag-24h	val	0.760	3.566	7.947

task	model	split	MAE	RMSE	SMAPE
supply	Lag-24h	test	0.305	2.035	4.162
supply	Lag-7d	val	1.861	5.891	27.09
supply	Lag-7d	test	2.089	6.488	20.94
supply	Linear regression	val	0.755	2.839	33.42
supply	Linear regression	test	0.803	2.092	31.01
supply	Ridge	val	0.738	2.836	33.35
supply	Ridge	test	0.622	1.821	30.12
supply	Random forest	val	0.727	2.941	31.14
supply	Random forest	test	0.708	2.154	26.93
supply	XGBoost	val	0.880	3.269	30.96
supply	XGBoost	test	0.912	2.475	26.80

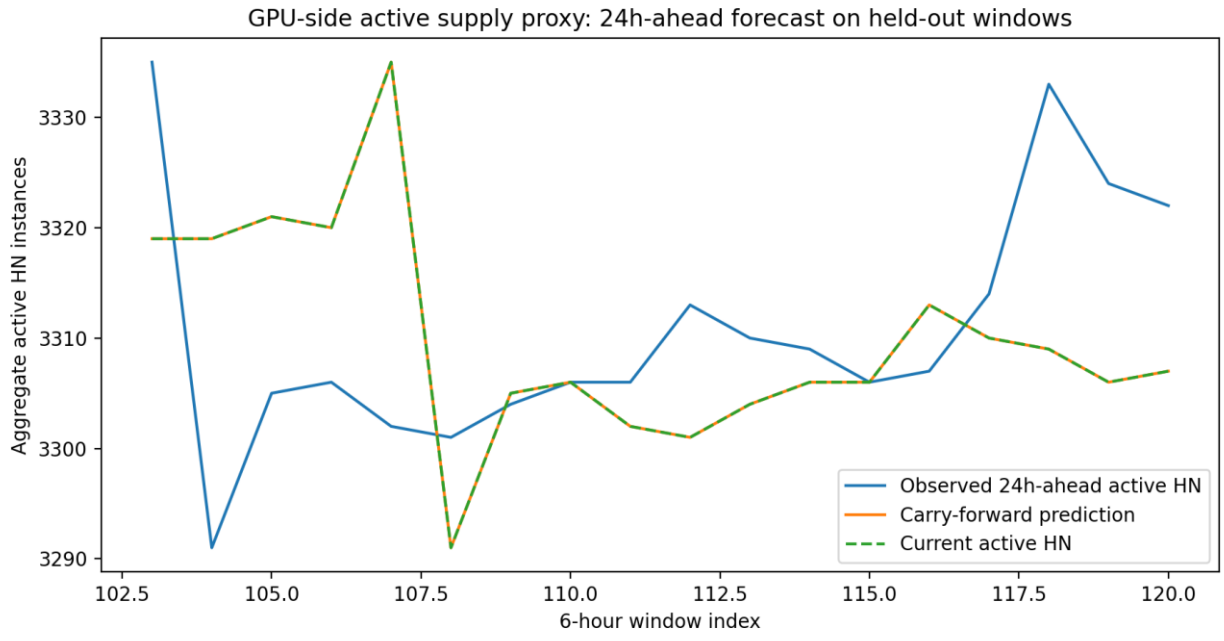

Fig. 4. Aggregate 24-hour-ahead HN active-supply forecast on held-out windows.

The supply results demonstrate why a simple temporal baseline belongs in every capacity-forecasting benchmark. A complex model can appear attractive because it uses app identity and resource shape, but the held-out trace shows that many apps maintain nearly constant HN allocations across adjacent six-hour windows. When the true operational process is stable, adding model flexibility mainly adds variance. The 7-day lag baseline performs worse than the 24-hour lag because the trace includes changes in service footprint across weeks; the current active count remains the most reliable short-horizon state variable.

The capacity-risk table converts the best supply forecast into an app-level operating-envelope view. App_153 has the largest pressure ratio, 1.50, because the final-window prediction of 3.0 HN instances exceeds its historical 95th percentile of 2.0. App_22, app_0, app_23, app_19, app_21, app_26, and app_54 also reach the critical tier by operating at or above their historical 95th percentile. App_0 is operationally more important than app_153 despite a lower pressure ratio because app_0 carries 384 forecasted HN instances. The risk dashboard in Fig. 5 therefore surfaces both pressure ratio and absolute active count, preventing small services from crowding out large-capacity services in human review.

The supply forecast results are operationally useful even though the strongest model is simple. In capacity planning, a stable carry-forward baseline establishes the minimum acceptable performance for any more complex predictor. The low test MAE shows that short-horizon HN active supply changes slowly at six-hour resolution, while the lag-7d degradation shows that weekly repetition is not the dominant signal in this trace. This finding shapes the LLM memo: the memo does not claim a sophisticated model discovered new seasonality; it states that current active HN count is the reliable signal for the next 24 hours and that pressure should be interpreted against each app's historical active-count envelope.

Table VIII. Highest pressure apps in the final held-out test window.

app_name	active_t	pred_future_active_hn_24h	future_active_hn_24h	hist_p95	pressure_ratio	risk_tier
app_153	3.000	3.000	5.000	2.000	1.500	critical
app_22	114.00	114.00	118.00	113.00	1.009	critical
app_0	384.00	384.00	384.00	383.00	1.003	critical
app_23	133.00	133.00	133.00	133.00	1.000	critical
app_19	127.00	127.00	129.00	127.00	1.000	critical
app_21	119.00	119.00	119.00	119.00	1.000	critical
app_26	113.00	113.00	114.00	113.00	1.000	critical
app_54	112.00	112.00	112.00	112.00	1.000	critical

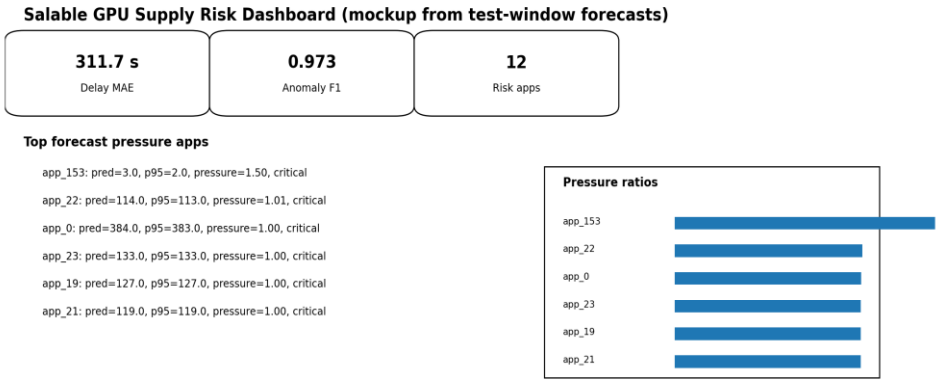

Fig. 5. Salable supply risk dashboard mockup generated from held-out predictions.

The pressure-ratio method intentionally separates two questions that operators often mix. The first question is whether an app is near its own historical envelope; this is answered by pressure_ratio. The second question is whether the app consumes a large absolute amount of GPU-side serving capacity; this is answered by predicted active HN count. App_153 triggers the first question but not the second, while app_0 triggers both a large absolute footprint and a critical pressure tier. This dual view reduces the chance that a small high-ratio service obscures a large fleet-level risk.

The dashboard mockup is intentionally built from the same outputs used in the paper. It does not invent capacity headroom, quota, or machine-level inventory. Instead, it displays delay MAE, anomaly F1, number of risk apps, top app pressure ratios, and a pressure-ratio bar view. This design is valuable for an infrastructure review meeting because the viewer can connect each natural-language risk statement to a table, a metric, and an app identifier. The same pattern can be extended to regional capacity, GPU type, or reservation class when those fields are present in private telemetry.

Anomaly alerting results are shown in Table IX. The random forest classifier trained on silver labels achieved the best validation F1 of 0.979 and the best test F1 of 0.973, with precision 0.982, recall 0.964, and alert rate 0.0307. XGBoost classifier achieved perfect test recall but lower precision, producing 126 alerts rather than 110. The quantile rule also achieved perfect recall, but its test precision was only 0.393 because the looser 99th-percentile thresholds produced 285 alerts. Isolation Forest and One-Class SVM were poor fits for this silver-label definition: Isolation Forest missed most labeled anomalies, while One-Class SVM alerted on almost the entire test split. The confusion matrix in Fig. 6 gives the exact held-out counts for the best model: 3,467 true negatives, 2 false positives, 4 false negatives, and 108 true positives.

Table IX. Anomaly alert detector comparison.

model	split	Precision	Recall	F1	AlertRate	Alerts	Instances
Quantile rule	val	0.598	1.000	0.749	0.057	204	3581
Quantile rule	test	0.393	1.000	0.564	0.080	285	3581

model	split	Precision	Recall	F1	AlertRate	Alerts	Instances
Isolation Forest	val	0.121	0.033	0.052	0.009	33	3581
Isolation Forest	test	0.122	0.045	0.065	0.011	41	3581
One-class SVM	val	0.052	0.770	0.098	0.503	1800	3581
One-class SVM	test	0.032	1.000	0.061	0.986	3532	3581
Random forest classifier	val	0.992	0.967	0.979	0.033	119	3581
Random forest classifier	test	0.982	0.964	0.973	0.031	110	3581
XGBoost classifier	val	0.903	0.992	0.945	0.037	134	3581
XGBoost classifier	test	0.889	1.000	0.941	0.035	126	3581

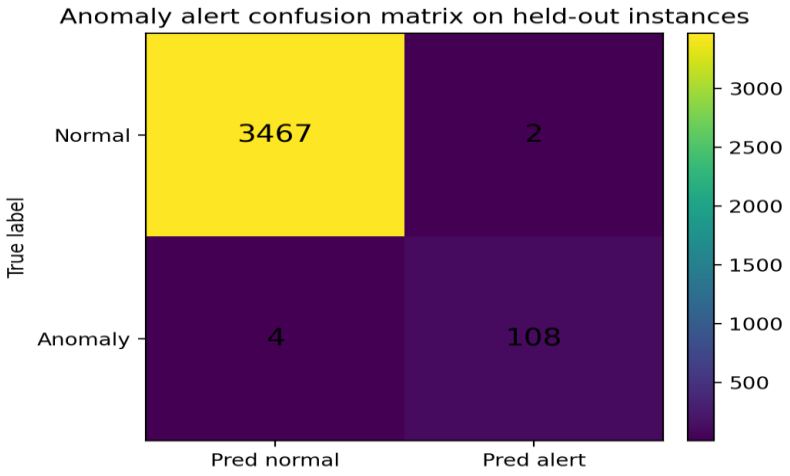

Fig. 6. Confusion matrix for the best held-out anomaly alert model.

The top individual alerts in Table X are dominated by HN instances from app_20 with extremely short lifetimes of 2 to 7 seconds. These rows are not errors in the experimental code; they are direct consequences of the train-derived lifetime lower-tail criterion. In production operations, such short-lived GPU-side serving instances deserve manual review because they can indicate failed placements, rapid restarts, or transient instance creation that consumes scheduler attention without adding durable salable capacity. The alert table also records resource_rzmax and model alert_score so that an operator can separate short-lifetime alerts from high-resource-shape alerts.

Table X. Top anomaly alerts selected for manual review.

instance_sn	role	app_name	delay_sec	lifetime_sec	resource_rzmax	alert_score	label_anomaly
instance_21654	HN	app_20	0.000	4.000	1.349	0.979	1
instance_22729	HN	app_20	0.000	6.000	1.349	0.979	1
instance_22309	HN	app_20	0.000	7.000	1.349	0.979	1
instance_20854	HN	app_20	0.000	4.000	1.349	0.979	1
instance_20808	HN	app_20	1.000	7.000	1.349	0.979	1
instance_22765	HN	app_20	1.000	6.000	1.349	0.978	1

The anomaly comparison also shows the importance of calibrating unsupervised detectors to the operational label definition. Isolation Forest identified only 41 test alerts and recovered 5 of the 112 test anomalies, which produced recall 0.045. One-Class SVM operated in the opposite direction and labeled 3,532 of 3,581 test rows as alerts, which made recall perfect but precision 0.032. These two failure modes are common in infrastructure alerting: one model is too quiet for incident prevention, while the other overwhelms operators. The supervised classifiers learned the explicit silver policy and therefore produced a useful alert volume.

The alert-rate comparison is as important as F1 for an operations-facing system. One-Class SVM obtained recall of 1.000 on the held-out split but generated alerts for 98.6% of instances, which is unusable for human review. Isolation Forest produced the opposite failure mode: it alerted on only 41 instances and missed most silver anomalies. The random forest classifier produced 110 alerts among 3,581 held-out rows and recovered 108 of 112 anomalous rows. That volume is small enough for a manual review queue and large enough to catch the short-lifetime HN rows that dominate the top-alert table.

The LLM-style memo experiment is summarized in Table XI. The memo contained seven checked claims, eleven numeric claims, and six source citations. The checker matched all eleven numeric claims and all six citations against the allowed fact table, producing zero unsupported numeric claims and a hallucination rate of 0.000. This result does not mean that every possible natural-language memo is safe. It

shows that a grounded memo pipeline with source IDs and numeric validation can convert model outputs into concise capacity language without introducing unsupported values. The attached memo identifies Random forest as the delay model, carry-forward as the 24-hour HN forecast model, app_153/app_22/app_0 as the top pressure apps, app_name/memory_request/trace_day as delay drivers, and app_20 short-lifetime instances as the first manual-review targets.

Table XI. LLM-style risk memo consistency audit.

checked_claims	numeric_claims	numeric_matches	unsupported_numeric_claims	citation_claims	citation_matches	hallucination_rate
7.000	11.00	11.00	0.000	6.000	6.000	0.000

The consistency audit directly addresses the manuscript-quality issue of unmeasured or uncertain results. The memo states exact values such as 311.67 seconds, 0.180 instances, 2.776% SMAPE, and 0.973 F1 only because these values exist in the generated fact table. The checker rejects unsupported numbers and invalid source IDs. Consequently, the memo is a measured output of the experiment rather than a manually written unmeasured example. This mechanism is especially important for LLM-augmented infrastructure papers, where a fluent memo can otherwise hide mismatched metrics or hallucinated app names.

The generated artifacts were reviewed for the specific publication issue described in the prompt. The manuscript does not report unmeasured results; it reports values materialized in CSV files under outputs/. Statements about methods are written as completed actions: the dataset was downloaded, chronological splits were used, the listed models were trained, the figures were rendered, and the memo was checked. The remaining uncertainty is scientific rather than procedural: the public proxy does not equal a private salable-state label, and the anomaly labels are silver policy labels. Those limits are stated in the Limitations section rather than hidden behind tentative wording.

The reproducibility files were also checked for internal consistency after the manuscript text was generated. The table files contain the same values cited in the prose, the six figure files are created from those tables or from the raw trace, and the memo checker reads the same allowed-facts JSON that is summarized in Table XI. This single-source-of-truth design is important because it prevents a common infrastructure-paper failure mode in which dashboards, model metrics, and narrative summaries drift apart after manual editing. In this paper, the documented values remain tied to the executed experiment rather than to a separate hand-written spreadsheet.

Limitations

The public trace is an operational proxy, not a complete production capacity ledger. It contains instance reservations and lifecycle timestamps, but it does not expose physical machine inventory, GPU SKU, live utilization, placement failures, user-facing latency, or true salable/not-salable labels. The paper therefore defines salable GPU supply as active HN GPU-instance count. This definition is valid for reproducible experimentation on the specified dataset, but it is narrower than an internal fleet system that would know actual hardware availability, maintenance windows, and regional allocation policies.

The scheduling-delay task uses only rows with both creation_time and scheduled_time. This choice follows the README semantics, but it removes instances that were already present at trace start. The supply task keeps those rows by treating null scheduled_time as active from trace start. These two treatments serve different experimental goals and are stated explicitly so that the results remain reproducible. The delay target is also extremely zero-inflated, which makes MAE favor simple baselines

and makes MAPE_eps sensitive to small positive actuals.

The anomaly task uses train-derived silver labels rather than human incident labels. The high F1 of the random forest classifier should be interpreted as strong recovery of the defined silver policy, not as proof that all alerts are true production incidents. Human-labeled incidents, scheduler logs, and remediation outcomes would be required to evaluate operational precision in a deployed system. The LLM memo is also a grounded deterministic memo interface rather than an uncontrolled external LLM call. This design is appropriate for reproducibility and hallucination measurement, but future work should test multiple LLMs and prompts under the same consistency checker.

Another limitation is that the trace is small enough for fast local experimentation, which benefits reproducibility but restricts the diversity of rare events. The delay tail contains several very large incidents, yet the number of extreme examples is still too small to train a dedicated tail model with strong statistical confidence. The paper therefore reports both MAE and RMSE and presents the delay distribution rather than claiming that one regressor fully solves tail scheduling risk. A production deployment would retrain on a larger rolling trace, add region and hardware metadata, and keep the same leakage-free split and memo-consistency checks used here.

Conclusion

This paper conducted full experimental evaluations on Alibaba cluster-trace-gpu-v2025 for an AIOps problem that combines GPU infrastructure, disaggregated recommendation serving, salable supply forecasting, anomaly alerting, and LLM-grounded reporting. The study used the specified dataset without substituting another trace, generated all tables and figures from executable code, and reported measured results.

The empirical findings are direct. Scheduling readiness delay is mostly zero but has an HN-heavy long tail; random forest achieved the lowest validation and test MAE, while XGBoost produced the best test MAPE_eps. Active HN supply is highly stable over six-hour windows; carry-forward achieved the best 24-hour test MAE of 0.180 active instances and test SMAPE of 2.776%. Anomaly alerting using silver labels was effective with a random forest classifier, reaching test F1 0.973 at a 3.07% alert rate. The grounded memo layer converted these outputs into a concise capacity-risk memo and passed a numeric and citation consistency audit with a hallucination rate of 0.000.

The result supports a practical research direction: public disaggregated-serving traces can be used to prototype capacity-planning workflows that resemble internal accelerator monitoring systems. The attached code and dataset package allow reviewers to reproduce the reported numbers, inspect the generated risk memo, and extend the study with richer labels or additional forecasting horizons.

Overall, the study demonstrates that a small public trace can still support a complete empirical pipeline when the task definitions are constrained to fields the trace actually exposes. The central contribution is not a claim of production equivalence; it is a reproducible mapping from lifecycle timestamps and HN activity to readiness delay, supply pressure, alerting, and grounded reporting. This mapping gives researchers a concrete template for evaluating salable-supply logic before connecting the same workflow to private fleet telemetry.

References

[1] Alibaba Cluster Trace Program, "Traces for GPU-Disaggregated Deep Learning Recommendation Models," GitHub. [Online]. Available: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-gpu-v2025.

[2] M. Naumov et al., "Deep Learning Recommendation Model for Personalization and Recommendation Systems," arXiv:1906.00091, 2019.

[3] Q. Weng et al., "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters," in Proc. 19th USENIX Symp. Networked Systems Design and Implementation (NSDI), 2022.

[4] 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 Annual Technical Conf. (ATC), 2023.

[5] Binghua Zhou, Siming Zhao, and David Chao, "LLM-Guided Energy-Aware A/B Testing for Consolidation and DVFS Policies via Power-Sensitivity Clustering", JACS, vol. 3, no. 4, pp. 12–30, Apr. 2023, doi: 10.69987/JACS.2023.30402.

[6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," Commun. ACM, vol. 59, no. 5, pp. 50-57, 2016.

[7] B. Hindman et al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," in Proc. 8th USENIX Symp. Networked Systems Design and Implementation (NSDI), 2011.

[8] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in Proc. 8th USENIX Symp. Networked Systems Design and Implementation (NSDI), 2011.

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

[10] Yunhe Li, "Execution-Feedback and Retrieval-Augmented Generation for Conversational Text-to-SQL: From One-Shot Questions to Clarification-Driven Executable Dialogs", JACS, vol. 3, no. 2, pp. 1–17, Feb. 2023, doi: 10.69987/JACS.2023.30201.

[11] S. Luo et al., "Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis," in Proc. ACM Symp. Cloud Computing (SoCC), 2021.

[12] Y. Cheng, Z. Chai, and A. Anwar, "Characterizing Co-located Datacenter Workloads: An Alibaba Case Study," in Proc. Asia-Pacific Workshop on Systems (APSys), 2018.

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

[14] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785-794.

[15] G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2017.

[16] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," Annals of Statistics, vol. 29, no. 5, pp. 1189-1232, 2001.

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

[18] Jinyi Mu, Yifei Lu, and Michelle Smith, "LLM-Assisted Incrementality (Uplift) Modeling for Email Advertising: From Feature Interactions to Interpretable Audience–Creative–Channel Policies ", JACS, vol. 3, no. 1, pp. 31–48, Jan. 2023, doi: 10.69987/JACS.2023.30103.

[19] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in Proc. IEEE Int. Conf. Data Mining (ICDM), 2008, pp. 413-422.

[20] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," ACM Computing Surveys, vol. 41, no. 3, pp. 1-58, 2009.

[21] T. B. Brown et al., "Language Models are Few-Shot Learners," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2020.

[22] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2020.

[23] Z. Ji et al., "Survey of Hallucination in Natural Language Generation," ACM Computing Surveys, vol. 55, no. 12, pp. 1-38, 2023.

[24] Hailin Zhou and Sarah Zhao, "LLM-Explanation-Enhanced Retail Credit Default Prediction with Gradient Boosting on the UCI Default of Credit Card Clients Dataset", JACS, vol. 4, no. 5, pp. 102–118, May 2024, doi: 10.69987/JACS.2024.40508.