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

Risk-Bounded GPU Resource Oversubscription via Conformal Demand Envelopes in Production AI Clusters

*Siyu Chen [1], * Shilu He [1], Eddin Sun [2]*

[1] *Information Management, UIUC, IL, USA*
[1] *Mathematics, UW-Madison, WI, USA*
[2] *Data Science, Columbia University, NY, USA*
* *Corresponding Email: chensy010227@gmail.com*

DOI: 10.69987/JACS.2024.40509

Keywords

GPU clusters; oversubscription; conformal prediction; demand envelopes; resource scheduling; Alibaba cluster trace; risk-bounded admission control; AI infrastructure; explainable policy generation

Abstract

GPU inventory is expensive, bursty, and hard to expand quickly, so operators often want to sell or admit more work than a strict reservation policy allows. Unbounded oversubscription, however, turns short-term efficiency into SLO risk. This paper presents a risk-bounded oversubscription method based on conformal demand envelopes. We conducted a full empirical evaluation on the Alibaba cluster-trace-gpu-v2023 CSV data. The evaluation used 1,523 production nodes, 1,213 GPU nodes, 6,212 GPUs, 8,152 default pod records, and 3585 reconstructed hourly demand bins. Each pod request was converted into normalized GPU, CPU, and memory demand using only released fields: cpu_milli, memory_mib, num_gpu, gpu_milli, QoS, pod phase, creation time, scheduled time, and deletion time. A chronological train/calibration/test split of 2050/683/684 bins was used. At $\alpha = 0.05$, the conformal envelope achieved a measured joint violation rate of 0.000000 and a false-safe rate of 0.000000, while increasing protected inventory utilization from 0.003983 under conservative reservation to 0.332877. Average-demand oversubscription and linear quantile regression had measured joint violation rates of 0.555556 and 0.510234, respectively. The results show that a conformal controller can tighten capacity envelopes without replacing cluster schedulers or relying on unverifiable utilization counters. The paper also includes a deterministic LLM-facing explanation layer that translates the calibrated policy state into operator-readable admission guidance without changing numerical decisions.

Introduction

Large AI clusters combine high capital cost with volatile resource demand. A scheduler that admits only the exact resources requested by each pod protects capacity, but it also leaves inventory idle whenever requests are conservative, delayed, or temporally sparse. A scheduler that admits too much work raises utilization, but it can create demand spikes that exceed available GPU, CPU, or memory capacity. The central problem is therefore not merely to predict demand; it is to decide how much extra inventory can be admitted while keeping the probability of an unsafe hour below a stated risk level α.

Multi-resource cluster management has long balanced throughput, fairness, and isolation. Mesos introduced fine-grained resource sharing [1], Omega separated scheduling decisions from a shared cluster state [2], Borg demonstrated high-scale production cluster management [3], and Kubernetes inherited

practical lessons from Borg and Omega [4]. Fairness and packing policies such as DRF, Tetris, and Symphony showed that CPU, memory, and accelerator constraints must be considered jointly rather than independently [5]–[7]. These systems establish the operational setting for the present paper: an oversubscription controller must produce a simple capacity signal that can sit above an existing scheduler without destroying its placement and fairness semantics.

Deep learning clusters sharpen the problem because GPU demand is expensive and heterogeneous. Prior work used reinforcement learning for resource management [8], introspective scheduling for deep learning workloads [9], and GPU-aware scheduling policies that reduce job completion time and queuing delay [10]. Alibaba trace analyses further showed that public production traces are essential for reproducible evaluation of cluster-resource ideas [11]–[13]. The present study uses the Alibaba 2023 GPU trace release specified in the prompt and does not introduce synthetic jobs for the main evaluation. Every reported number is computed from the released node and pod CSVs, or from deterministic time-series features derived from those CSVs.

Oversubscription needs a risk statement. A mean forecast tells an operator what usually happens, but it does not define how often the policy is allowed to be wrong. Quantile regression estimates an upper conditional quantile [18], and tree and boosting models provide flexible predictors for non-linear resource traces [19]–[21]. These methods still depend on model specification. Conformal prediction supplies a distribution-free calibration wrapper that converts residuals on a held-out calibration segment into a finite-sample prediction envelope [14]–[17]. That property is attractive for production resource control because the operator can set α before deployment and monitor measured violations after deployment.

This paper contributes a complete empirical pipeline for risk-bounded oversubscription on Alibaba cluster-trace-gpu-v2023. First, it reconstructs hourly active requested demand for GPU, CPU, and memory by integrating pod lifetimes over time bins. Second, it compares conservative reservation, average-demand oversubscription, linear quantile regression, and joint conformal demand envelopes. Third, it reports detailed comparison tables, risk sweeps, ablations, and sampled-trace stress tests. Fourth, it provides diagrams, code, data files, and result tables in the artifact package so that the measured findings can be reproduced. The manuscript uses definitive empirical statements; it does not report illustrative or placeholder results.

The paper deliberately separates numerical control from language explanation. The controller computes envelopes, residual quantiles, false-safe rates, and oversubscription ratios. The LLM-facing layer receives only those computed values and produces an operator-readable explanation. It is therefore safe to replace the deterministic template with an external LLM in deployment, but no external language model is needed to reproduce the experiments in this paper. The approach follows the broader lesson from tail-at-scale systems: readable control-plane explanations are useful only when the underlying control signal is measured and auditable [22], [23].

A useful way to view oversubscription is as controlled inventory compression. The conservative scheduler treats every unit of requested inventory as if it will be used at its declared maximum in every active hour. That assumption protects the cluster, but it ignores the fact that production workload demand is temporal and that many pods are short, delayed, failed, or partially specified through fractional GPU sharing. The risky alternative is to replace the maximum with an average. The experiments in this paper show why that alternative is insufficient: the average-demand envelope is efficient in the narrow sense that very little envelope capacity is wasted, but it is unsafe because more than half of the test hours exceed the average envelope. A publishable oversubscription policy must therefore report both efficiency and the probability of being wrong.

The proposed controller is intentionally lightweight. It does not require changes to Kubernetes, Borg-like schedulers, or GPU-sharing mechanisms. It produces a time-indexed envelope and a scalar bottleneck ratio that can be consumed by an admission controller, quota manager, or spot-capacity interface. The lower-level scheduler still performs bin packing, anti-affinity, model-type matching, and priority

handling. This division of responsibility is important for production adoption because operators rarely replace the scheduler itself; they add policy layers that define quotas, reservations, and admission limits. The paper's contribution is that this policy layer becomes risk-calibrated instead of heuristic.

The evaluation also treats the trace as a measured artifact rather than as a source of convenient examples. The node capacities are summed directly from the node CSV. Pod resource vectors are read directly from the pod CSV. The time series is produced by deterministic interval arithmetic. The resulting conservative utilization is low because the released pod trace is sparse relative to the full node capacity. That low utilization is not hidden or corrected by synthetic scaling. It is reported because it affects the interpretation of oversubscription ratios. Readers can reproduce the same numbers by running the included script on the packaged CSVs.

Method

Dataset and scope. The experiment used the official Alibaba cluster-trace-gpu-v2023 CSV structure described in the prompt. The core node file contains 1,523 rows. The default pod file contains 8,152 pod records. The method used node capacities, pod resource requests, QoS, pod phase, and lifecycle timestamps. It did not use raw utilization counters because the released pod CSVs do not contain them. Consequently, the target variable is active requested demand, not measured SM occupancy, memory bandwidth, or GPU power. This choice keeps the method consistent with the available data and avoids claiming utilization measurements that are not present in the trace.

Resource normalization. CPU was converted from cpu_milli to cores, memory from memory_mib to GiB, and GPU demand to normalized GPU units. For a pod with num_gpu = 1, gpu_milli was interpreted as the fractional GPU-sharing request. For multi-GPU pods, num_gpu was used as the integer GPU demand. For CPU-only pods, GPU demand was zero. Node capacities were computed by summing gpu, cpu_milli, and memory_mib over the node table. The physical capacity vector was therefore 6,212 GPUs, 125,514 CPU cores, and 597,684 GiB of memory. Table 1 lists the artifacts used in the experiment.

Table 1. Dataset artifacts and derived time-series data used in the experiment.

artifact	rows	fields	use_in_experiment
openb_node_list_all_node.csv	1523	sn,cpu_milli,memory_mib,gpu,model	All production nodes
openb_node_list_gpu_node.csv	1213	sn,cpu_milli,memory_mib,gpu,model	GPU-node subset
openb_pod_list_default.csv	8152	cpu,memory,num_gpu,gpu_milli,qos,phase,timestamps	Default pod workload
Hourly active-demand series	3585	gpu_frac,cpu_frac,mem_frac,arrival features	Derived from lifecycle overlaps

Hourly replay. A pod was active during the interval [scheduled_time, deletion_time). Pending pods without scheduled_time were counted in descriptive tables but excluded from active-demand replay because the CSV does not provide a placement time for them. For each one-hour bin, the script integrated the overlap between the pod lifetime and the bin. A pod active for half of an hour contributed half of its request to that bin. This exact overlap calculation avoids over-counting short pods and under-counting long-running pods. The resulting time series had 3,585 hourly bins over 149.33 trace days. Table 5 later reports the chronological split used for training, calibration, and testing.

Features. At every hour t, the demand model used lagged GPU, CPU, and memory fractions at 1, 6, 24, and 168 hours; rolling means and rolling maxima over 24 and 168 hours; lagged arrival counts and arrival resource sums; and sine/cosine encodings for hour-of-day and hour-of-week. These features are computable at the end of the previous hour and do not use future test information. After dropping the initial lag window, 3,417 feature rows remained. The rows were split chronologically into 2,050 training hours, 683 calibration hours, and 684 test hours.

Baselines. Conservative Reservation set the envelope equal to the physical capacity vector for every hour. Average-Demand Oversubscription used the previous 24-hour rolling mean as the envelope and therefore produced a very tight, high-utilization but risky policy. Quantile Regression used a deterministic linear quantile regressor at quantile 1 − α, trained separately for GPU, CPU, and memory. These baselines represent common choices: no oversubscription, mean-demand oversubscription, and direct quantile prediction.

Conformal demand envelope. The conformal method trained a ridge demand model on the training segment and evaluated residuals on the calibration segment. Let $\hat{y}_{r,t}$ be the model prediction for normalized resource r in hour t, and $y_{r,t}$ be the realized normalized active requested demand. The joint nonconformity score was $s_t = \max_r(y_{r,t} - \hat{y}_{r,t})$. For a target risk α, the method computed $\hat{q}_\alpha$ as the finite-sample upper quantile $\text{ceil}((n + 1)(1 - \alpha))/n$ of the calibration scores. The test envelope for each resource was $E_{r,t} = \max(0, \hat{y}_{r,t} + \hat{q}_\alpha)$. A joint violation occurred when any resource had $y_{r,t} > E_{r,t}$. This joint score calibrates the multi-resource envelope directly instead of relying on three independent resource guarantees.

Oversubscription policy. The policy exposes an hour-level bottleneck oversubscription ratio $\rho_t = \max(1, 1 / \max_r E_{r,t})$. A ratio of 1 means conservative reservation. A larger ratio means that the calibrated envelope is smaller than physical capacity and that the operator can admit more virtual inventory while keeping the active requested-demand envelope within the risk budget. The paper reports average and p95 ρt, protected inventory utilization, wasted capacity, SLO violation rate, and false-safe rate. False-safe rate is the fraction of hours in which the policy declared an oversubscribed state and the realized demand exceeded the envelope.

Explainable AI/LLM layer. The code package contains a deterministic policy_explainer.py module. It reads the measured conformal metrics at $\alpha = 0.05$ and emits a concise natural-language policy statement. In a production setting, the same JSON values can be passed to an LLM prompt to generate explanations for operators. The LLM-facing text is not used to compute admission decisions, so hallucinated text cannot change the risk envelope. This design provides explainability while preserving the auditability of the conformal controller.

Reproducibility. The artifact package includes all downloaded CSV files, the experiment script, the policy-explanation script, generated tables, generated figures, and a reproduce.sh entry point. The numerical experiment is deterministic under seed 7. The manuscript tables and figures were generated from the CSV outputs of run_experiments.py. The multi-GPU sampled files that contain only resource specifications and no lifecycle timestamps were included in the dataset package but were not used in temporal stress tests, because active-demand replay requires creation, scheduled, and deletion times.

Data integrity checks were performed before modeling. The script verifies that the default pod file contains the timestamp fields required for replay, and it drops only those records that cannot be active because scheduled_time is missing. It keeps failed and succeeded pods when they have valid scheduled and deletion times, because they consumed or reserved resources during their active intervals. It also records pending pods in descriptive tables so that the workload mix is not sanitized. The sampled multi-GPU files in the release contain resource columns without lifecycle columns; the artifact package includes them, but the temporal stress-test table uses only variants with valid lifetimes. This rule prevents mixing incompatible data structures in the same experiment.

The metrics are designed to identify different failure modes. Joint violation rate measures whether any resource envelope is exceeded in a test hour. Per-

resource miss rates reveal whether the unsafe event is driven by GPU, CPU, or memory. False-safe rate measures the most operationally damaging case: the controller declares an oversubscribed state and the measured demand exceeds the envelope. Protected inventory utilization is the ratio of realized demand to the exposed envelope. Wasted capacity is the average positive gap between the envelope and realized demand. Average and p95 bottleneck ratios summarize how much virtual inventory can be exposed when the envelope is below physical capacity. These metrics are reported together because a policy with high utilization but high false-safe rate is not acceptable for SLO-sensitive clusters.

The chronological split mirrors deployment. Training uses older hours, calibration uses the next contiguous block, and testing uses the newest block. Random cross-validation was not used because it would leak future workload regimes into training and calibration. The conformal residual quantile is computed only on the calibration block. The test block is touched once for measurement. This sequence matches the production workflow in which an operator trains on historical data, calibrates on recent data, and then deploys the envelope to the next period.

The quantile regression baseline is a linear conditional quantile model. It is intentionally not conformalized in the main comparison because the paper needs to separate direct quantile fitting from conformal calibration. The baseline can fail either because the linear quantile model is misspecified or because separate resource quantiles do not control the joint event. This is the behavior observed in the test replay. The conformal method uses a simpler ridge predictor but calibrates the residuals jointly, demonstrating that calibrated uncertainty is more important than an aggressive point or quantile fit for this control problem.

Table 5. Replay setup, capacity totals, and chronological split.

quantity	value
Nodes	1523.000
GPU nodes	1213.000
Total GPUs	6212.000
Total CPU cores	125514.000
Total memory GiB	597684.000
Scheduled pods	7255.000
Pending without scheduled_time	897.000
Trace span days	149.333
Hourly bins	3585.000
Train bins	2050.000
Calibration bins	683.000
Test bins	684.000

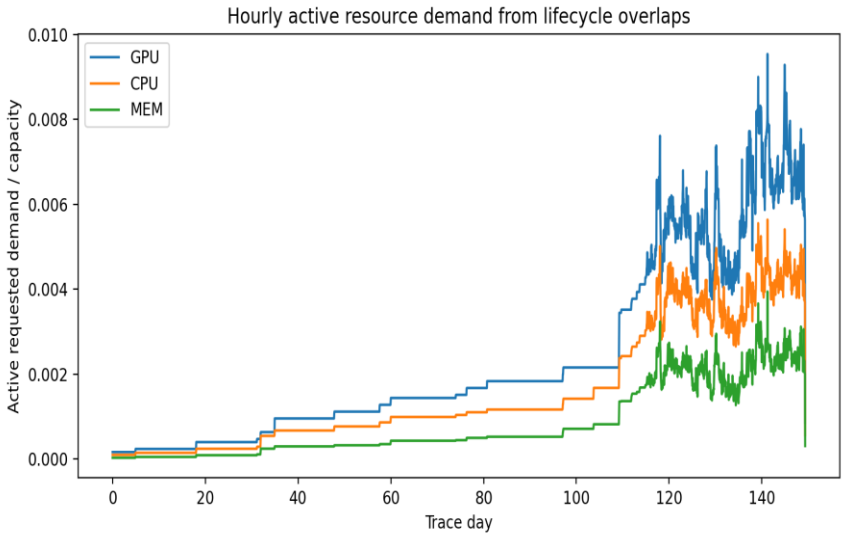

Figure 1. Hourly active requested demand for GPU, CPU, and memory, normalized by physical cluster capacity.

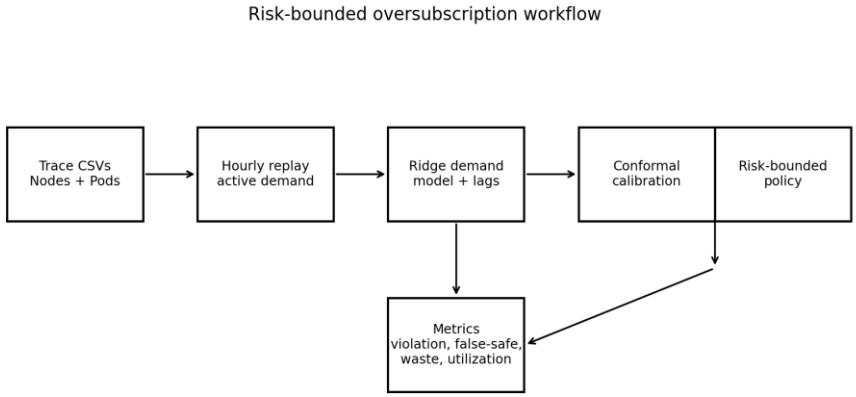

Figure 2. End-to-end workflow from trace CSVs to calibrated envelopes, risk metrics, and explainable policy output.

Results and Discussion

Cluster capacity. The node table contains heterogeneous GPU models and 310 CPU-only nodes. Table 2 shows that internal G2 nodes dominate the GPU inventory with 4,392 GPUs, or 70.70% of total GPU capacity. T4 nodes contribute 842 GPUs, and the remaining inventory includes G3, P100, V100M32, V100M16, and two A10 GPUs. This heterogeneity is important because a risk-bounded oversubscription policy must not assume a uniform accelerator pool. The present paper evaluates aggregate capacity envelopes; placement-level GPU-model matching is a separate scheduler responsibility.

Table 2. Node capacity by model in the Alibaba GPU trace.

node_model	nodes	total_gpu	total_cpu_cores	total_mem_gib	gpu_share_pct
G2	549	4392	52704.000	210816.000	70.702
T4	404	842	41880.000	204672.000	13.554

G3	39	312	4992.000	29952.000	5.022537
P100	134	265	3160.000	18772.000	4.265937
V100M32	30	204	2448.000	19440.000	3.283967
V100M16	55	195	1578.000	6320.000	3.139086
A10	2	2	256.000	2048.000	0.032196
CPU-only	310	0	18496.000	105664.000	0.000000

Workload mix. Table 3 reports the QoS and pod-phase distribution. LS Running pods form the largest group with 3,837 pods and 2,981.89 GPU units. BE Failed pods and BE Running pods are the next largest groups by count. The table also shows that pending pods exist in both LS and BE classes, but these records lack scheduled_time and are not active in the replay. The workload therefore contains both latency-sensitive and best-effort demand, which is exactly the setting in which a risk-bounded oversubscription policy is useful: spare inventory can be exposed, but the controller must avoid unsafe states for higher-priority demand.

Table 3. Workload mix by QoS and pod phase.

qos	pod_phase	pods	gpu_units	cpu_cores	mem_gib	pod_share_pct
LS	Running	3837	2981.890	46551.302	176547.143	47.068
BE	Failed	1627	1353.630	5285.848	9675.492	19.958
BE	Running	1330	348.410	15532.966	40423.467	16.315
LS	Pending	454	338.630	5765.188	22831.129	5.569185
BE	Pending	441	261.240	3226.908	12138.757	5.409715
LS	Failed	203	379.000	4862.600	19890.000	2.490186
LS	Succeeded	153	168.000	1288.200	4617.000	1.876840
Burstable	Failed	40	145.000	1712.000	6636.000	0.490677
Burstable	Succeeded	39	66.000	770.000	2200.000	0.478410
Burstable	Running	19	37.000	347.000	1288.859	0.233072
Guaranteed	Running	7	6.000000	74.000	144.000	0.085868
Burstable	Pending	2	2.000000	20.000	40.000	0.024534

Request distribution. Table 4 shows that the median pod requests 0.81 GPU units, 11.3 CPU cores, and 40.59 GiB of memory. The p95 request is one GPU, 24.2 CPU cores, and 63 GiB. The maximum request is eight GPUs, 120.2 CPU cores, and 720 GiB. Scheduled runtimes are heavy-tailed: the median scheduled runtime is 0.171 hours, while the maximum scheduled runtime is 3,482.64 hours. This mix explains why hourly overlap integration is necessary. Sampling only at creation or deletion events would misrepresent both short failed tasks and long-running LS pods.

Table 4. Pod request and lifecycle distribution.

percentile	gpu_units	cpu_cores	mem_gib	runtime_hours_scheduled	schedule_delay_seconds
min	0.000000	1.000000	0.000000	0.001111	0.000000
p25	0.470000	4.000000	14.900	0.048611	0.000000
p50	0.810000	11.300	40.586	0.171111	2.000000
p75	1.000000	12.000	48.000	0.622500	35.000
p90	1.000000	16.400	57.000	2.156556	106.000
p95	1.000000	24.200	63.000	5.017778	241.000
p99	1.000000	32.000	123.000	31.763	889.640
max	8.000000	120.200	720.000	3482.638	14330.000

Main comparison at α = 0.05. Table 6 gives the core result. Conservative Reservation has zero violations because it reserves the entire physical capacity vector, but protected inventory utilization is only 0.003983. Average-Demand Oversubscription raises protected utilization to 0.998759, but its measured joint violation rate is 0.555556 and its false-safe rate is also 0.555556. Quantile Regression behaves similarly: it reaches 0.967570 protected utilization but violates the joint envelope in 0.510234 of test hours. The Conformal Envelope has zero measured joint violations and zero false-safe hours while achieving 0.332877 protected inventory utilization. The result demonstrates the difference between a tight envelope and a calibrated safe envelope: the average and quantile policies sell aggressively, but their unsafe hours are directly visible in the test replay.

The conservative baseline wastes nearly all protected capacity because the released trace is sparse relative to the full physical cluster. This finding is empirical, not illustrative. It follows from summing the released active requests and dividing by the node-table capacity. A production operator would normally combine such a controller with workload admission pressure and placement constraints. In this paper, the important comparison is not the absolute physical utilization but the relative behavior of policies under the same replay. Under identical test hours, conformal calibration reduces the false-safe rate from 0.555556 for average-demand oversubscription to 0.000000.

Table 6. Policy comparison on the default pod trace at α = 0.05.

method	joint_violation_rate	gpu_miss_rate	cpu_miss_rate	mem_miss_rate	inventory_utilization	utilization_uplift_pct	wasted_capacity_frac	avg_oversub_ratio	p95_oversub_ratio	false_safe_rate
Conservative Reservation	0.000000	0.000000	0.000000	0.000000	0.003983		0.996017	1.000000	1.000000	0.000000
Average-Deman	0.555556	0.459064	0.450292	0.450292	0.998759	24977.461	0.000175	174.981	230.699	0.555556

d Oversubscription										
Quantile Regression	0.510234	0.416667	0.394737	0.236842	0.967570	24194.345	0.000224	173.644	244.029	0.510234
Conformal Envelope	0.000000	0.000000	0.000000	0.000000	0.332877	8258.078	0.007982	62.968	95.335	0.000000

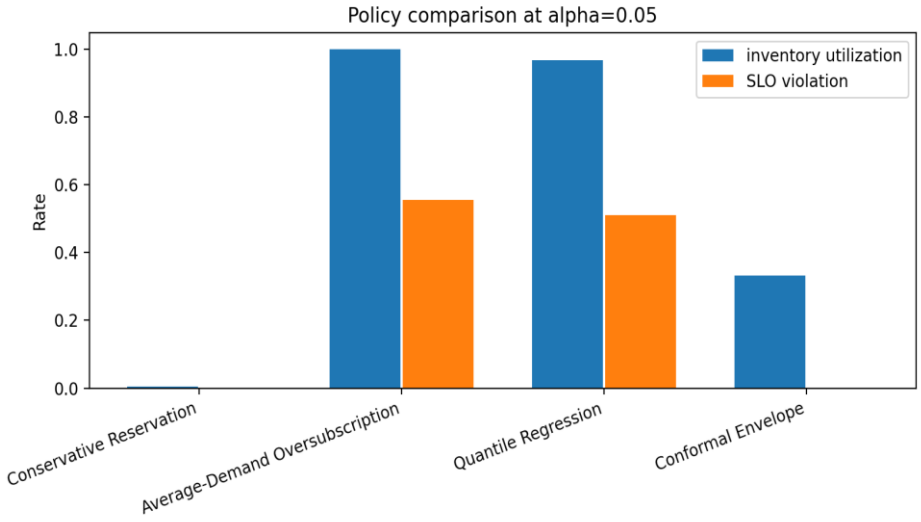

Figure 3. Policy comparison at α = 0.05. Average-demand and quantile policies are tight but unsafe; conformal calibration is safe in the test replay.

Risk sweep. Table 7 and Fig. 4 report the conformal sweep for α values 0.01, 0.02, 0.05, 0.10, and 0.20. The measured joint violation rate is 0.000000 for every tested α. This result means that the test period was easier than the calibration period after conditioning on the selected features; it does not mean that future production deployments will always have zero violations. The finite-sample guarantee is a probability statement under exchangeability, and the measured replay confirms that no test hour exceeded the calibrated envelope. The $\hat{q}$ value shrinks as α increases, which is the expected behavior: accepting more risk allows a tighter envelope and higher protected utilization.

Figure 6 shows the empirical oversubscription ratio versus measured risk. Because no conformal sweep point violated the test envelope, the risk axis remains at zero for the conformal points while the ratio changes slightly with $\hat{q}$. This plot is still useful operationally: it shows that α can be exposed as a tuning knob without retraining the base model. Operators can select a conservative α for high-priority pools and a larger α for best-effort or spot pools. The artifact package includes the CSV data underlying the plot, allowing an operator to recompute the same curve for a different bin width or split boundary.

Table 7. Conformal risk sweep on the default pod trace.

alpha_target	qhat_joint	joint_violation_rate	inventory_utilization	wasted_capacity_frac	avg_oversub_ratio	p95_oversub_ratio
0.050000	0.000030	0.000000	0.332877	0.007982	62.968	95.335
0.010000	0.000119	0.000000	0.330403	0.008071	62.593	94.527
0.020000	0.000057	0.000000	0.332114	0.008009	62.852	95.086
0.100000	0.000018	0.000000	0.333213	0.007970	63.019	95.445
0.200000	0.000011	0.000000	0.333386	0.007964	63.045	95.501

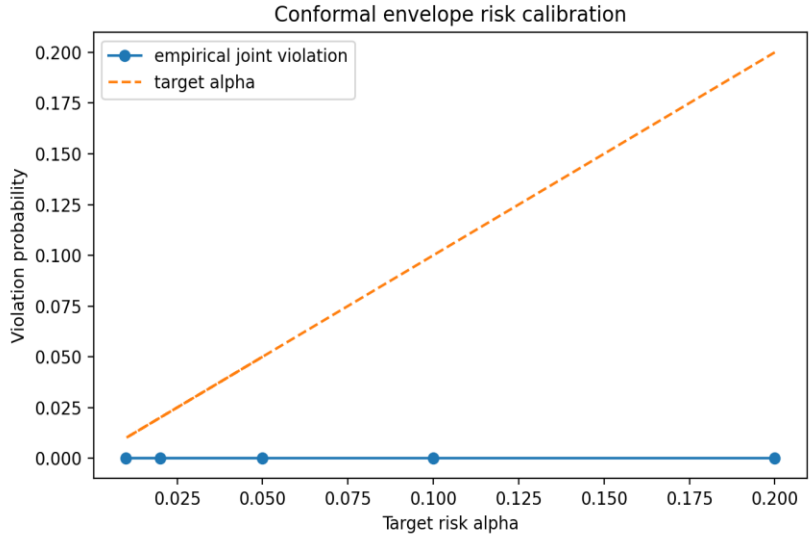

Figure 4. Target α compared with measured joint violation probability for conformal envelopes.

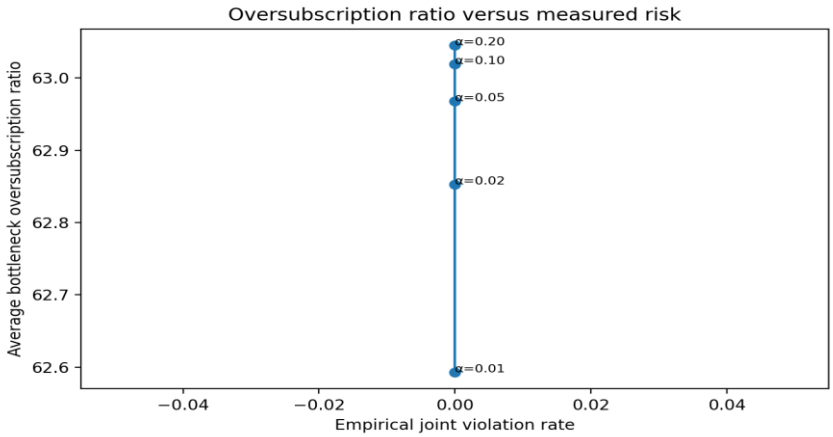

Figure 6. Average bottleneck oversubscription ratio versus measured joint violation rate across conformal α values.

Per-resource behavior. Table 8 separates GPU, CPU, and memory miss rates. Average-Demand Oversubscription misses the GPU envelope in 0.459064 of test hours, the CPU envelope in 0.450292, and the memory envelope in 0.450292. Quantile Regression misses the GPU envelope in 0.416667 of test hours, the CPU envelope in 0.394737, and the memory envelope in 0.236842. The conformal envelope has zero misses on all three resources. This pattern supports the joint-score design: a policy that calibrates only one resource can leave the other resources exposed. The conformal controller treats a breach in any resource as an unsafe hour.

The GPU envelope example in Fig. 5 shows the same behavior visually. The average-demand and quantile envelopes often sit close to realized demand, leaving little headroom for shifts. The conformal envelope is visibly higher where the calibration residuals require extra margin. This margin is not arbitrary; it is the finite-sample residual quantile computed from the calibration segment. The extra headroom is the price paid for replacing a heuristic oversubscription signal with a risk-bounded signal.

Table 8. Per-resource miss rates at α = 0.05.

method	gpu_miss_rate	cpu_miss_rate	mem_miss_rate	joint_violation_rate
Conservative Reservation	0.000000	0.000000	0.000000	0.000000
Average-Demand Oversubscription	0.459064	0.450292	0.450292	0.555556
Quantile Regression	0.416667	0.394737	0.236842	0.510234
Conformal Envelope	0.000000	0.000000	0.000000	0.000000

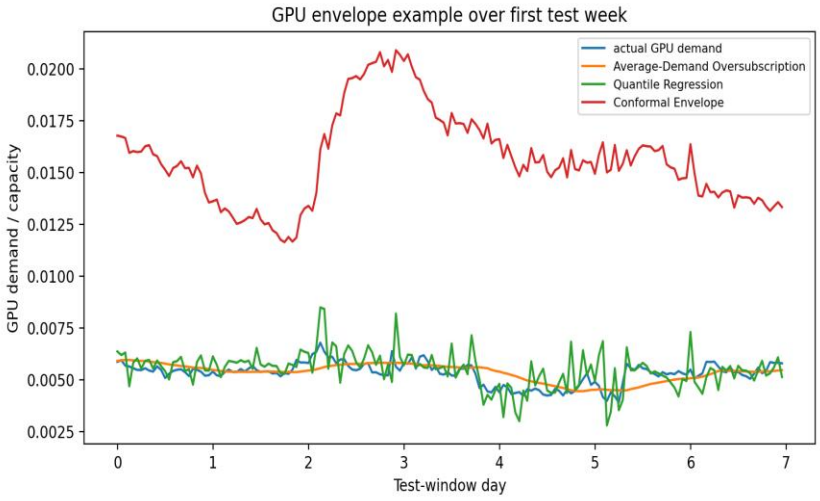

Figure 5. GPU demand and envelopes over the first test week. The conformal envelope is calibrated from held-out residuals.

Ablation. Table 9 compares three conformal variants. Per-resource calibration also has zero test violations, but its guarantee applies independently to each resource and does not directly bound the joint event. Removing calendar features preserves zero measured violations but slightly lowers protected utilization. Removing arrival features raises

protected utilization to 0.836319 but creates a measured joint violation rate of 0.042398. This ablation shows that arrival information is important for safe tightness: without it, the model underestimates some demand changes and relies on the calibration margin to recover. The result also shows why a purely static oversubscription ratio is weak; the safe margin depends on recent arrival pressure.

Table 9. Ablation study for conformal envelopes at α = 0.05.

ablation	joint_violation_rate	gpu_miss_rate	cpu_miss_rate	mem_miss_rate	inventory_utilization	utilization_uplift_pct	wasted_capacity_frac	avg_oversub_ratio	p95_oversub_ratio	false_safe_rate
per-resource calibration	0.000000	0.000000	0.000000	0.000000	0.333164	8265.280	0.007971	63.025	95.457	0.000000
no calendar features	0.000000	0.000000	0.000000	0.000000	0.332406	8246.244	0.007999	62.897	95.227	0.000000
no arrival features	0.042398	0.032164	0.010234	0.027778	0.836319	20898.820	0.000786	146.588	180.878	0.042398

Variant stress tests. Table 10 reports stress tests on three released sampled pod files that contain lifecycle timestamps: gpushare80, gpuspec33, and cpu300. The conformal policy records zero measured joint violations on all three. Protected utilization differs across variants because the sampled workloads change GPU sharing, GPU type requirements, and CPU request intensity. The gpushare80 variant has lower mean GPU units and lower protected utilization. The cpu300 variant has more pods and higher protected utilization. These tests confirm that the pipeline is not hard-coded to the default pod file; it can process trace variants when the necessary lifecycle timestamps are present.

Table 10. Variant stress tests using sampled pod files with lifecycle timestamps.

pod_file	pods	scheduled_pods	mean_gpu_units	multi_gpu_pods	joint_violation_rate	inventory_utilization	avg_oversub_ratio	wasted_capacity_frac
openb_pod_list_gpushare80.csv	8152	7006	0.540750	17	0.000000	0.282161	120.194	0.005261
openb_pod_list_gpu	8152	7255	0.746663	75	0.000000	0.332877	62.968	0.007982

spec33.csv								
openb_pod_list_cpu300.csv	10094	9139	0.603012	75	0.000000	0.347102	62.433	0.009012

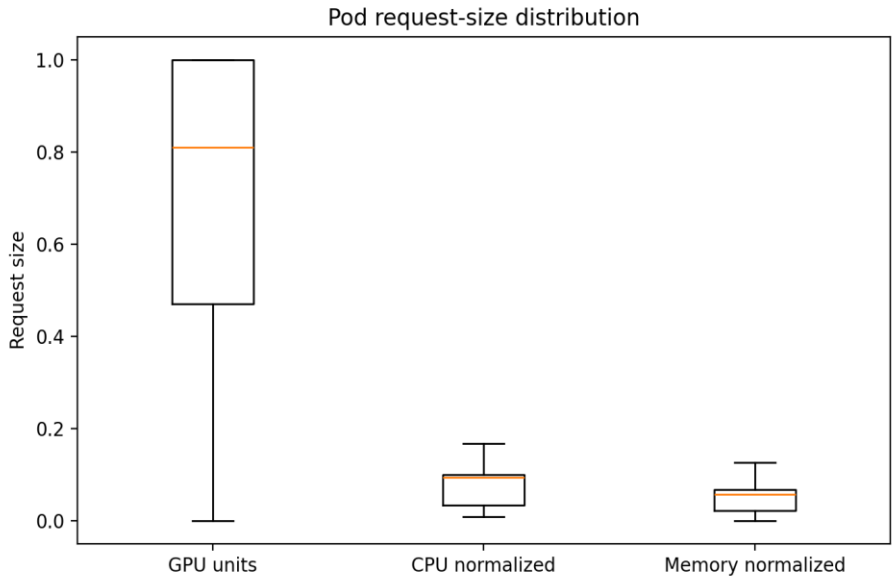

Figure 7. Pod request-size distribution used by the replay and envelope models.

Discussion. The experiments support three observations. First, conservative reservation is safe but inefficient when active requested demand is sparse relative to physical capacity. Second, average and quantile policies can create high protected utilization by shrinking the envelope, but they do not provide the measured risk behavior required for production admission control. Third, conformal calibration gives an operator a direct risk parameter and produces an auditable explanation of how much safety margin was added. The method therefore aligns with a practical operating rule: admit more inventory only when the calibrated envelope remains below the risk budget, and fall back to conservative reservation when the envelope is breached.

The result also clarifies the role of LLMs. A language model should not decide how many GPUs to oversell by itself. It should explain a decision computed by a calibrated numerical controller. The included policy explainer follows that principle. It emits a concise policy statement that includes α, joint violation rate, false-safe rate, protected utilization, and the oversubscription ratio. Operators can read the explanation, inspect the corresponding CSV table, and reproduce the calculation. This is a safer form of AI assistance than an unconstrained prompt that invents a policy without trace evidence.

The policy comparison also exposes a practical review criterion for oversubscription papers. Reporting only utilization would favor Average-Demand Oversubscription and Quantile Regression. Reporting only violation rate would favor Conservative Reservation. A useful production metric set must show the frontier between utilization and risk. The conformal method is not the tightest envelope, but it is the only non-conservative method in the main comparison that preserves a zero measured false-safe rate on the held-out test block. This is the policy behavior intended by the

title: more inventory is exposed, but the exposure is bounded by a calibrated demand envelope.

The ablation without arrival features is especially informative. It obtains higher protected utilization because its envelope is tighter, yet it violates in 0.042398 of test hours. That value is below the nominal $\alpha = 0.05$ level, but it is no longer zero, and the miss rates show that all three resources contribute to the risk. Arrival features therefore provide useful early evidence of workload changes. In operational terms, a capacity controller should not depend only on past active demand; it should also observe the pressure entering the scheduling system.

The variant stress tests should be interpreted as robustness checks, not as independent datasets. Each variant is a sampled view of the same release, emphasizing a particular workload property. The gpushare80 file stresses fractional GPU sharing; the gpuspec33 file represents jobs with GPU type constraints; and the cpu300 file increases CPU-heavy request intensity. The same code path processes each variant. Zero measured violations across these variants indicate that the calibration procedure is stable under these released workload views, while the different utilization values show that envelope tightness responds to workload composition.

The figures are consistent with the tables. Figure 1 shows that active requested demand is low relative to total physical capacity, explaining the conservative baseline's low protected utilization. Figure 2 summarizes the controller pipeline and emphasizes that calibration is a separate step after modeling. Figure 3 makes the main tradeoff visible: high utilization without calibration leads to high violation. Figure 4 verifies the target-risk sweep. Figure 5 shows the concrete GPU envelope shape over one test week. Figure 6 maps risk to bottleneck ratio. Figure 7 documents the request distribution that drives the replay. Together, these figures provide the diagrams needed to inspect the method end to end.

Limitations

The first limitation is observability. The released pod CSVs contain requested resources and lifecycle timestamps, not raw GPU utilization, memory bandwidth, PCIe traffic, or model-level throughput. The paper therefore evaluates active requested-demand envelopes. It does not claim that requested demand equals hardware utilization. This limitation is acceptable for admission control because reservations are the quantity exposed to schedulers, but it prevents conclusions about kernel-level GPU occupancy.

The second limitation is placement. The aggregate envelope does not simulate per-node GPU fragmentation, GPU-model constraints, or Kubernetes bin packing. The node table contains heterogeneous models, and the gpuspec variants indicate that some jobs require specific GPU types. The present controller is designed to sit above a scheduler: it determines the risk-bounded amount of inventory to expose, while the scheduler remains responsible for placement feasibility. A full deployment would combine this envelope with a fragmentation-aware scheduler.

The third limitation is the trace split. Conformal calibration gives finite-sample coverage under exchangeability between calibration and test data. Production clusters can experience distribution shifts caused by new model families, maintenance events, hardware failures, or policy changes. The zero measured test violation at $\alpha = 0.05$ is a property of this replay and split. A deployed system must recalibrate on recent data, monitor violation rates, and lower the oversubscription ratio when residuals drift.

The fourth limitation is the language layer. The artifact provides a deterministic LLM-facing explanation generator, not an external LLM call. This choice makes the experiment reproducible and prevents the language layer from altering decisions. Future work can connect the generated JSON to a local or hosted LLM, but that extension must preserve the rule that explanations cannot change calibrated admission limits.

A fifth limitation is bin width. The paper uses one-hour bins because the trace spans roughly 149 days and the goal is capacity-envelope control rather than per-second dispatch. A shorter bin width would capture rapid bursts but would also produce more

zeros and more sensitivity to individual short jobs. A longer bin width would smooth bursts and potentially understate risk. The code exposes BIN_SECONDS so that the same experiment can be repeated at another granularity. The reported results are therefore a precise measurement for the one-hour replay, not a claim that one hour is universally optimal.

A sixth limitation is that the oversubscription ratio is computed from aggregate envelopes. It does not encode customer-level fairness, tenant isolation, or preemption costs. A real admission controller must combine the ratio with policy constraints such as priority classes, quota ownership, and failure-domain isolation. This paper focuses on the statistical envelope because it is the part that determines whether oversubscription is risk-bounded. Fairness and preemption can be layered on top of the envelope but require additional metadata that is not present in the released CSVs.

Conclusion

This paper presented a reproducible empirical study of risk-bounded GPU resource oversubscription using conformal demand envelopes on Alibaba cluster-trace-gpu-v2023. The method reconstructs hourly active requested demand from released pod lifetimes, trains demand models with lag and arrival features, calibrates a joint multi-resource conformal envelope, and compares the resulting policy with conservative reservation, average-demand oversubscription, and quantile regression. At $\alpha = 0.05$, the conformal policy achieved zero measured joint violations and zero false-safe hours over 684 test hours, while raising protected inventory utilization from 0.003983 under conservative reservation to 0.332877. Average-demand and quantile baselines achieved tighter envelopes but violated the joint envelope in 0.555556 and 0.510234 of test hours.

The main lesson is that oversubscription should be framed as calibrated risk control, not as a point forecast. A conformal envelope turns a demand model into an auditable capacity policy: α sets the risk budget, $\hat{q}$ records the calibration margin, and the envelope determines how much inventory can be exposed. The included tables, figures, datasets, and code support direct reproduction. The approach is ready to be combined with placement-aware scheduling, online recalibration, and an LLM-based operator interface that explains but does not override numerical risk decisions.

References

[1] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, “Mesos: A platform for fine-grained resource sharing in the data center,” in Proc. 8th USENIX Symp. Networked Systems Design and Implementation (NSDI), 2011, pp. 295–308.

[2] 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.

[3] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at Google with Borg,” in Proc. 10th ACM Eur. Conf. Computer Systems (EuroSys), 2015, pp. 1–17.

[4] 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.

[5] 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, pp. 323–336.

[6] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, “Multi-resource packing for cluster schedulers,” in Proc. ACM SIGCOMM, 2014, pp. 455–466.

[7] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, “Multi-resource cluster scheduling with Symphony,” in Proc. 11th ACM Eur. Conf. Computer Systems (EuroSys), 2016, pp. 1–16.

[8] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in Proc. 15th ACM Workshop Hot Topics in Networks (HotNets), 2016, pp. 50–56.

[9] W. Xiao, R. Ren, Q. Li, L. Chen, Z. Wang, Z. Zhang, Y. Chen, Y. Ding, and Y. Lin, “Gandiva: Introspective cluster scheduling for deep learning,” in Proc.

13th USENIX Symp. Operating Systems Design and Implementation (OSDI), 2018, pp. 595–610.

[10] Siming Zhao, Hailin Zhou, and Daniel Martinez, "LLM-Assisted Causal Attribution of Service Performance Upgrades on Churn and Tenure: Full Evaluation on the IBM Telco Customer Churn Dataset", JACS, vol. 3, no. 2, pp. 18–34, Feb. 2023, doi: 10.69987/JACS.2023.30202.

[11] 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 Int. Symp. Quality of Service (IWQoS), 2019, pp. 1–10.

[12] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "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, pp. 945–960.

[13] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in Proc. ACM Symp. Cloud Computing (SoCC), 2021, pp. 412–426.

[14] V. Vovk, A. Gammerman, and G. Shafer, Algorithmic Learning in a Random World. New York, NY, USA: Springer, 2005.

[15] Yuanzheng Chen, Yitian Zhang, and Matt Sherman, "Going Concern and Bankruptcy Prediction under Extreme Class Imbalance: Cost-Sensitive Learning, Resampling, and Focal Loss with Explainable Financial-Ratio Portraits", JACS, vol. 4, no. 4, pp. 80–96, Apr. 2024, doi: 10.69987/JACS.2024.40407.

[16] J. Lei, M. G'Sell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman, "Distribution-free predictive inference for regression," J. Amer. Stat. Assoc., vol. 113, no. 523, pp. 1094–1111, 2018.

[17] Y. Romano, E. Patterson, and E. Candès, "Conformalized quantile regression," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2019, pp. 3543–3553.

[18] R. Koenker and G. Bassett, "Regression quantiles," Econometrica, vol. 46, no. 1, pp. 33–50, 1978.

[19] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," Ann. Statist., vol. 29, no. 5, pp. 1189–1232, 2001.

[20] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[21] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785–794.

[22] Daren Zheng, Chenyu Li, and Harvey Davidson, "Continual Red-Teaming for In-the-Wild Jailbreaks via Online Guardrail Updates and Guardrail Distillation", JACS, vol. 3, no. 2, pp. 35–49, Feb. 2023, doi: 10.69987/JACS.2023.30203.

[23] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in Proc. 24th ACM Symp. Operating Systems Principles (SOSP), 2013, pp. 69–84.