Regime-Aware Resource Demand Forecasting for Cloud Scheduling: When History Beats ML and When ML Matters

Kalab M. Kiros, Jinwei Liu*
Department of Computer and Information Sciences
Florida A&M University, Tallahassee, FL 32307, USA
Email: {kalab1.kiros, jinwei.liu}@famu.edu

***Abstract*—Efficient cluster scheduling requires reliable forecasts of resource demand, yet production workloads are heterogeneous, bursty, and strongly time-dependent. Using the Google Cluster-Usage Traces v3, we study leakage-safe prediction of three scheduling-relevant metrics: mean CPU demand, normalized memory pressure (average memory relative to assigned memory), and tail CPU demand (p95) as a burst-risk indicator. We develop a preprocessing and evaluation protocol that explicitly addresses two common threats to validity in trace-based learning: post-execution feature leakage (e.g., usage-derived fields) and identity leakage from random splits when recurring workloads appear in both training and test sets. Under time-ordered and gap-based splits, we compare gradient-boosted tree models (LightGBM) against strong history-only baselines (LastSeen and EMA) and perform a cold-start analysis by evaluating the first K occurrences of each workload entity. Results reveal a clear regime shift: for warm, recurring workloads, simple entity-history predictors achieve near-optimal accuracy and consistently outperform learned models; however, in cold-start settings where history is unavailable, LightGBM substantially improves CPU mean and tail forecasts (e.g., large gains in R^2 for first-occurrence entities). These findings support a practical scheduling strategy: a hybrid, regime-aware policy that uses machine learning (ML) as a cold-start fallback and switches to lightweight history-based prediction as observations accumulate.**

***Index Terms*—resource demand forecasting, scheduling, cloud, machine learning, leakage, regime shift**

I. Introduction

Modern cloud computing infrastructures rely on large-scale clusters to host heterogeneous, dynamic, and time-dependent workloads. Efficient scheduling in such environments requires accurate estimates of future resource demand so that compute and memory resources can be allocated effectively while maintaining service-level objectives (SLOs). In particular, prediction-driven overcommitment and peak-aware allocation motivate forecasting not only average resource usage, but also high-percentile behavior that captures burst risk [1–4].

Recent scheduler systems show that modern cluster management increasingly relies on workload-aware and adaptive decision making to improve objectives such as goodput, fairness, placement efficiency, job completion time, and energy or carbon efficiency [5–8]. However, these systems primarily study scheduler design and optimization. In contrast, our work focuses on the upstream forecasting question: when does a learned resource-demand predictor provide value over simple history-based baselines under leakage-safe evaluation?

Trace-based machine learning for resource forecasting also introduces important evaluation pitfalls. Production traces often include both scheduling-time metadata and post-execution measurements [9, 10]. If usage-derived fields are mistakenly used as predictors, models may benefit from post-run feature leakage by accessing information that would not be available at scheduling time. More broadly, data leakage has been shown to produce overly optimistic conclusions in ML-based studies and to undermine reproducibility [11]. In addition, recent production workload studies show that large-scale cloud and GPU-cluster traces contain substantial workload heterogeneity, temporal structure, and recurring user or workload behavior [2, 12]. These properties can create identity leakage when random train/test splits allow the same workload entities to appear in both training and testing partitions. Such leakage can inflate performance estimates and obscure the true temporal generalization ability of forecasting models.

This paper studies leakage-safe resource demand forecasting using the Google Cluster-Usage Traces v3 dataset. We focus on three scheduling-relevant targets: mean CPU demand, normalized memory pressure, and tail CPU demand (p95), which serves as a burst-risk indicator. Our evaluation restricts all predictors to scheduling-time information and past-only historical features, while post-execution measurements are used only for target construction.

We compare a gradient-boosted tree model, LightGBM, against lightweight history-based predictors, including LastSeen and Exponential Moving Average (EMA). To understand when each approach is useful, we perform a regime-aware analysis across workload recurrence levels, from true cold-start entities to warm recurring workloads with substantial history. Our results show a clear regime shift. In cold-start settings, where little or no entity history is available, LightGBM improves CPU forecasting by exploiting cross-entity scheduling-time patterns. However, once workloads accumulate sufficient history, simple history-based predictors rapidly become highly competitive and often outperform learned models with much lower complexity. The contributions of this paper are summarized as follows:

* Corresponding Author. Email: jinwei.liu@famu.edu.

- We present a leakage-safe forecasting protocol for production cluster traces that avoids post-execution feature leakage and reduces identity leakage through time-ordered evaluation.
- We compare LightGBM with strong lightweight history baselines under cold-start and warm recurring workload regimes.
- We show that ML provides its clearest benefit in sparse-history CPU forecasting, while simple history-based predictors dominate warm recurring workloads.
- We provide empirical evidence supporting a practical regime-aware forecasting strategy for cloud scheduling.

II. Related Work

A. Workload Forecasting in Cloud Environments

Accurate workload forecasting has long been recognized as a key enabler of efficient resource allocation in cloud systems. Recent work has emphasized that modern production workloads are highly dynamic, bursty, and often non-stationary, making naive provisioning strategies inefficient. Cortez *et al.* [13] detailed Azure's VM workload and proposed Resource Central (RC) to predict workloads for improved resource management in large cloud platforms. They demonstrated that machine learning models significantly outperform traditional statistical baselines in predicting CPU and memory utilization. More recently, Rossi *et al.* [14] proposed uncertainty-aware workload forecasting methods for cloud computing, arguing that accurate demand prediction remains central to balancing service quality and provisioning cost in realistic deployments. Diao *et al.* [12] analyzed forecasting algorithms for intelligent resource scaling on real-world cloud workloads. Their results highlight that production traces often contain recurring workload patterns but also significant heterogeneity and burstiness, making forecasting highly context dependent. This observation is directly relevant to our setting: if workloads are partly repetitive, then simple history-based predictors may already provide strong performance, whereas ML may be most valuable in cold-start or sparse-history scenarios.

B. Learning-Based Scheduling and Resource Management

In parallel with workload forecasting, a large body of recent systems work has investigated machine learning and optimization for scheduling and resource management. Pollux [5] showed that co-adaptive scheduling can improve goodput in deep learning clusters by jointly optimizing resource allocation and training efficiency. Shockwave [6] extended this direction by introducing future-aware scheduling for dynamically adaptive ML jobs, improving both fairness and makespan. CASSINI [7] further demonstrated that cluster scheduling can benefit from richer workload-aware abstractions by jointly reasoning about communication patterns and placement, substantially improving average and tail job completion time. Most recently, GREEN [8] incorporated carbon awareness into ML-cluster scheduling, showing that modern scheduling policies increasingly optimize not only job completion time but also broader efficiency objectives.

These studies collectively show that predictive and adaptive scheduling policies can significantly outperform static heuristics in large-scale systems. However, they are primarily concerned with scheduler design and optimization under known or observed workload characteristics, rather than explicitly studying leakage-safe trace forecasting or the distinction between cold-start and recurring workload regimes.

C. Leakage, Evaluation Protocols, and History-Based Baselines

Although ML-based forecasting is increasingly popular, trace-based evaluation is vulnerable to methodological pitfalls. One risk is *post-execution leakage*, where usage-derived quantities are inadvertently used as predictors for targets that are themselves derived from those same measurements. Another is *identity leakage*, where random train/test splits allow recurring entities to appear in both partitions, enabling memorization rather than true temporal generalization. These concerns are especially important in production cluster traces, where workloads often recur over time and entity behavior can be highly persistent. For time-dependent data, prior work on forecasting evaluation has shown that random cross-validation can be invalid or misleading when temporal dependence is present [15]. This motivates leakage-safe protocols based on chronological ordering and temporal separation.

Our work differs from much of the prior literature in two ways. First, we restrict all predictors to scheduling-time information and past-only history statistics, using post-run measurements strictly for target construction. Second, rather than assuming that ML always dominates history-based prediction, we explicitly compare LightGBM against lightweight entity-history baselines such as LastSeen and EMA under time-ordered and gap-based splits. This enables us to ask not only whether ML improves accuracy, but also *when* it improves accuracy.

III. Problem Statement

Efficient production cluster scheduling requires reliable forecasts of future resource demand. Given a time-ordered stream of workload observations, the goal is to predict scheduling-relevant resource targets using only information available at or before scheduling time.

In this work, we consider three prediction targets: mean CPU demand, normalized memory pressure, and p95 CPU demand. Mean CPU captures average compute usage, normalized memory pressure measures memory usage relative to assigned memory, and p95 CPU captures burst-risk behavior relevant to peak-aware scheduling.

The study is guided by two research questions:

1) How can resource demand be forecast from production traces without introducing post-execution or identity leakage?
2) Under which workload regimes does machine learning outperform strong history-based predictors such as LastSeen and EMA?

To answer these questions, we evaluate LightGBM and history-based predictors under time-ordered and gap-based

TABLE I: Cold-start and warm regime proportions based on entity occurrence order.

Bucket	Count	Percent (%)
ColdStart_K1 (first only)	2,261	0.56
ColdStart_K5 (first 5)	7,472	1.84
ColdStart_K20 (first 20)	20,562	5.07
Warm (20+)	385,332	94.93

splits. We further separate workloads by recurrence level, distinguishing cold-start cases with little or no prior history from warm recurring workloads with sufficient historical observations. The objective is not only to maximize aggregate prediction accuracy, but also to identify when ML adds practical value and when lightweight historical forecasting is sufficient.

IV. Data and Leakage-Safe Protocol

A. Dataset Overview

This study uses the Google Cluster-Usage Traces v3, a production cluster trace containing time-indexed workload and resource-usage measurements from Google cluster environments. The trace includes collection- and instance-level metadata, request-time scheduling information, assigned memory, resource requests, and post-execution usage summaries. In this work, we use only scheduling-time metadata and past-only history features as predictors, while post-execution fields such as `average_usage`, `maximum_usage`, and `cpu_usage_distribution` are reserved strictly for target construction. We focus on three scheduling-relevant targets: mean CPU demand, normalized memory pressure, and p95 CPU demand as a burst-risk indicator. Because the trace is time-indexed and contains recurring workload entities, it is well suited for studying cold-start and warm recurring forecasting regimes under leakage-safe evaluation.

Table I shows that most observations belong to warm recurring workloads, while true cold-start examples form a small but operationally important subset. This imbalance motivates a regime-aware evaluation rather than relying only on aggregate test-set performance.

B. Threats to Validity in Trace-Based Forecasting

Two major threats arise when applying machine learning to production cluster traces.

(T1) Post-run feature leakage. Several fields in the trace are measured during or after execution, including `average_usage`, `maximum_usage`, and `cpu_usage_distribution`. If these fields are used as model inputs, the model can indirectly observe the target it is supposed to predict. This produces unrealistically high performance and does not reflect information available at scheduling time. To avoid this, we use post-execution measurements only to construct labels and exclude them from all predictor sets.

(T2) Identity leakage from random splitting. Many workloads recur over time under identifiers such as `collection_logical_name`. If examples are randomly split into training and testing partitions, the same workload entity may appear in both sets, allowing models to memorize entity-specific behavior rather than generalize temporally. To reduce this risk, we sort all observations chronologically, construct history features using only prior observations, and evaluate using time-ordered splits with a temporal gap.

C. Leakage-Safe Preprocessing

Our preprocessing pipeline follows four leakage-safe principles.

First, serialized resource fields are parsed into numeric CPU and memory variables. For example, `resource_request` is parsed into request-time CPU and memory features, while `average_usage` is used only to construct post-run targets. Similarly, `cpu_usage_distribution` is parsed to derive the p95 CPU target.

Second, we define a strict feature whitelist containing only information available at or before scheduling time. These features include request-time CPU and memory, assigned memory, scheduling class, priority, collection type, cluster identifier, and lightweight derived request features such as log-transformed requests and request-to-memory ratios. Usage-derived fields are excluded from the model inputs.

Third, all historical features are constructed in a past-only manner. Observations are sorted by time, and entity-history features such as last-seen values, rolling means, and rolling standard deviations are computed using one-step shifting so that the current or future target is never used to predict itself.

Fourth, recurring workload identity is defined primarily using `collection_logical_name`. This identity is used to construct LastSeen, EMA, and occurrence-index buckets, but raw entity identifiers are not used directly as learned-model inputs. This design supports comparison between cold-start workloads with little or no history and warm recurring workloads with sufficient prior observations.

Missing numeric values are imputed using training-set medians, and missing categorical values are imputed using training-set modes before one-hot encoding. For normalized memory pressure, observations with invalid or nonpositive assigned memory are treated as missing for that target, while valid values above one are retained because they may reflect temporary overuse, accounting effects, or scaling behavior.

V. Methodology

A. Problem Formulation and Data Representation

We consider a time-ordered stream of instance observations indexed by $i \in \{1, \ldots, N\}$. Each observation i is associated with: (i) a timestamp t_i taken from `start_time`; (ii) a workload entity identifier e_i, defined primarily by `collection_logical_name`; (iii) a vector of scheduling-time features $\mathbf{x}_i \in \mathbb{R}^d$; and (iv) a vector of post-execution regression targets $\mathbf{y}_i \in \mathbb{R}^3$ derived from usage measurements. The dataset is represented as

$$\mathcal{D} = \{(t_i, e_i, \mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}, \qquad t_1 \le t_2 \le \cdots \le t_N. \quad (1)$$

The learned model is intended to predict future resource demand using only information available at or before scheduling time. Therefore, post-execution usage fields are used only to construct labels and are excluded from the predictor set. Raw entity identifiers such as `collection_logical_name`

and `user` are used to construct past-only history features and recurrence-regime buckets, but are not directly included as learned-model inputs. This design reduces direct identity memorization while still allowing explicit comparison between cold-start and recurring workload regimes. Trace semantics follow the official Google Cluster-Usage Traces v3 documentation [16].

B. Forecasting Targets

We predict three scheduling-relevant targets: mean CPU demand, normalized memory pressure, and tail CPU demand. Because these quantities are derived from execution-time measurements, they are treated strictly as labels and are never used as predictors.

a) Mean CPU demand: Let $\overline{u}_i^{\mathrm{cpu}}$ denote the average CPU utilization of instance i over its measurement window. We define

$$y_{i,\mathrm{mean}}^{\mathrm{cpu}} = \overline{u}_i^{\mathrm{cpu}}. \tag{2}$$

b) Normalized memory pressure: Let $\overline{u}_i^{\mathrm{mem}}$ denote average memory usage and let a_i^{mem} denote assigned memory from `assigned_memory`. We define

$$y_{i,\mathrm{norm}}^{\mathrm{mem}} = \overline{u}_i^{\mathrm{mem}} / a_i^{\mathrm{mem}}, \qquad a_i^{\mathrm{mem}} > 0. \tag{3}$$

Observations with invalid or nonpositive assigned memory are treated as missing for this target. Values above one are retained because temporary overages may reflect accounting behavior, scaling effects, or short-term pressure beyond the assigned-memory value.

c) Tail CPU demand (p95): For each observation, the trace provides an 11-point coarse CPU usage distribution

$$\mathbf{q}_i = (q_{i,0}, q_{i,10}, \ldots, q_{i,100}), \tag{4}$$

where $q_{i,p}$ denotes the estimated pth percentile of CPU utilization. We approximate p95 CPU demand by linear interpolation between $q_{i,90}$ and $q_{i,100}$:

$$\begin{aligned} y_{i,\mathrm{p95}}^{\mathrm{cpu}} &= q_{i,90} + (95-90)/(100-90)\,(q_{i,100} - q_{i,90}) \\ &= q_{i,90} + 0.5\,(q_{i,100} - q_{i,90})\,. \end{aligned} \tag{5}$$

This target captures burst-risk behavior and complements mean CPU demand. We use p95 CPU as a direct forecasting target because it is operationally relevant for peak-aware scheduling.

C. Feature Construction

All predictors are restricted to two categories: scheduling-time features and past-only history features. Let $\epsilon > 0$ be a small numerical constant; in our implementation, $\epsilon = 10^{-12}$.

1) Scheduling-Time Features: From the `resource_request` structure, we extract requested CPU and requested memory:

$$r_i^{\mathrm{cpu}} = \mathrm{ReqCPU}(i), \qquad r_i^{\mathrm{mem}} = \mathrm{ReqMem}(i). \tag{6}$$

To reduce skewness, we also use log-transformed request features:

$$x_{i,\log}^{\mathrm{cpu}} = \log(1 + r_i^{\mathrm{cpu}}), \qquad x_{i,\log}^{\mathrm{mem}} = \log(1 + r_i^{\mathrm{mem}}). \tag{7}$$

We include lightweight request-ratio features:

$$x_{i,\mathrm{ratio}}^{\mathrm{mem}} = r_i^{\mathrm{mem}} / a_i^{\mathrm{mem}}, \qquad a_i^{\mathrm{mem}} > 0, \tag{8}$$

$$x_i^{\mathrm{cpu/mem}} = r_i^{\mathrm{cpu}} / (r_i^{\mathrm{mem}} + \epsilon). \tag{9}$$

We also include a simple constraint-complexity feature. Let $\mathrm{Tok}(\cdot)$ tokenize the `constraint` field into alphanumeric tokens. Then

$$x_i^{\mathrm{con}} = \left|\mathrm{Tok}(\texttt{constraint}_i)\right|. \tag{10}$$

Categorical scheduling-time metadata include fields such as `instance_events_type`, `scheduling_class`, `collection_type`, `priority`, `collections_events_type`, `cluster`, and `event`. Numeric features are imputed using training-set medians, while categorical features are imputed using training-set modes and one-hot encoded.

2) Past-Only History Features: All observations are sorted chronologically before history construction. For an entity e, define the set of prior indices for observation i as

$$\mathcal{P}_i(e) = \{j < i : e_j = e\}. \tag{11}$$

All history features are computed only from $\{(e_j, \mathbf{y}_j) : j < i\}$ using one-step shifting. Thus, the current observation and future observations are never used to construct predictors.

a) Last-seen features: Let $j^\star(i,e) = \max \mathcal{P}_i(e)$ when $\mathcal{P}_i(e) \neq \emptyset$. For a target series y, the last-seen statistic is

$$\mathrm{Last}(i, e; y) = \begin{cases} y_{j^\star(i,e)}, & \text{if } \mathcal{P}_i(e) \neq \emptyset, \\ \mathrm{NaN}, & \text{otherwise.} \end{cases} \tag{12}$$

We compute last-seen values for all three targets.

b) Rolling history features: Let $\mathcal{P}_i^{(w)}(e)$ denote the most recent w indices in $\mathcal{P}_i(e)$. We compute rolling means and standard deviations as

$$\mu_i^{(w)}(e; y) = \frac{1}{|\mathcal{P}_i^{(w)}(e)|} \sum_{j \in \mathcal{P}_i^{(w)}(e)} y_j, \tag{13}$$

$$\sigma_i^{(w)}(e; y) = \sqrt{\frac{1}{|\mathcal{P}_i^{(w)}(e)|} \sum_{j \in \mathcal{P}_i^{(w)}(e)} \left(y_j - \mu_i^{(w)}(e; y)\right)^2}. \tag{14}$$

If the required minimum history length is unavailable, the rolling feature is treated as missing and handled by the training-set imputation procedure.

c) Global rolling summaries: To capture broad temporal drift, we also compute global rolling means over the last w observations, independent of entity:

$$W_i^{(w)} = \{j : \max(1, i - w) \leq j < i\}, \tag{15}$$

$$\mu_{i,\mathrm{glob}}^{(w)}(y) = \frac{1}{|W_i^{(w)}|} \sum_{j \in W_i^{(w)}} y_j. \tag{16}$$

These features are also shifted so that only past observations contribute.

D. Modeling Approach

1) LightGBM Predictor: Let $\phi(\cdot)$ denote the preprocessing transformation applied to raw feature vector $\mathbf{x}_i$. The final model input is

$$\mathbf{z}_i = \phi(\mathbf{x}_i). \tag{17}$$

TABLE II: Fixed LightGBM hyperparameter settings.

Parameter	Value	Purpose
`n_estimators`	2500	Large ensemble with small learning rate.
`learning_rate`	0.03	Conservative boosting updates.
`num_leaves`	63	Controls tree complexity.
`min_data_in_leaf`	80	Reduces unstable sparse-region splits.
`feature_fraction`	0.9	Column subsampling.
`bagging_fraction`	0.9	Row subsampling.
`bagging_freq`	1	Applies bagging at each boosting step.
`reg_alpha`	0.2	ℓ_1 regularization.
`reg_lambda`	2.0	ℓ_2 regularization.
`random_state`	42	Reproducibility.

Our learned predictor is a multi-output gradient-boosted decision tree model based on LightGBM [17]. We fit one regressor per target using `MultiOutputRegressor` [18]. Let $k \in \{1, 2, 3\}$ index the targets $\{y^{\text{mem}}_{\text{norm}}, y^{\text{cpu}}_{\text{mean}}, y^{\text{cpu}}_{\text{p95}}\}$. The prediction for target k is

$$\hat{y}_i^{(k)} = f_k(\mathbf{z}_i). \tag{18}$$

Each LightGBM regressor is an additive ensemble of regression trees:

$$f_k(\mathbf{z}) = \sum_{m=1}^{M} \eta h_{k,m}(\mathbf{z}), \tag{19}$$

where M is the number of trees, η is the learning rate, and $h_{k,m}$ is the mth regression tree for target k.

2) History-Based Baselines: We compare LightGBM against two low-overhead history-based baselines.

a) LastSeen: For entity e_i, the LastSeen prediction is

$$\hat{y}_i^{\text{LS}} = \text{Last}(i, e_i; y). \tag{20}$$

If no prior value exists, the training-period target median is used as the fallback prediction.

b) Exponential Moving Average: For each entity e and target k, let $s_i^{(k)}(e)$ denote the EMA state. Prediction uses the previous state:

$$\hat{y}_i^{\text{EMA},(k)} = s_{i-1}^{(k)}(e_i). \tag{21}$$

The state is updated after observing the current target:

$$s_i^{(k)}(e_i) = \begin{cases} y_i^{(k)}, & \text{if the state is uninitialized,} \\ \alpha y_i^{(k)} + (1-\alpha) s_{i-1}^{(k)}(e_i), & \text{otherwise.} \end{cases} \tag{22}$$

We use $\alpha = 0.3$.

3) Fixed LightGBM Settings: Table II reports the fixed LightGBM hyperparameters used in all experiments. The settings are held constant across targets to provide a controlled comparison against history-based baselines. We do not claim these hyperparameters are globally optimal; tuning and model selection are treated as limitations and future work.

E. Regime Definition and Evaluation Protocol

To avoid overly optimistic results under temporal dependence, we use time-ordered evaluation rather than random splitting, consistent with best practices for time-series evaluation [15]. All observations are sorted chronologically before preprocessing, history construction, and model evaluation.

1) Time-Ordered Split with Temporal Gap: Let $\rho \in (0, 1)$ be the training fraction and $\gamma \in (0, 1)$ be the temporal gap fraction. We define

$$n_{\text{train}} = \lfloor \rho N \rfloor, \tag{23}$$

$$n_{\text{gap}} = \lfloor (\rho + \gamma) N \rfloor. \tag{24}$$

The partitions are

$$\mathcal{D}_{\text{train}} = \{1, \ldots, n_{\text{train}}\}, \tag{25}$$

$$\mathcal{D}_{\text{gap}} = \{n_{\text{train}} + 1, \ldots, n_{\text{gap}}\}, \tag{26}$$

$$\mathcal{D}_{\text{test}} = \{n_{\text{gap}} + 1, \ldots, N\}. \tag{27}$$

All preprocessing statistics, including imputation values and one-hot vocabularies, are fit only on $\mathcal{D}_{\text{train}}$.

2) Occurrence-Index Buckets: We formalize workload recurrence using an occurrence index:

$$\text{occ}(i) = \sum_{j<i} \mathbb{I}\{e_j = e_i\}. \tag{28}$$

This counts how many times entity e_i appeared before observation i in the time-ordered stream. Within the test set, we evaluate four recurrence regimes:

$$\mathcal{B}_{K=1} = \{i \in \mathcal{D}_{\text{test}} : \text{occ}(i) = 0\}, \tag{29}$$

$$\mathcal{B}_{K<5} = \{i \in \mathcal{D}_{\text{test}} : \text{occ}(i) < 5\}, \tag{30}$$

$$\mathcal{B}_{K<20} = \{i \in \mathcal{D}_{\text{test}} : \text{occ}(i) < 20\}, \tag{31}$$

$$\mathcal{B}_{\text{warm}} = \{i \in \mathcal{D}_{\text{test}} : \text{occ}(i) \geq 20\}. \tag{32}$$

These buckets are used only for evaluation and interpretation. They allow us to compare true cold-start workloads, sparse-history workloads, transitional workloads, and warm recurring workloads. Unlike the earlier draft, we do not present the threshold rule as a fully evaluated scheduler-integrated hybrid policy. Instead, we use the buckets to identify where ML provides measurable forecasting value and where history-based predictors are the stronger default.

F. Evaluation Metrics

We evaluate all predictors using coefficient of determination (R^2) and mean absolute error (MAE) [19, 20]. For target k and evaluation subset $\mathcal{S}$, define

$$\bar{y}_{\mathcal{S}}^{(k)} = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} y_i^{(k)}. \tag{33}$$

Then

$$R^2_{\mathcal{S}}(k) = 1 - \sum_{i \in \mathcal{S}} \left(y_i^{(k)} - \hat{y}_i^{(k)} \right)^2 / \sum_{i \in \mathcal{S}} \left(y_i^{(k)} - \bar{y}_{\mathcal{S}}^{(k)} \right)^2. \tag{34}$$

MAE is defined as

$$\text{MAE}_{\mathcal{S}}(k) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left| y_i^{(k)} - \hat{y}_i^{(k)} \right|. \tag{35}$$

We report both metrics because they capture complementary behavior. R^2 measures explained variance within each regime, while MAE measures absolute error in the original target scale. This distinction is important because LightGBM may improve variance-based performance in sparse-history CPU regimes, while history-based predictors can still achieve lower absolute error once recurrence is established.

VI. Performance Evaluation

This section evaluates forecasting performance across different workload recurrence regimes. We compare three predictors:

TABLE III: R^2 performance across recurrence regimes.

Target	Model	$K=1$			$K<5$			$K<20$			$K\geq 20$			**Overall**		
		Mem	CPU	P95	Mem	CPU	P95	Mem	CPU	P95	Mem	CPU	P95	Mem	CPU	P95
R^2	LastSeen	-0.063	-0.128	-0.395	0.770	0.754	0.635	0.862	0.934	0.881	0.9969	0.9962	0.9962	0.972	0.981	0.980
	EMA	-0.063	-0.128	-0.395	0.770	0.754	0.635	0.861	0.933	0.877	0.9946	0.9926	0.9933	0.970	0.978	0.977
	LightGBM	0.070	0.751	0.383	0.297	0.937	0.824	0.430	0.978	0.935	0.9938	0.9947	0.9886	0.889	0.991	0.981

TABLE IV: MAE performance across recurrence regimes.

Target	Model	$K=1$			$K<5$			$K<20$			$K\geq 20$			**Overall**		
		Mem	CPU	P95	Mem	CPU	P95	Mem	CPU	P95	Mem	CPU	P95	Mem	CPU	P95
MAE	LastSeen	0.262	0.00889	0.01459	0.0809	0.00261	0.00431	0.0295	0.00096	0.00158	0.00064	0.00002	0.00003	0.00262	0.000085	0.00014
	EMA	0.262	0.00889	0.01459	0.0828	0.00264	0.00435	0.0318	0.00106	0.00175	0.00180	0.00006	0.00010	0.00385	0.00013	0.00021
	LightGBM	0.204	0.00647	0.01052	0.1022	0.00262	0.00424	0.0522	0.00147	0.00229	0.0109	0.00029	0.00054	0.0137	0.00037	0.00066

- **LastSeen**: predicts the most recent observed value for the same workload entity,
- **EMA**: an exponential moving average with smoothing factor $\alpha = 0.3$,
- **LightGBM**: a gradient-boosted tree model trained on scheduling-time features.

Performance is reported using R^2 and Mean Absolute Error (MAE) across four regimes: first occurrence ($K = 1$), early recurrence ($K < 5$), transition recurrence ($K < 20$), and warm recurring workloads ($K \geq 20$).

Tables III and IV show that forecasting performance is strongly regime-dependent. For true cold-start workloads ($K = 1$), history-based predictors have no prior entity observations and perform poorly, producing negative R^2 values across all targets. In contrast, LightGBM achieves substantially stronger cold-start performance, especially for CPU forecasting. It reaches $R^2 = 0.751$ for mean CPU demand and $R^2 = 0.383$ for p95 CPU demand, showing that scheduling-time features contain useful cross-entity predictive signal even when entity history is unavailable.

As limited history becomes available, history-based predictors rapidly improve. For $K < 5$, LastSeen and EMA already achieve strong memory-pressure performance ($R^2 \approx 0.77$), while LightGBM remains strongest for CPU mean and p95. This indicates that the cold-start ML advantage is most pronounced for CPU-related targets, while memory pressure benefits more quickly from even limited workload history.

In the transition regime ($K < 20$), both model classes remain competitive, but the results become target-dependent. LightGBM achieves the highest R^2 for CPU mean and CPU p95, while LastSeen and EMA produce lower MAE for several targets. This distinction is important because R^2 captures explained variance, whereas MAE reflects absolute prediction error in the original target scale.

For warm recurring workloads ($K \geq 20$), history-based predictors dominate. LastSeen achieves near-perfect R^2 values across all three targets and substantially lower MAE than LightGBM. The overall test-set results are therefore heavily influenced by the warm regime, since most observations belong to recurring workloads. Overall, LastSeen is strongest for memory pressure and remains highly competitive for CPU targets, while LightGBM achieves the best overall R^2 for mean CPU demand.

These results support a precise empirical conclusion: under leakage-safe evaluation, LightGBM is most useful in true cold-start settings, particularly for mean CPU and p95 CPU forecasting. However, simple history-based predictors rapidly become competitive as recurrence increases and dominate warm recurring workloads. Thus, the contribution of this paper is empirical and methodological rather than algorithmic: it identifies where ML materially helps and where strong history baselines are the better default.

VII. Conclusion

This paper studies leakage-safe resource demand forecasting for cloud scheduling. To avoid overly optimistic evaluation, we restrict predictors to scheduling-time information and past-only historical features, while using post-execution measurements only for target construction. We also use time-ordered and gap-based evaluation to reduce temporal and identity leakage. The results show a clear recurrence-dependent regime shift. LightGBM provides its strongest value in true cold-start settings, especially for mean CPU and p95 CPU demand, where no entity history is available. However, once workloads accumulate even modest history, LastSeen and EMA rapidly become competitive. For warm recurring workloads, LastSeen achieves near-optimal performance and consistently outperforms the learned model in MAE. Memory pressure is also more favorable to history-based prediction than to LightGBM once limited recurrence exists. These findings suggest that ML should not be treated as a universal replacement for simple forecasting baselines in production scheduling. Instead, ML is best understood as a cold-start forecasting tool, while lightweight history-based predictors are often preferable for recurring workloads. This work is limited to prediction accuracy on a single production trace and does not evaluate scheduler-level outcomes such as placement quality, overcommitment failures, resource waste, SLO violations, or job completion time. Future work should validate these regime-aware forecasting insights in scheduler-integrated simulations and under additional traces, larger temporal gaps, and entity-holdout evaluations.

Acknowledgment

This research was supported in part by the U.S. NSF under Grant DUE-2400459 and Grant DBI-2417643; in part by the NTIA/CMC Grant 12-09-C13055; in part by the title III grant; in part by the generous funding from the Cyber Policy Institute at Florida A&M University; and in part by the IBM Fellowship Award.

References

[1] S. Deng, H. Zhao, B. Huang, C. Zhang, F. Chen, Y. Deng, J. Yin, S. Dustdar, and A. Y. Zomaya. Cloud-native computing: A survey from the perspective of services. *Proceedings of the IEEE*, 112(1):12–46, 2024.

[2] 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. of NSDI*, Renton, 2022.

[3] N. Bashir, N. Deng, K. Rzadca, D. E. Irwin, S. Kodak, and R. Jnagal. Take it to the limit: Peak prediction-driven resource overcommitment in datacenters. In *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys)*, pages 556–573. Association for Computing Machinery (ACM), 2021.

[4] J. Liu, H. Shen, and L. Chen. CORP: Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems. In *Proc. of IEEE CLUSTER*, 2016.

[5] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *Proc. of OSDI*, 2021.

[6] P. Zheng, R. Pan, T. Khan, S. Venkataraman, and A. Akella. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In *Proc. of NSDI*, 2023.

[7] S. Rajasekaran, M. Ghobadi, and A. Akella. CASSINI: Network-aware job scheduling in machine learning clusters. In *Proc. of NSDI*, 2024.

[8] K. Xu, D. Sun, H. Tian, J. Zhang, and K. Chen. GREEN: Carbon-efficient resource scheduling for machine learning clusters. In *Proc. of NSDI*, 2025.

[9] Z. Jiang, T. Zhao, C. xia, and A. Zomaya. Atlas: Alibaba dataset and benchmark for learning-augmented scheduling. In *Proc. of ICLR*, 2026.

[10] C. Yang, Y. Li, M. Maas, M. Uysal, U. U. Hafeez, A. Merchant, and R. McDougall. A bring-your-own-model approach for ml-driven storage placement in warehouse-scale computers. In *Proc. of MLSys*, 2025.

[11] S. Kapoor and A. Narayanan. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9), 2023.

[12] Y. Diao, D. Horn, A. Kipf, O. Shchur, I. Benito, W. Dong, D. Pagano, P. Pfeil, V. Nathan, B. Narayanaswamy, and T. Kraska. Forecasting algorithms for intelligent resource scaling: An experimental analysis. In *Proceedings of the 2024 ACM Symposium nn Cloud Computing (SoCC)*, pages 126–143, 2024.

[13] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proc. of ACM SOSP*, Shanghai, 2017.

[14] A. Rossi, A. Visentin, D. Carraro, S. Prestwich, and K. N. Brown. Forecasting workload in cloud computing: Towards uncertainty-aware predictions and transfer learning. *Cluster Computing*, 2025.

[15] C. Bergmeir, R. J. Hyndman, and B. Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 2018.

[16] J. Wilkes. ClusterData 2019 traces. https://github.com/google/cluster-data/blob/master/ClusterData2019.md [accessed in May 2026].

[17] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[18] scikit-learn developers. MultiOutputRegressor — scikit-learn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html [accessed in May 2026].

[19] scikit-learn developers. r2_score — scikit-learn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html [accessed in May 2026].

[20] scikit-learn developers. mean_absolute_error — scikit-learn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html [accessed in May 2026].