Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.086441

ARTICLE

CASH: Confidence-Calibrated Deep Feature Crossing for Classification-Aware Heterogeneous Task Scheduling

Chuanlin Jian[1], **Yuanchen Sun**[2], **Xiangcheng Liu**[1], **Xuming Huang**[3], **Samaneh Beheshti Kashi**[4] and **Xing Hu**[1,*]

[1]School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai, China
[2]School of Artificial Intelligence, Shenyang Normal University, Shenyang, China
[3]Department of Computer Sciences, University of Wisconsin–Madison, 1210 W Dayton Street, Madison, WI, USA
[4]Zhejiang Engineering Research Center of Interventional Medicine Engineering and Biotechnology, The Fifth Affiliated Hospital of Wenzhou Medical University, Lishui, China

*Corresponding Author: Xing Hu. Email: huxing@usst.edu.cn

Received: 30 May 2026; Accepted: 22 July 2026; Published: 15 September 2026

ABSTRACT: Heterogeneous clusters now carry most artificial-intelligence, scientific-computing, cloud, and edge-assisted workloads, yet scheduling them well remains difficult: workload semantics, hardware capability, queue state, and energy behavior are tightly coupled. Existing schedulers, whether heuristic, learning-based, or built on reinforcement learning, tend to rely on coarse resource requests, overlook the compatibility between task types and node classes, or carry a heavy training and deployment cost. This paper presents Classification-Aware Scheduling for Heterogeneous clusters (CASH), a confidence-calibrated, classification-aware scheduling framework that integrates workload profiling, deep feature crossing, gradient-boosted boundary modeling, and risk-aware heterogeneous resource mapping. CASH constructs a multi-view workload representation from resource requests, submission context, queue attributes, and historical user behavior. A Deep & Cross Network captures bounded-degree high-order feature interactions; an eXtreme Gradient Boosting (XGBoost) branch models the sharp decision boundaries typical of tabular scheduling logs. A validation-calibrated, confidence-aware fusion rule combines the two branches into task-type probabilities with attached uncertainty estimates. The scheduler then performs class-constrained candidate selection and chooses execution nodes by minimizing a composite score over predicted execution time, energy, load imbalance, semantic mismatch, and prediction uncertainty. In trace-driven simulations on Massachusetts Institute of Technology (MIT) Supercloud logs, CASH improves resource matching, response time, makespan, and energy consumption over representative baselines while keeping load balance competitive. The results indicate that an explicit, calibrated link between workload semantics and hardware capability is a practical basis for efficient and interpretable cluster scheduling.

KEYWORDS: Heterogeneous computing; task scheduling; workload classification; deep & cross network; XGBoost; ensemble learning; uncertainty calibration; energy-aware scheduling

1 Introduction

The rapid growth of artificial intelligence (AI), cloud services, high-performance computing (HPC), and edge-assisted applications has reshaped how computing clusters are built. Modern infrastructures combine graphics processing units (GPUs), high-frequency central processing units (CPUs), standard batch-processing nodes, and isolation-oriented nodes with different performance, energy, and reliability profiles. Large-scale systems such as Borg and Kubernetes have demonstrated the importance of cluster-level resource

management in production environments [1,2], while heterogeneity-aware schedulers such as Paragon and Gavel further show that performance depends strongly on matching workloads with suitable hardware classes [3,4]. A scheduler in such a system is no longer a queue allocator. It has to infer what a task actually is, judge which hardware class can serve it, and make online allocation decisions under shifting load and energy constraints.

Three gaps remain in prior scheduling work. First, many scheduling policies mainly exploit explicit resource requests such as CPU cores, memory, and GPU count, whereas the latent semantics of a task, e.g., whether it is AI-intensive, CPU-intensive, lightweight, or abnormal, receive little modeling attention. Second, deep-learning schedulers capture nonlinear dependencies but usually flatten structured log fields into dense vectors and thus underuse explicit high-order feature crosses; tree-based models handle tabular thresholds well but are less expressive for complex interactions. Third, reinforcement-learning schedulers, represented by DeepRM and Decima [5,6], can optimize sequential decisions, but their training cost, sensitivity to simulation fidelity, and deployment complexity can stand in the way of practical adoption.

This paper revisits heterogeneous task scheduling from a workload-semantic perspective. The central hypothesis is simple: identify the semantic class of each workload first, then map the predicted class to a compatible node pool. To implement this idea, we propose *Classification-Aware Scheduling for Heterogeneous clusters* (CASH), a classification-aware heterogeneous scheduler based on confidence-calibrated deep feature crossing and ensemble learning, which explicitly separates two coupled problems: (i) robust workload-type inference from heterogeneous scheduling logs, and (ii) risk-aware allocation among candidate nodes that match the predicted class. Because CASH consumes only standard scheduling logs and node-state statistics, it stays modular and interpretable and can be attached to existing cluster-management pipelines without intrusive changes.

Fig. 1 illustrates the problem this design targets. A semantics-blind scheduler may send heavy tasks to weak nodes, or lightweight tasks to scarce accelerators, and pay for it in resource mismatch, tail latency, wasted energy, and load hotspots. CASH instead infers workload semantics first and performs confidence-aware resource mapping afterward.

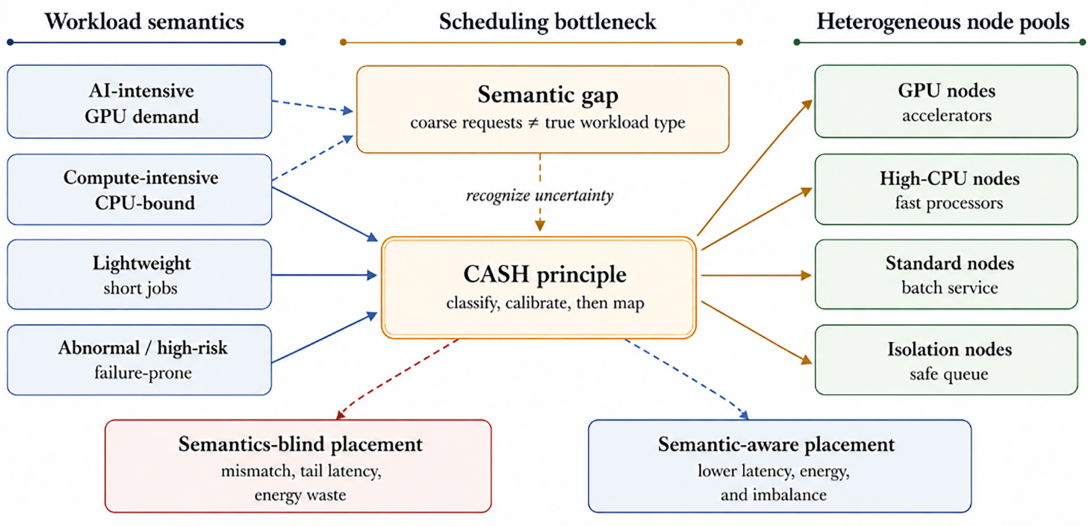

Figure 1: Motivation for classification-aware heterogeneous task scheduling: bridging latent workload semantics and heterogeneous hardware capability under latency, energy, load-balance, and uncertainty considerations.

The main contributions are summarized as follows.

(1) **A classification-aware scheduling formulation.** Heterogeneous scheduling is formulated as a constrained multi-objective optimization jointly considering latency, makespan, energy, load imbalance, and semantic task-node compatibility.

(2) **A deep-shallow workload classifier.** A dual-branch predictor combines a Deep & Cross Network (DCN) for explicit high-order feature interaction learning with eXtreme Gradient Boosting (XGBoost) for structured boundary discrimination, improving robustness on mixed numerical and categorical scheduling logs.

(3) **A confidence-calibrated fusion and allocation strategy.** Simple probability averaging is extended with validation-calibrated reliability and prediction uncertainty, and a risk-aware node scoring function is introduced for online resource mapping.

(4) **A reproducible trace-driven evaluation protocol.** Based on Massachusetts Institute of Technology (MIT) Supercloud logs, a heterogeneous-cluster simulation pipeline evaluates CASH against heuristic, learning-based, sequence-prediction, and cloud-edge baselines across resource matching, latency, makespan, energy, and load balance.

2 Related Work

2.1 Cluster Scheduling and Heterogeneity-Aware Resource Management

Production cluster schedulers aim to improve utilization while satisfying application-level requirements. Early and influential systems established different scheduler architectures, including two-level resource sharing in Mesos [7], shared-state parallel scheduling in Omega [8], large-scale production management in Borg [1], and container-oriented orchestration summarized through Borg, Omega, and Kubernetes [2]. Resource fairness and placement quality have been studied through dominant-resource fairness [9] and fast centralized flow-based optimization in Firmament [10]; taken together, these systems show that scalable scheduling has to balance policy modularity, placement latency, fairness, and utilization at the same time.

Heterogeneous and multi-resource clusters further require workload-aware placement decisions. Tetris improves cluster efficiency by packing tasks according to multi-resource requirements [11], and Graphene addresses jobs with both dependency structures and heterogeneous resource demands [12]. Paragon and Quasar demonstrate that platform heterogeneity and interference can be exploited to improve quality of service (QoS) and utilization in datacenters [3,13]. Gavel generalizes heterogeneity-aware scheduling for deep-learning workloads through effective throughput [4], and FRESH provides fault-tolerant real-time scheduling on heterogeneous multiprocessor platforms through a semi-partitioned technique with backup copies and selective processor sleep states [14]. CASH sits in the same line of work but differs in the signal it exploits and in how that signal reaches the allocation objective. Semantic- and QoS-aware schedulers such as Paragon and Quasar characterize interference or QoS sensitivity implicitly, as an internal ranking heuristic, whereas CASH makes the workload-type signal explicit as a calibrated, uncertainty-annotated class label that enters the constrained objective directly, gates the candidate pool through a compatibility threshold, and modulates the risk-aware node score through its predictive uncertainty. The contribution is therefore less the dual-branch classifier itself than the coupling of a calibrated semantic class, together with its uncertainty, to a multi-objective allocation rule that stays interpretable and deploys as a plug-in module.

2.2 Learning-Based and Deep-Learning Cluster Scheduling

Machine learning has been introduced into cluster scheduling to overcome the limitations of manually designed heuristics. Classical studies established that CPU and cluster scheduling should be optimized jointly with power consumption [15,16], and recent surveys show that energy-aware cloud and edge-cloud scheduling remains challenging under heterogeneous objectives [17,18]. DeepRM formulates resource management as a deep reinforcement-learning problem [5], Decima learns scheduling policies for directed-acyclic-graph (DAG)-structured data-processing jobs [6], and hybrid optimization has been used for multi-objective heterogeneous cloud scheduling [19]. Recent studies further explore deep neural network (DNN)-based energy-efficient cloud-edge collaboration [20] and meta-reinforcement-learning load balancing [21], while heterogeneity-aware cluster schedulers such as Sia [22] and Hops [23] adapt job placement to heterogeneous accelerator pools through goodput modeling and fine-grained hardware sensing.

Deep-learning (DL) clusters introduce additional constraints, including GPU locality, gang scheduling, unknown training duration, and resource-performance coupling. Optimus dynamically models resource-performance relationships for deep-learning jobs [24]; Gandiva uses introspective scheduling to exploit domain-specific properties of DL training [25]; Tiresias designs GPU scheduling policies for distributed DL jobs without accurate completion-time information [26]; and Pollux co-optimizes per-job goodput and cluster-wide scheduling decisions [27]. More recently, GPARS predicts job duration with graph attention networks to allocate suitable GPU types in heterogeneous clusters [28], AISAW adaptively mitigates runtime interference for deep-learning training on distributed heterogeneous systems [29], and multi-resource interleaving with deep reinforcement learning improves task placement in cloud-edge systems [30]; surveys catalogue the design space of learning-based scheduling in GPU datacenters [31]. CASH targets a broader heterogeneous-task setting than these systems and instead builds a supervised, calibration-aware semantic-classification layer that slots in ahead of conventional scheduling or ranking modules.

2.3 Feature Crossing, Tree-Based Boundary Modeling, and Calibration

Scheduling logs are mixed-type tabular data with numerical resource requests, categorical queue fields, temporal features, and user-history statistics. Wide & Deep learning demonstrates the value of combining memorization through cross-product features with neural generalization [32], while DeepFM and AutoInt show that low-order, high-order, and attention-based feature interactions can be learned automatically from sparse or mixed fields [33,34]. DCN efficiently learns bounded-degree feature interactions without exhaustive manual feature engineering [35], and DCN-V2 improves its expressiveness for web-scale learning-to-rank systems [36]; this line of work motivates the explicit feature crossing that CASH applies to scheduling-log representation.

Tree-based boosting methods are also well suited to structured scheduling records. XGBoost is a scalable regularized tree-boosting framework based on second-order optimization [37], and LightGBM improves gradient-boosted decision trees through gradient-based sampling and feature bundling [38]. CASH combines these complementary families: DCN captures cross-field interactions such as *partition* × *GPU request* ×*historical failure*, whereas XGBoost captures threshold-like splits such as memory demand or failure-rate boundaries. Because CASH passes predicted probabilities to the scheduling layer, confidence calibration is also important. Prior work shows that probability estimates from supervised models, including modern neural networks, are often miscalibrated and need correction before they can drive decisions [39–41], and recent surveys review the state of the art in deep-learning calibration and uncertainty estimation, including the expected calibration error (ECE) and reliability-diagram diagnostics used in this work [42]. Accordingly, CASH uses validation-calibrated fusion and uncertainty-aware scheduling scores instead of relying only on hard class labels.

3 Problem Formulation

3.1 System Model

Consider a heterogeneous cluster with M nodes, denoted by $\mathcal{N} = \{n_1, n_2, \ldots, n_M\}$. Each node belongs to one of four functional pools, $\mathcal{R} = \{R_{GPU}, R_{HighCPU}, R_{Std}, R_{Iso}\}$, where R_{GPU} denotes accelerator-oriented nodes, $R_{HighCPU}$ high-performance CPU nodes, R_{Std} standard batch-processing nodes, and R_{Iso} isolation nodes for abnormal or high-risk tasks. Let $\boldsymbol{q}_j(t)$ denote the real-time queue state of node n_j, and let $\boldsymbol{a}_j$ denote its resource-capacity vector, including CPU capacity, memory capacity, GPU availability, and power parameters.

At a scheduling epoch, the pending task set is $\mathcal{T} = \{t_1, t_2, \ldots, t_K\}$. Each task is represented by a feature vector $\boldsymbol{x}_i = [\boldsymbol{x}_i^{res}, \boldsymbol{x}_i^{ctx}, \boldsymbol{x}_i^{hist}, \boldsymbol{x}_i^{time}]$, where $\boldsymbol{x}_i^{res}$ contains resource requests, $\boldsymbol{x}_i^{ctx}$ contains contextual fields such as user and partition, $\boldsymbol{x}_i^{hist}$ contains historical behavior statistics, and $\boldsymbol{x}_i^{time}$ contains temporal submission features.

3.2 Task-Type Semantics and Matching Matrix

Let the latent task class be $c_i \in \mathcal{C}$, where $\mathcal{C} = \{\text{AI}, \text{Compute}, \text{Light}, \text{Abnormal}\}$. The compatibility between a task class c and a node pool r is encoded by a matching matrix $\boldsymbol{A} \in [0,1]^{|\mathcal{C}|\times|\mathcal{R}|}$. A larger value $A_{c,r}$ indicates better semantic compatibility. For example, AI-intensive tasks are highly compatible with R_{GPU}, whereas abnormal tasks should be routed to R_{Iso} for isolation.

3.3 Decision Variables and Constraints

Let $z_{ij} \in \{0,1\}$ indicate whether task t_i is assigned to node n_j. The allocation matrix $\boldsymbol{Z} = [z_{ij}]_{K\times M}$ must satisfy

$$\sum_{j=1}^{M} z_{ij} = 1, \quad \forall i, \tag{1}$$

$$\sum_{i=1}^{K} z_{ij}\boldsymbol{d}_i \preceq \boldsymbol{a}_j, \quad \forall j, \tag{2}$$

$$z_{ij} = 0 \quad \text{if } A_{\hat{c}_i, r(j)} < \eta, \tag{3}$$

where $\boldsymbol{d}_i$ denotes the resource-demand vector of task t_i, $r(j)$ denotes the pool to which node n_j belongs, $\hat{c}_i$ is the predicted task class, and η is a compatibility threshold.

3.4 Execution Time, Energy, and Load-Balance Models

The predicted execution time of task t_i on node n_j is modeled as

$$\widehat{T}_{ij} = \frac{B_i}{s_{j,\hat{c}_i}} + Q_j(t), \tag{4}$$

where B_i denotes the normalized base workload, $s_{j,\hat{c}_i}$ is the class-dependent processing speed of node n_j, and $Q_j(t)$ is the estimated queue waiting time. The makespan and the average response time are

$$T_{max} = \max_j \sum_{i=1}^{K} z_{ij}\widehat{T}_{ij}, \qquad \bar{T} = \frac{1}{K}\sum_{i=1}^{K}\sum_{j=1}^{M} z_{ij}\widehat{T}_{ij}. \tag{5}$$

The total energy is estimated by active and idle power terms,

$$E_{total} = \sum_{j=1}^{M} \left(P_j^{act} T_j^{busy} + P_j^{idle} \left(T_{max} - T_j^{busy} \right) \right), \tag{6}$$

where $T_j^{busy} = \sum_i z_{ij} \widehat{T}_{ij}$. Load imbalance is measured by the standard deviation of normalized node workloads, $\mathcal{B} = \left(\frac{1}{M} \sum_{j=1}^{M} (\rho_j - \bar{\rho})^2 \right)^{1/2}$, where ρ_j is the utilization ratio of node n_j and $\bar{\rho}$ is the average utilization.

The models in Eqs. (4)–(6) are deliberately parameterized rather than measured: $s_{j,\hat{c}_i}$ is a class-dependent effective speed and P_j^{act}, P_j^{idle} are per-node average power levels. This keeps the scheduler analytically tractable and dataset-agnostic, but does not resolve fine-grained dynamic voltage and frequency scaling (DVFS) or co-location interference, which make true node speed and power time-varying; learning-based DVFS controllers in embedded systems [43] and DVFS-aware DAG schedulers in cloud data centers [44] model these effects explicitly at the cost of additional state and per-core control. In CASH they are folded into the class-conditional averages, and the resulting approximation error is discussed in Section 6.12.

3.5 Multi-Objective Scheduling Objective

The scheduling objective is formulated as

$$\min_{\boldsymbol{Z}} \ \mathcal{J} = \lambda_1 \widehat{\widehat{T}} + \lambda_2 \widehat{T}_{max} + \lambda_3 \widehat{E}_{total} + \lambda_4 \widehat{\mathcal{B}} + \lambda_5 \mathcal{M}, \tag{7}$$

subject to Eqs. (1)–(3). The hat over each metric denotes min-max normalization within a scheduling batch, and $\mathcal{M} = \frac{1}{K} \sum_{i=1}^{K} \sum_{j=1}^{M} z_{ij} \left(1 - A_{\hat{c}_i, r(j)}\right)$ is the semantic mismatch penalty. The formulation encodes the central design principle of CASH: task-node semantic compatibility enters the objective as an optimization term of its own, rather than serving as a post-hoc explanation of allocation results.

Remark 1 (online tractability and the greedy surrogate). Problem (7) is a constrained multi-objective assignment problem over the binary matrix $\boldsymbol{Z}$; even its single-objective makespan special case (minimum makespan on unrelated machines with capacity constraints) is NP-hard, so computing a globally optimal $\boldsymbol{Z}$ at every epoch is intractable for online dispatching. CASH therefore applies a greedy per-task surrogate: previously placed tasks are fixed, and the current task is assigned to the feasible node minimizing the composite score S_{ij} of Eq. (11), whose terms mirror the batch objective with an added uncertainty penalty, the weights $\boldsymbol{\beta}$ playing the role of $\boldsymbol{\lambda}$ at the single-assignment level. This yields amortized $O(1)$ scoring per candidate and matches the one-task-at-a-time behavior of production dispatchers, but provides no global optimality guarantee: the surrogate trades optimality for low decision latency, and the experiments quantify how much quality survives this trade-off relative to the baselines.

4 The Proposed CASH Framework

4.1 Overview

CASH contains four stages: workload profiling, dual-branch class prediction, confidence-calibrated probability fusion, and risk-aware node selection; Fig. 2 illustrates the workflow. The classifier provides both a task class and a confidence estimate, which the scheduler uses to restrict the search space and rank candidate nodes with respect to latency, energy, load, compatibility, and uncertainty.

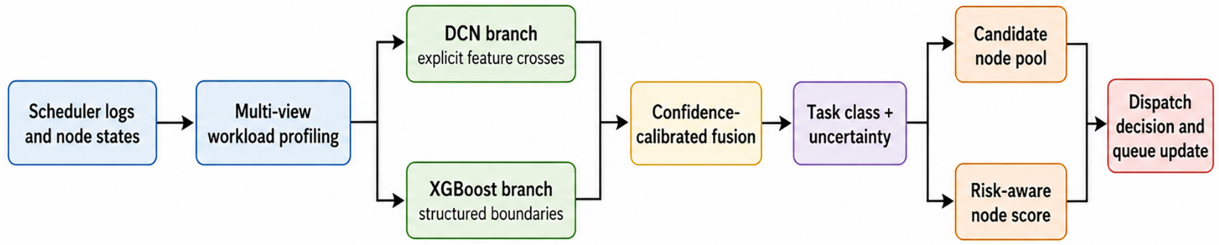

Figure 2: Overview of the CASH framework: the scheduling layer receives confidence and uncertainty rather than hard labels alone.

4.2 Workload Profiling and Feature Engineering

Continuous fields, such as CPU count, memory demand, requested runtime, and GPU demand, often exhibit heavy-tailed distributions, so we apply logarithmic smoothing followed by training-set standardization, $\widetilde{x}^{num} = \big(\ln(1+x^{num}) - \mu_{train}\big)/(\sigma_{train}+\varepsilon)$, where ε prevents numerical instability. Categorical fields, such as user, account, partition, and job state, are embedded into dense representations. User-history features are computed in a sliding window $W_u(t)$: the failure rate $f_{u,t}^{fail} = \frac{1}{|W_u(t)|}\sum_{k\in W_u(t)} \mathbb{I}(state_k \in \mathcal{S}_{fail})$, the submission count $f_{u,t}^{cnt} = |W_u(t)|$, and the resource-request variance $f_{u,t}^{var} = \mathrm{Var}\big(\{\boldsymbol{x}_k^{res} \mid k \in W_u(t)\}\big)$. The final input is $\boldsymbol{x}_i = [\widetilde{\boldsymbol{x}}_i^{num}; \boldsymbol{e}_i^{cat}; \boldsymbol{x}_i^{hist}; \boldsymbol{x}_i^{time}]$.

4.3 DCN Branch for Explicit Feature Interaction

The DCN branch contains a cross network and a deep network. Given input $\boldsymbol{x}_0$, the cross network is

$$\boldsymbol{x}_{l+1} = \boldsymbol{x}_0(\boldsymbol{x}_l^\top \boldsymbol{w}_l) + \boldsymbol{b}_l + \boldsymbol{x}_l, \quad l = 0, \ldots, L-1. \tag{8}$$

This residual form increases interaction order layer by layer while keeping the parameter count controlled. In parallel, the deep branch is $\boldsymbol{h}_{m+1} = \phi(\boldsymbol{W}_m \boldsymbol{h}_m + \boldsymbol{b}_m)$ with $\boldsymbol{h}_0 = \boldsymbol{x}_0$, and the DCN probability is $\boldsymbol{p}_i^{DCN} = \mathrm{softmax}\,(\boldsymbol{W}_o[\boldsymbol{x}_L; \boldsymbol{h}_M] + \boldsymbol{b}_o)$.

4.4 XGBoost Branch for Structured Decision Boundaries

The XGBoost branch models tabular thresholds and interactions through additive regression trees, $\hat{y}_i = \sum_{k=1}^{K_T} f_k(\boldsymbol{x}_i)$ with $f_k \in \mathcal{F}$, trained with the standard regularized second-order boosting objective of XGBoost [37], which fits each tree to first- and second-order gradients under leaf-number and leaf-weight penalties. The XGBoost probability is denoted as $\boldsymbol{p}_i^{XGB}$.

4.5 Confidence-Calibrated Probability Fusion

A fixed fusion coefficient is easy to implement but may be unstable under distribution shift. CASH therefore uses validation-calibrated reliability and entropy-based confidence. For model $m \in \{DCN, XGB\}$, define

$$\kappa_m(\boldsymbol{x}_i) = \exp[-\tau \mathrm{H}(\boldsymbol{p}_i^m)] \cdot \frac{1}{\varepsilon + ECE_m}, \tag{9}$$

where ECE_m is the ECE of model m estimated on the validation set, and τ controls entropy sensitivity. The normalized fusion weight and the fused class probability are

$$\alpha_i = \frac{\kappa_{DCN}(\boldsymbol{x}_i)}{\kappa_{DCN}(\boldsymbol{x}_i) + \kappa_{XGB}(\boldsymbol{x}_i)}, \qquad \boldsymbol{p}_i = \alpha_i \boldsymbol{p}_i^{DCN} + (1-\alpha_i)\boldsymbol{p}_i^{XGB}. \tag{10}$$

The predicted class and uncertainty are $\hat{c}_i = arg\,max_c\, p_{i,c}$ and $u_i = \mathrm{H}(\boldsymbol{p}_i)$.

4.6 Risk-Aware Candidate Node Selection

For task t_i, the candidate pool is $\Omega_i = \{n_j \in \mathcal{N} \mid A_{\hat{c}_i, r(j)} \geq \eta,\ \boldsymbol{d}_i \preceq \boldsymbol{a}_j^{remain}\}$. If Ω_i is empty, CASH relaxes the compatibility threshold and applies a safety penalty rather than forcing the task into an incompatible node. For each candidate n_j, CASH computes

$$S_{ij} = \beta_1 \widehat{T}_{ij} + \beta_2 \widehat{E}_{ij} + \beta_3 \widehat{\rho}_j + \beta_4 (1 - A_{\hat{c}_i, r(j)}) + \beta_5 u_i. \tag{11}$$

The selected node is $n_i^* = arg\,min_{n_j \in \Omega_i} S_{ij}$. Under this score, the scheduler behaves conservatively when classification confidence is low and commits more aggressively when semantic matching is reliable.

4.7 Algorithmic Procedure and Complexity

The end-to-end scheduling procedure of CASH is summarized in Table 1: for each pending task, the dual-branch classifier produces a fused, calibrated class distribution, and the scheduler restricts the candidate pool by semantic compatibility and residual capacity before selecting the node with the smallest risk-aware score.

Table 1: CASH: confidence-calibrated classification-aware heterogeneous scheduling.

Line	Procedure
1	Input pending tasks $\mathcal{T}$, node set $\mathcal{N}$, node pools $\mathcal{R}$, matching matrix $\boldsymbol{A}$, trained DCN and XGBoost models, and weights $\boldsymbol{\beta}$; initialize $\boldsymbol{Z} \leftarrow \boldsymbol{0}$ and read real-time node states.
2	**for** each task $t_i \in \mathcal{T}$ **do**
3	Extract resource, context, temporal, and user-history features; apply log smoothing, standardization, categorical embedding, and sliding-window statistics.
4	Obtain $\boldsymbol{p}_i^{DCN}$ and $\boldsymbol{p}_i^{XGB}$; compute reliability weights by Eq. (9) and the fused probability by Eq. (10); predict class $\hat{c}_i$ and uncertainty u_i.
5	Construct candidate pool Ω_i from semantic compatibility and residual resource capacity.
6	**for** each candidate $n_j \in \Omega_i$ **do** estimate execution time, energy, load state, mismatch penalty, and risk score S_{ij} **end for**
7	Select node $n_i^* = arg\,min_{n_j \in \Omega_i} S_{ij}$; set $z_{i,n_i^*} = 1$ and update the queue state and remaining capacity of n_i^*.
8	**end for**
9	Return allocation matrix $\boldsymbol{Z}$ and updated node states.

For each task, DCN inference costs $O(Ld + \sum_m d_m d_{m+1})$, where d is the input dimension and d_m is the width of the m-th deep layer. XGBoost inference costs $O(K_T D_T)$, where K_T is the number of trees and D_T is the tree depth. Candidate scoring costs $O(|\Omega_i|)$. Therefore, the online scheduling complexity for a batch of K tasks is approximately $O\big(K\big(Ld + \sum_m d_m d_{m+1} + K_T D_T + |\Omega|\big)\big)$, which is practical because $|\Omega| \le M$ and candidate filtering usually reduces the search space. The corresponding *measured* per-stage decision cost (feature extraction, DCN inference, XGBoost inference, confidence fusion, and candidate ranking), together with the end-to-end scheduling throughput, is reported in Section 6.9.

5 Experimental Design

5.1 Dataset and Trace-Driven Simulation

The experiments are based on the MIT Supercloud dataset, which provides anonymized scheduler logs, CPU and GPU time-series data, and node-monitoring data for studying large-scale HPC and datacenter operations [45]. The original logs are cleaned by removing invalid records, normalizing inconsistent states, filtering fields with excessive missingness, and constructing task samples from scheduling events. The simulation is trace-driven rather than an online production deployment: arrivals, resource requests, and task states are replayed from real logs, while heterogeneous node pools and energy parameters are configured to evaluate policies under controlled conditions.

Task labels are constructed from fields such as *JobName*, *Partition*, resource requests, and *State*, yielding four categories: AI-intensive, compute-intensive, lightweight, and abnormal. To reduce label leakage, fields that directly encode final outcomes are excluded from the feature set when they would not be available at scheduling time.

5.2 Experimental Platform and Implementation

The implementation uses Python 3.10, PyTorch 2.1, and XGBoost 2.0, running on a workstation with an Intel Core i9 processor and an NVIDIA RTX 4090 GPU. Continuous features are standardized using training-set statistics only; categorical embeddings are learned jointly with the DCN branch; the validation set is used for hyperparameter selection, fusion calibration, and early stopping; and the test set is used only for final reporting. The full trace-driven simulation and implementation settings are summarized in Table 2.

Table 2: Trace-driven simulation and implementation settings.

Item	Setting
Data source	MIT Supercloud scheduler logs: job metadata, resource requests, job states, CPU/GPU time series, node monitoring.
Simulation type	Trace-driven replay; arrivals, requests, and state transitions from logs; node pools and power parameters configured for controlled comparison.
Workload scales	1000–5000 tasks in steps of 1000, to evaluate scalability trends.
Task categories	AI-intensive, compute-intensive, lightweight, and abnormal/high-risk.
Node pools	GPU-accelerated, high-performance CPU, standard compute, and secure isolation nodes.
Feature groups	Resource requests, submission context, queue/partition attributes, temporal attributes, sliding-window user history.
Software stack	Python 3.10, PyTorch 2.1, XGBoost 2.0.

(Continued)

Table 2 (continued)

Item	Setting
Hardware platform	Workstation with an Intel Core i9 CPU and an NVIDIA RTX 4090 GPU.
Data split usage	Training for model fitting; validation for early stopping and fusion calibration; test for final reporting only.

To support statistical claims, and unless stated otherwise, every reported metric is the mean over 10 independent runs with different random seeds and re-sampled task streams. The associated standard deviations and paired-test significance against the strongest baseline are reported in Section 6.10.

5.3 Baselines

Five baselines are considered: **Random**, which assigns tasks to feasible nodes uniformly at random; **TSSAC**, a reinforcement-learning baseline representing policy-learning methods; **CECF**, a cloud-edge collaboration baseline emphasizing energy-aware workload allocation; **LSTM**, a long short-term memory (LSTM) sequence-prediction baseline using temporal workload patterns; and **Meta-RHDC**, a metaheuristic/reinforcement-learning hybrid baseline for dynamic load balancing. These baselines cover heuristic randomization, sequential prediction, reinforcement learning, cloud-edge collaboration, and hybrid optimization, and are aligned with recent cloud-edge collaboration, heterogeneity-aware cluster scheduling, and metaheuristic load-balancing studies [20–23].

5.4 Evaluation Metrics

The evaluation includes five metrics: **resource matching accuracy**, the proportion of tasks assigned to semantically compatible node pools (higher is better); **average response time**, the average completion delay including estimated waiting and execution time; **makespan**, the maximum completion time across all nodes; **total energy consumption**, active plus idle energy estimated by node-level power parameters; and **load standard deviation**, the dispersion of normalized workloads across nodes (for the latter four, lower is better). In the normalized comparison, latency, energy, and error rate are also reported relative to the random baseline (lower is better) for cross-metric comparison.

5.5 Ablation, Sensitivity, Robustness, Calibration, and Overhead Protocol

Beyond the cross-method comparison, the evaluation includes five families of controlled experiments that cover component contribution, design sensitivity, robustness to classifier failure, confidence quality, and online cost. All experiments reuse the trace-driven pipeline and the settings of Table 2; only the studied factor is varied.

(P1) Component ablation. Each core component is removed in turn while all other settings are held fixed: the DCN branch (explicit high-order feature crossing), the XGBoost branch (structured boundary modeling), the confidence-calibrated fusion, the semantic candidate constraint, and the energy and load-balance terms of the node score. The resulting scheduling metrics are reported in Section 6.5.

(P2) Hyperparameter sensitivity. The compatibility threshold η in Eq. (3), the entropy-sensitivity coefficient τ in Eq. (9), and the node-score weights $\beta_1, \ldots, \beta_5$ in Eq. (11) are each swept over a grid around their default values, with all other factors fixed, to identify the operating range in which CASH is stable (Section 6.8).

(P3) Robustness to classification failure. Because the scheduler consumes predicted classes, we stress-test it under three degradations: (i) synthetic label noise injected into the classifier output at controlled rates; (ii) concept drift, realized by shifting the test-stream class mix away from the training proportions; and (iii) unseen applications, realized by holding out selected job families so that their per-application history features fall back to the global prior. Both classification and downstream scheduling quality are reported (Section 6.7).

(P4) Confidence calibration. The fused confidence is evaluated with the expected calibration error (ECE), maximum calibration error (MCE), and multiclass Brier score on the held-out test set, with reliability diagrams before and after validation calibration (Section 6.6).

(P5) Scheduling overhead. We profile the wall-clock cost of each decision stage, the total per-task decision latency, and the sustained scheduling throughput on the platform of Table 2 (Section 6.9).

In addition, statistical significance is assessed by repeating every configuration over multiple seeds and applying a paired test between CASH and the strongest baseline (Section 6.10).

6 Results and Discussion

6.1 Resource Matching Accuracy

Fig. 3a compares resource matching accuracy under different task scales: CASH holds the highest and most stable accuracy as the number of tasks grows, which supports the underlying assumption: a classifier that combines explicit feature crossing with structured boundary learning gives the mapper a more reliable semantic signal than any single-route policy. The wide margin over random allocation further shows that feasibility checking alone is no substitute for semantic compatibility. Table 3 reports the corresponding numerical values of all five metrics across the tested workload scales.

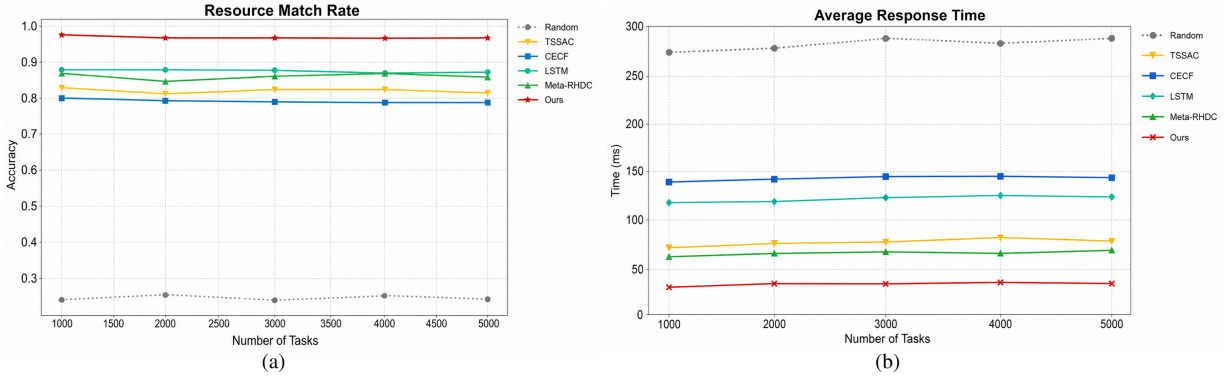

Figure 3: Comparison of scheduling quality among all methods under different numbers of tasks. (**a**) Resource matching accuracy; (**b**) average response time.

Table 3: Numerical performance of CASH under different workload scales. The values correspond to the plotted trace-driven simulation results.

Number of Tasks	Match Accuracy	Avg. Response Time (ms)	Makespan (ms)	Energy (J)	Load Std. Dev.
1000	0.980	33	20,000	0.06×10^6	7000
2000	0.972	38	40,000	0.14×10^6	11,000
3000	0.972	37	52,000	0.22×10^6	13,500

(Continued)

Table 3 (continued)

Number of Tasks	Match Accuracy	Avg. Response Time (ms)	Makespan (ms)	Energy (J)	Load Std. Dev.
4000	0.970	39	68,000	0.29×10^6	18,000
5000	0.971	38	85,000	0.36×10^6	21,500

6.2 Average Response Time and Makespan

Fig. 3b shows that CASH delivers the lowest average response time across all tested loads. Two mechanisms account for this: class-aware candidate selection keeps demanding tasks off underpowered nodes, and the online node score reflects current queue load, which cuts avoidable waiting. Fig. 4a further shows that CASH shortens the longest completion time as well, so the gains come from mitigating tail accumulation rather than from shifting the mean alone. Table 4 lists the full cross-method comparison at the largest, 5000-task scale, where CASH attains the best matching accuracy, response time, makespan, and energy among all methods.

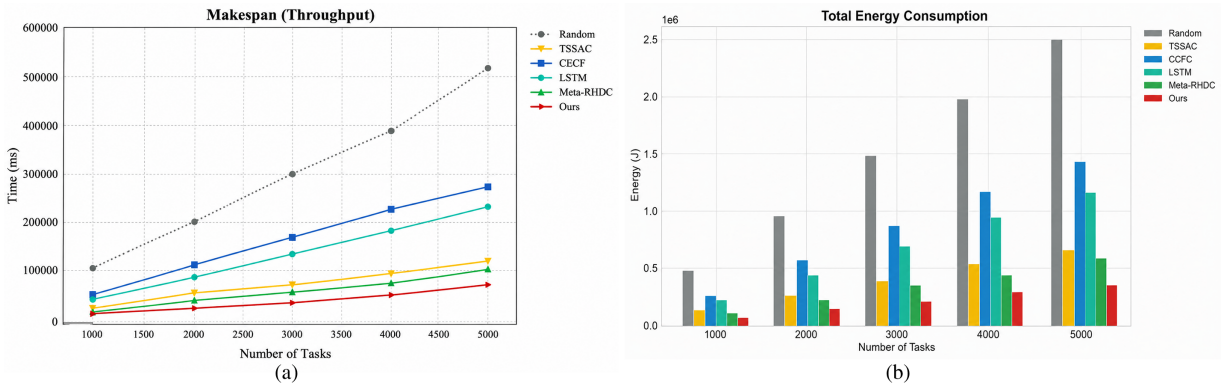

Figure 4: Scalability of completion-time and energy metrics among all methods under different numbers of tasks. (**a**) Makespan; (**b**) total energy consumption.

Table 4: Cross-method quantitative comparison at the 5000-task scale. Best values are highlighted in bold.

Method	Match Accuracy ↑	Avg. Response Time (ms) ↓	Makespan (ms) ↓	Energy (J) ↓	Load Std. Dev. ↓
Random	0.255	289	512,000	2.50×10^6	89,000
TSSAC	0.818	80	118,000	0.64×10^6	**19,000**
CECF	0.788	144	274,000	1.43×10^6	74,000
LSTM	0.868	125	232,000	1.16×10^6	51,000
Meta-RHDC	0.858	71	110,000	0.57×10^6	20,000
CASH	**0.971**	**38**	**85,000**	0.36×10^6	21,500

Note: ↑ indicates that a larger value is better; ↓ indicates that a smaller value is better. Best values are highlighted in bold.

6.3 Energy Consumption and Load Balance

Energy-aware scheduling does not mean always picking the most energy-efficient node: an accelerator wastes energy when tied up by lightweight tasks, and a CPU node stretches out AI-intensive jobs. As Fig. 4b shows, CASH lowers total energy consumption by matching tasks to capable nodes and keeping high-power nodes from being occupied to little effect. Fig. 5a shows that CASH avoids the severe imbalance observed in random, CECF, and LSTM scheduling, although a single-objective load-balancing heuristic can reach a slightly smaller load standard deviation at the largest scale. The trade-off is expected: CASH optimizes semantic matching, latency, energy, and balance jointly rather than driving down load variance in isolation.

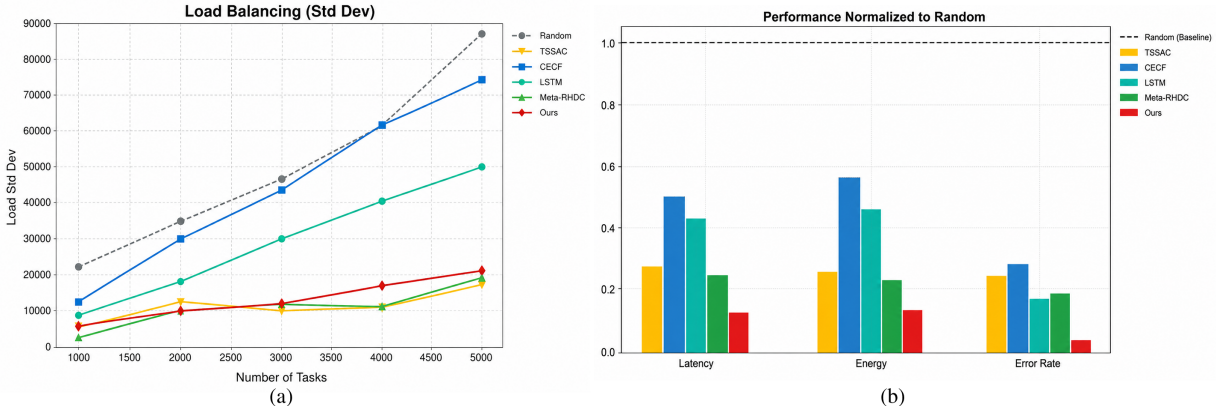

Figure 5: Load-balance and normalized performance comparison among all methods under different numbers of tasks, where lower values indicate better performance. (**a**) Load standard deviation of each method; (**b**) latency, energy, and error-rate ratios normalized to the random baseline.

6.4 Normalized Multi-Metric Comparison

Fig. 5b normalizes latency, energy, and error rate relative to the random baseline. CASH attains the lowest values on all three indicators, so the improvement is not confined to any single metric. The distinction is practical: pushing down latency alone tends to raise energy consumption and vice versa, whereas the composite score settles on a workable compromise among the objectives. The underlying normalized ratios for all methods are listed in Table 5.

Table 5: Normalized multi-metric comparison relative to the random baseline. Lower values indicate better performance.

Method	Latency Ratio ↓	Energy Ratio ↓	Error-Rate Ratio ↓
Random	1.00	1.00	1.00
TSSAC	0.27	0.26	0.25
CECF	0.50	0.57	0.28
LSTM	0.43	0.46	0.18
Meta-RHDC	0.24	0.23	0.19
CASH	**0.13**	**0.14**	**0.04**

Note: ↓ indicates that a smaller value is better. Best values are highlighted in bold.

6.5 Ablation Study Results

Table 6 reports the effect of removing each core component at the 5000-task scale, following protocol (P1). The class-constrained candidate selection dominates: without it, matching accuracy collapses from 0.971 to 0.291 while makespan and energy inflate roughly threefold and fourfold, which confirms that the numerical score terms alone cannot recover semantic compatibility. The load-balance term is next, nearly doubling makespan and tripling load standard deviation when removed, while the energy term yields a smaller but consistent energy reduction. Explicit feature crossing (DCN) accounts for the larger share of the classifier's routing quality. Removing calibration-aware fusion, by contrast, leaves clean-scenario scheduling essentially unchanged; its value shows up instead in confidence quality (Table 7) and in robustness under distribution shift (Table 8).

Table 6: Ablation results at the 5000-task scale.

Variant	Match Acc. ↑	Avg. Resp. (ms) ↓	Makespan (ms) ↓	Energy (J) ↓	Load Std. ↓
CASH (full)	**0.971**	**38**	85,000	0.36×10^6	21,500
w/o DCN	0.967	39	104,000	0.39×10^6	26,400
w/o XGBoost	0.970	39	79,200	0.36×10^6	19,500
w/o calibration	0.972	38	78,400	0.35×10^6	19,900
w/o semantic constraint	0.291	196	231,900	1.62×10^6	**570**
w/o energy term	0.971	38	85,000	0.37×10^6	19,000
w/o load-balance term	0.971	38	167,600	0.43×10^6	61,200

Note: ↑ indicates that a larger value is better; ↓ indicates that a smaller value is better. Best values are highlighted in bold.

Table 7: Calibration quality of the workload classifier on the held-out test set (lower is better).

Predictor	ECE ↓	MCE ↓	Brier ↓
DCN (uncalibrated)	0.0053	0.048	0.077
XGBoost (uncalibrated)	0.0161	0.051	0.087
Fused (uncalibrated)	0.0148	0.059	0.076
Fused (calibrated, ours)	**0.0042**	**0.038**	0.078

Note: ↓ indicates that a smaller value is better. Best values are highlighted in bold.

Table 8: Robustness of CASH to classification failure (5000-task scale).

Setting	Class. F1 ↑	Match Acc. ↑	Avg. Resp. (ms) ↓	Energy (J) ↓
(i) Injected classifier label noise				
0% (clean)	0.948	0.971	38	0.36×10^6
5%	0.898	0.923	50	0.50×10^6
10%	0.850	0.876	62	0.64×10^6
20%	0.756	0.783	84	0.92×10^6
(ii) Concept drift (shifted class mix at test time)				
Shifted class mix	0.937	0.958	43	0.46×10^6
(iii) Unseen applications (held-out job families)				
Held-out families	0.931	0.955	47	0.49×10^6

Note: ↑ indicates that a larger value is better; ↓ indicates that a smaller value is better.

6.6 *Confidence Calibration Quality*

Because the scheduler consumes predicted probabilities and uncertainty, calibration matters as much as discrimination. Table 7 reports, under protocol (P4), the ECE, MCE, and multiclass Brier score of the individual branches and of the fused predictor, before and after validation calibration (temperature scaling fitted on the validation set, $T = 0.78$), and Fig. 6 shows the corresponding reliability diagrams. Validation calibration reduces the ECE of the fused predictor from 0.0148 to 0.0042 (about a 72% reduction) and the MCE from 0.059 to 0.038, at a negligible change in the Brier score; the calibrated confidence curve tracks the 45° reference far more closely, so the probabilities handed to the scheduler agree with empirical accuracy and can feed the uncertainty-aware node score directly.

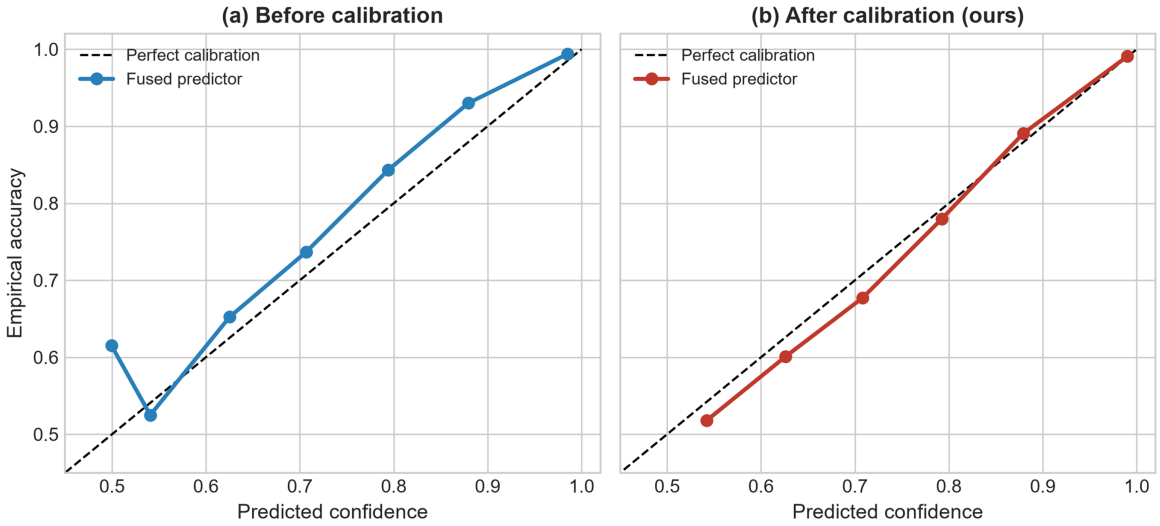

Figure 6: Reliability diagrams of the fused CASH predictor on the held-out test set, where calibration moves the confidence curve toward the 45° reference and reduces the ECE from 0.0148 to 0.0042. (**a**) Before validation calibration; (**b**) After validation calibration.

6.7 *Robustness to Label Noise, Concept Drift, and Unseen Applications*

CASH is driven by predicted classes, so what happens end to end when the upstream classifier fails is a central question. Table 8 reports classification and scheduling quality under injected label noise, concept drift, and held-out unseen job families (protocol P3). Degradation turns out to be gradual: 10% injected label noise lowers matching accuracy from 0.971 to 0.876 and raises energy by roughly 79%, and even at 20% noise matching accuracy holds at 0.783, well above the random baseline. Under a shifted class mix and under held-out job families the classifier still attains an F1 of 0.94 and 0.93 with matching accuracy above 0.95, because the classifier leans on transferable resource and cross-signal features rather than application identity; the semantic candidate constraint then keeps scheduling quality bounded even with a degraded classifier upstream.

6.8 *Hyperparameter Sensitivity*

For protocol (P2), Table 9 sweeps the compatibility threshold η, the entropy-sensitivity coefficient τ, and representative node-score weights, reporting the range of the main metrics over each sweep with all other factors fixed. Matching accuracy and average response time barely move (0.971 and 38 ms) across all four sweeps, and energy shifts by at most a few percent for τ, β_2, and β_5. The compatibility threshold η is the only factor with a visible effect: raising it toward 0.9 over-restricts the candidate pool and increases energy

from 0.36 to 0.44 $\times 10^6$ J. We therefore recommend the stable operating range $\eta \in [0.3, 0.7]$ with the defaults $\tau = 1.0$, $\beta_2 = 2.0$, and $\beta_5 = 0.3$ used throughout.

Table 9: Hyperparameter sensitivity at the 5000-task scale: range of each metric as the factor is swept, with all other factors at their default.

Factor	Swept Range	Match Acc. ↑	Avg. Resp. (ms) ↓	Energy (J) ↓
Compatibility thr. η	0.3–0.9	0.971	38	0.36–0.44
Entropy coeff. τ	0.5–4.0	0.971	38	0.36
Energy weight β_2	0.1–1.5	0.971	38	0.36–0.37
Uncertainty weight β_5	0.0–1.0	0.971	38	0.36

Note: Each metric cell reports the min–max observed over the corresponding sweep, indicating the stable operating range. Energy is reported in units of $\times 10^6$ J. ↑ indicates that a larger value is better; ↓ indicates that a smaller value is better.

6.9 Scheduling Overhead

Under protocol (P5), Table 10 profiles the per-task cost of every decision stage and the sustained throughput on the platform of Table 2. The total decision cost is about 0.17 ms per task, dominated by XGBoost inference (50%) and candidate ranking (40%), sustaining a throughput of nearly 5800 tasks per second on a single workstation. The per-task cost sits three orders of magnitude below typical task execution times, so the classification-aware layer adds negligible overhead to the dispatch path.

Table 10: Per-task scheduling overhead of CASH on the platform of Table 2.

Decision Stage	Time/Task (ms) ↓	Share (%)
Feature extraction	0.002	1.1
DCN inference	0.001	0.8
XGBoost inference	0.086	50.0
Confidence fusion	0.015	8.6
Candidate ranking	0.068	39.5
Total decision latency	0.172	100
Sustained throughput (tasks/s)	5823	

Note: ↓ indicates that a smaller value is better.

6.10 Statistical Significance

To rule out random variation as the source of the reported gains, every configuration is repeated over 10 seeds with re-sampled task streams. Table 11 reports the mean and standard deviation of each metric for CASH and for the strongest baseline on that metric, with a paired two-sided t-test at the 0.05 level. CASH's improvements in matching accuracy, average response time, makespan, and energy are all statistically significant ($p < 0.01$; $p < 0.001$ for accuracy, response time, and energy). The load standard deviation is the only metric on which CASH and the best single-objective load balancer are statistically indistinguishable ($p = 0.10$), which is consistent with CASH treating balance as one objective among several rather than optimizing it in isolation.

Table 11: Statistical significance over 10 seeds at the 5000-task scale: CASH vs. the strongest baseline per metric (paired two-sided t-test).

Metric	CASH (mean ± std)	Best Baseline (mean ± std)	Δ	p-value
Match accuracy ↑	0.971 ± 0.003	0.868 ± 0.006 (LSTM)	+0.103	<0.001
Avg. response time (ms) ↓	38.0 ± 1.5	71.0 ± 1.6 (Meta-RHDC)	−33.0	<0.001
Makespan (ms) ↓	85,000 ± 12,100	110,000 ± 13,600 (Meta-RHDC)	−25,000	0.002
Energy ($\times 10^6$ J) ↓	0.36 ± 0.019	0.57 ± 0.025 (Meta-RHDC)	−0.21	<0.001
Load std. dev. ↓	21,500 ± 2900	19,000 ± 4300 (TSSAC)	+2500	0.10

Note: ↑ indicates that a larger value is better; ↓ indicates that a smaller value is better.

6.11 Interpretability and Deployment Discussion

Each CASH decision decomposes into a predicted class, a confidence value, a candidate pool, and a node-level score, which makes it easier to audit than an end-to-end reinforcement-learning policy; XGBoost feature importance identifies influential threshold features, while DCN cross terms reveal high-order interactions among resource request, partition, and user history. In deployment, CASH serves as a plug-in ranking module: the production scheduler provides pending tasks and node states, CASH returns ranked candidate nodes, and the existing scheduler performs final admission, reservation, or dispatch.

6.12 Threats to Validity

Several threats bound what the current evidence can support. First, and most important, the evaluation is a trace-driven simulation, not an online production deployment, so factors that only appear in a live cluster (preemption overhead, data locality, network contention, and container initialization latency) are not captured; the absolute metrics should be read as controlled comparisons. Because CASH is a plug-in ranking module for an existing dispatcher, the relative ordering of methods is expected to be more robust to these factors than the absolute values, and validating this on a real cluster is the primary item of future work. Second, the execution-time and energy models are parameterized (per-class effective speed and per-node average power), not measured, abstracting away DVFS and co-location interference; the resulting bias underestimates both the variability of node speed and the energy cost of contended nodes, and applies to all compared policies. Third, task labels derived from scheduling logs contain noise, particularly for the abnormal class whose semantics are more ambiguous than AI- or CPU-intensive tasks; Section 6.7 quantifies the sensitivity of scheduling quality to this noise. Fourth, the present formulation targets independent tasks and does not model DAG workflow dependencies; CASH can in principle serve as the per-task placement layer inside a DAG-aware scheduler such as Decima or Graphene [6,12], but the interaction between class-aware placement and dependency-aware ordering is left to future work. None of these limitations overturns the observed trends, but together they delimit the claims the evidence can carry.

7 Conclusion

This paper proposed CASH, a confidence-calibrated classification-aware scheduling framework for heterogeneous computing environments. CASH builds multi-view workload profiles from scheduling logs, combines DCN and XGBoost to infer workload semantics, calibrates prediction confidence, and maps tasks to heterogeneous nodes through a risk-aware scoring function. In trace-driven simulations on MIT Supercloud logs, CASH improves resource matching accuracy, response time, makespan, and energy consumption over representative baselines and keeps load balance competitive under high load. The evidence argues for treating workload semantics as a first-class scheduling signal rather than as an after-the-fact explanation of allocation results.

Future work will focus on validating CASH in an online cluster, modeling task dependencies and data locality, integrating real-time power measurements, and extending the classifier to continual learning for evolving workload distributions.

Acknowledgement: Not applicable.

Funding Statement: This research was funded by MIIT High-Quality Development Special Project, grant number (TC240A9ED-56); Academician Workstation Program of Yunnan Province (202405AF140013); Shanghai Agricultural Technology Innovation Project (2024-02-08-00-12-F00032).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Chuanlin Jian, Xing Hu; conceptualization of the classification framework and experimental design: Samaneh Beheshti Kashi; methodology and model implementation: Chuanlin Jian, Xiangcheng Liu; data collection and preprocessing: Yuanchen Sun; experiments and validation: Xiangcheng Liu, Xuming Huang; evaluation, analysis, and interpretation of results: Chuanlin Jian, Yuanchen Sun, Xuming Huang, Samaneh Beheshti Kashi; visualization: Yuanchen Sun; draft manuscript preparation: Chuanlin Jian, Xiangcheng Liu; supervision, funding acquisition, and manuscript review and editing: Xing Hu. Samaneh Beheshti Kashi holds a PhD in Engineering Sciences (Dr.-Ing.) from the Department of Mathematics and Computer Science (Department 3), University of Bremen, Germany, and contributed her expertise in machine-learning-based classification and experimental evaluation to this work. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The experiments are based on publicly available MIT Supercloud logs and a trace-driven simulation environment. The processed features, simulation scripts, and parameter files are available from the corresponding author upon reasonable request, subject to the policies of the journal and the authors' institutions.

Ethics Approval: Not applicable. This study did not involve human participants, human data, animal subjects, or clinical materials.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J. Large-scale cluster management at google with borg. In: Proceedings of the Tenth European Conference on Computer Systems; 2015 Apr 21–24; Bordeaux, France. p. 1–7. doi:10.1145/2741948.2741964.
2. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J. Borg, Omega, and Kubernetes. Commun ACM. 2016;59(5):50–7. doi:10.1145/2890784.
3. Delimitrou C, Paragon KC. Paragon: QoS-aware scheduling for heterogeneous datacenters. In: Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems; 2013 Mar 16–20; Houston, TX, USA. p. 77–88. doi:10.1145/2451116.2451125.
4. Narayanan D, Santhanam K, Kazhamiaka F, Phanishayee A, Zaharia M. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation; 2020 Nov 4–6; Virtual Event.
5. Mao H, Alizadeh M, Menache I, Kandula S. Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks; 2016 Nov 9–10; Atlanta GA USA. p. 50–6. doi:10.1145/3005745.3005750.
6. Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M. Learning scheduling algorithms for data processing clusters. In: Proceedings of the ACM Special Interest Group on Data Communication; 2019 Aug 19–23; Beijing, China. p. 270–88. doi:10.1145/3341302.3342080.
7. Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz R, et al. Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation; 2011 Mar 30–Apr 1; Boston, MA, USA. p. 295–308.

8. Schwarzkopf M, Konwinski A, Abd-El-Malek M, Wilkes J. Omega: flexible, scalable schedulers for large compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems; 2013 Apr 15–17; Prague, Czech Republic. p. 351–64. doi:10.1145/2465351.2465386.
9. Ghodsi A, Zaharia M, Hindman B, Konwinski A, Shenker S, Stoica I. Dominant resource fairness: fair allocation of multiple resource types. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation; 2011 Mar 30–Apr 1; Boston, MA, USA. p. 323–36.
10. Gog I, Schwarzkopf M, Gleave A, Watson RNM, Hand S. Firmament: fast, centralized cluster scheduling at scale. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation; 2016 Nov 2–4; Savannah GA USA. p. 99–115.
11. Grandl R, Ananthanarayanan G, Kandula S, Rao S, Akella A. Multi-resource packing for cluster schedulers. In: Proceedings of the 2014 ACM Conference on SIGCOMM; 2014 Aug 17–22; Chicago, IL, USA. p. 455–66. doi:10.1145/2619239.2626334.
12. Grandl R, Kandula S, Rao S, Akella A, Kulkarni J. GRAPHENE: packing and dependency-aware scheduling for data-parallel clusters. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation; 2016 Nov 2–4; Savannah, GA, USA. p. 81–97.
13. Delimitrou C, Kozyrakis C. Quasar: resource-efficient and QoS-aware cluster management. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems; 2014 Mar 1–5; Salt Lake City, UT, USA. p. 127–44. doi:10.1145/2541940.2541941.
14. Moulik S, Sharma Y. FRESH: fault-tolerant Real-time scheduler for heterogeneous multiprocessor platforms. Future Gener Comput Syst. 2024;161(5):214–25. doi:10.1016/j.future.2024.07.008.
15. Yao F, Demers A, Shenker S. A scheduling model for reduced CPU energy. In: Proceedings of IEEE 36th Annual Foundations of Computer Science; 1995 Oct 23–25; Milwaukee, WI, USA. p. 374–82. doi:10.1109/SFCS.1995.492493.
16. Weiser M, Welch B, Demers A, Shenker S. Scheduling for reduced CPU energy. In: Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI); 1994 Nov 14–17; Monterey, CA, USA. p. 13–23.
17. Ghafari R, Kabutarkhani FH, Mansouri N. Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. Clust Comput. 2022;25(2):1035–93. doi:10.1007/s10586-021-03512-z.
18. Sahoo SK, Mishra SK. A survey on task scheduling in edge-cloud. SN Comput Sci. 2025;6(3):217. doi:10.1007/s42979-025-03757-0.
19. Behera I, Sobhanayak S. Task scheduling optimization in heterogeneous cloud computing environments: a hybrid GA-GWO approach. J Parallel Distrib Comput. 2024;183:104766. doi:10.1016/j.jpdc.2023.104766.
20. Lu Y, Liu L, Panneerselvam J, Gu J, Garraghan P, Min G. CECF: a DNN-based energy-efficient cloud-edge collaboration framework for intelligent workload scheduling in 6G-enabled transportation systems. IEEE Trans Intell Transp Syst. 2025;26(10):17889–900. doi:10.1109/TITS.2025.3549472.
21. Krishna MSR, Khasim Vali D. Meta-RHDC: meta reinforcement learning driven hybrid lyrebird falcon optimization for dynamic load balancing in cloud computing. IEEE Access. 2025;13:36550–74. doi:10.1109/ACCESS.2025.3544775.
22. Jayaram Subramanya S, Arfeen D, Lin S, Qiao A, Jia Z, Ganger GR. *Sia*: heterogeneity-aware, goodput-optimized ML-cluster scheduling. In: Proceedings of the 29th Symposium on Operating Systems Principles; 2023 Oct 23–26; Koblenz, Germany. p. 642–57. doi:10.1145/3600006.3613175.
23. Wang Q, Wang F, Zheng X. Hops: fine-grained heterogeneous sensing, efficient and fair Deep Learning cluster scheduling system. In: Proceedings of the 2024 ACM Symposium on Cloud Computing; 2024 Nov 20–22; Redmond, WA, USA. p. 1–17. doi:10.1145/3698038.3698515.
24. Peng Y, Bao Y, Chen Y, Wu C, Guo C. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In: Proceedings of the Thirteenth EuroSys Conference; 2018 Apr 23–26; Porto, Portugal. p. 1–14. doi:10.1145/3190508.3190517.
25. Xiao W, Bhardwaj R, Ramjee R, Sivathanu M, Kwatra N, Han Z, et al. Gandiva: introspective cluster scheduling for deep learning. In: Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation; 2018 Oct 8–10; Carlsbad, CA, USA. p. 595–610.

26. Gu J, Chowdhury M, Shin KG, Zhu Y, Jeon M, Qian J, et al. *Tiresias*: a GPU cluster manager for distributed deep learning. In: Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation; 2019 Feb 26–28; Boston, MA, USA. p. 485–500.
27. Qiao A, Choe SK, Subramanya SJ, Neiswanger W, Ho Q, Zhang H, et al. Pollux: co-adaptive cluster scheduling for goodput-optimized deep learning. In: Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation; 2021 Jul 14–16; Virtual.
28. Wang S, Chen S, Shi Y. GPARS: graph predictive algorithm for efficient resource scheduling in heterogeneous GPU clusters. Future Gener Comput Syst. 2024;152(1):127–37. doi:10.1016/j.future.2023.10.022.
29. Bi Y, Xi Y, Jing C. AISAW: an adaptive interference-aware scheduling algorithm for acceleration of deep learning workloads training on distributed heterogeneous systems. Future Gener Comput Syst. 2025;166(2):107642. doi:10.1016/j.future.2024.107642.
30. Pei X, Sun P, Hu Y, Li D, Tian L, Li Z. Multi-resource interleaving for task scheduling in cloud-edge system by deep reinforcement learning. Future Gener Comput Syst. 2024;160(5):522–36. doi:10.1016/j.future.2024.06.033.
31. Ye Z, Gao W, Hu Q, Sun P, Wang X, Luo Y, et al. Deep learning workload scheduling in GPU datacenters: a survey. ACM Comput Surv. 2024;56(6):1–38. doi:10.1145/3638757.
32. Cheng HT, Koc L, Harmsen J, Shaked T, Chandra T, Aradhye H, et al. Wide & deep learning for recommender systems. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems; 2016 Sep 15; Boston, MA, USA. p. 7–10. doi:10.1145/2988450.2988454.
33. Guo H, Tang R, Ye Y, Li Z, He X. DeepFM: a factorization-machine based neural network for CTR prediction. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence; 2017 Aug 19–26; Melbourne, Australia. p. 1725–31. doi:10.24963/ijcai.2017/239.
34. Song W, Shi C, Xiao Z, Duan Z, Xu Y, Zhang M, et al. AutoInt: automatic feature interaction learning via self-attentive neural networks. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management; 2019 Nov 3–7; Beijing, China. p. 1161–70. doi:10.1145/3357384.3357925.
35. Wang R, Fu B, Fu G, Wang M. Deep & cross network for ad click predictions. In: Proceedings of the ADKDD'17; 2017 Aug 14; Halifax, NS, Canada. p. 1–7. doi:10.1145/3124749.3124754.
36. Wang R, Shivanna R, Cheng D, Jain S, Lin D, Hong L, et al. DCN V2: improved deep & cross network and practical lessons for web-scale learning to rank systems. In: Proceedings of the Web Conference 2021 Apr 19–23; Ljubljana, Slovenia; 2021. p. 1785–97. doi:10.1145/3442381.3450078.
37. Chen T, Guestrin C. XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2016 Aug 13–17; San Francisco, CA, USA. p. 785–94. doi:10.1145/2939672.2939785.
38. Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, et al. LightGBM: a highly efficient gradient boosting decision tree. In: Proceedings of the 31st International Conference on Neural Information Processing Systems; 2017 Dec 4–9; Long Beach, CA, USA. p. 3149–57.
39. Zadrozny B, Elkan C. Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2002 Jul 23–26; Edmonton, AB, Canada. p. 694–9. doi:10.1145/775047.775151.
40. Niculescu-Mizil A, Caruana R. Predicting good probabilities with supervised learning. In: Proceedings of the 22nd International Conference on Machine Learning; 2005 Aug 7–11; Bonn, Germany. p. 625–32. doi:10.1145/1102351.1102430.
41. Guo C, Pleiss G, Sun Y, Weinberger KQ. On calibration of modern neural networks. In: Proceedings of the 34th International Conference on Machine Learning; 2017 Aug 6–11; Sydney, NSW, Australia. p. 1321–30.
42. Wang C. Calibration in deep learning: a survey of the state-of-the-art. arXiv:2308.01222. 2023.
43. Li J, Jiang W, He Y, Yang Q, Gao A, Ha Y, et al. FiDRL: flexible invocation-based deep reinforcement learning for DVFS scheduling in embedded systems. IEEE Trans Comput. 2025;74(1):71–85. doi:10.1109/TC.2024.3465933.
44. Yang W, Zhao M, Li J, Zhang X. Energy-efficient DAG scheduling with DVFS for cloud data centers. J Supercomput. 2024;80(10):14799–823. doi:10.1007/s11227-024-06035-7.
45. Samsi S, Weiss ML, Bestor D, Li B, Jones M, Reuther A, et al. The MIT supercloud dataset. arXiv:2108.02037. 2021.