EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

AML-TSCRA: A Machine Learning-Based Task Scheduling and Computational Resource Adaptive Allocation Method for Library Distributed Databases

Can Ma*

Zhejiang Industry Polytechnic College, ShaoXing, Zhejiang, 312000, China

Abstract

INTRODUCTION: As libraries increasingly rely on distributed database systems to manage growing data volumes and service demands, conventional task scheduling and resource allocation strategies often struggle to adapt to dynamic workloads, leading to suboptimal resource utilization and performance bottlenecks.
OBJECTIVES: This paper aims to develop an intelligent adaptive framework that jointly optimizes task scheduling and computational resource allocation, thereby improving responsiveness, load balancing, and overall system efficiency.
METHODS: We propose AML-TSCRA (Adaptive Machine Learning-based Task Scheduling and Computational Resource Allocation), which integrates LSTM-based load forecasting with reinforcement learning-based scheduling and adaptive resource allocation based on real-time system states.
RESULTS: Experiments on three representative datasets, including Amazon EC2 Logs, Google Cluster Data, and SLURM Workload Manager, show that AML-TSCRA achieves its strongest performance on Amazon EC2 Logs, with a task completion rate of 88.9%, resource utilization of 81.8%, an average response time of 1.31 s, and system throughput of 23.7 tasks/s. The model also shows moderate gains on Google Cluster Data and smaller but consistent advantages on the more heterogeneous SLURM workload. In addition, it demonstrates relatively better robustness than the learning-based baselines under noisy conditions.
CONCLUSION: AML-TSCRA improves the efficiency and adaptability of distributed library-oriented service systems by combining predictive foresight, adaptive scheduling, and coordinated resource allocation. The proposed framework provides a practical basis for intelligent resource orchestration in modern data-intensive environments.

Keywords: Adaptive Scheduling, Computational Resource Allocation, Machine Learning, Distributed Database, Reinforcement Learning

Received on 02 February 2026, accepted on 15 May 2026, published on 09 June 2026

doi: 10.4108//eetsis.11808

1. Introduction

With the development of the information age, data storage and management in large-scale systems such as libraries face significant challenges. Modern digital libraries are increasingly adopting distributed architectures to support intensive Research Data Management (RDM) and massive institutional repositories[1][2]. However, the bursty and tidal nature of library-specific workloads, ranging from high-frequency queries to large-scale background metadata indexing, has revealed the inefficiency and uneven resource utilization of traditional database task scheduling methods[3]. Particularly in distributed database environments, efficiently scheduling tasks and allocating computational resources has become a key issue for improving system performance. Although distributed computing and machine learning technologies provide new opportunities, existing research

*Corresponding author. Email: mc_2024@sina.com

often focuses on one aspect and lacks comprehensive solutions that integrate machine learning, especially in the application to specific, domain-aware scenarios like library information infrastructures[4][5].

Although research has addressed task scheduling and computational resource allocation, most methods rely on static models or empirical strategies, lacking adaptability. Rule-based scheduling methods cannot flexibly handle task priority changes, leading to low resource utilization[6][7]. While machine learning methods, such as reinforcement learning, have been applied, they have not effectively solved the issue of adaptive computational resource allocation, resulting in failure to improve system performance under high load conditions[8][9]. Therefore, how to integrate machine learning technologies to construct an efficient framework for task scheduling and adaptive computational resource allocation remains a central issue in current research.

To solve these problems, this paper proposes the AML-TSCRA model (Adaptive Machine Learning-based Task Scheduling and Computational Resource Allocation), a machine learning-based adaptive task scheduling and computational resource allocation method for library distributed databases. This model uses deep learning for load prediction and combines reinforcement learning to optimize task scheduling strategies, achieving proactive allocation of computational resources. AML-TSCRA automatically adjusts scheduling and allocation strategies based on task requirements and system load changes, thereby improving task execution efficiency and resource utilization under highly dynamic library access patterns.

Innovations in this paper include: (1) Proactive Scheduling Model: A machine learning-based adaptive task scheduling method that automatically adjusts strategies based on LSTM-driven foresight rather than merely reacting to real-time load changes. (2) Adaptive Computational Resource Allocation: A dynamic resource allocation model combining deep learning and reinforcement learning, enabling flexible resource allocation. (3) Closed-loop Integrated Optimization Framework: A comprehensive optimization framework that continuously feeds scheduling metrics back into the priority formulation, combining task characteristics and load variations to achieve global optimization specifically tailored for library workloads.

The contribution of this paper lies not only in proposing a new task scheduling and computational resource allocation model but also in providing an efficient resource management solution for large-scale data systems like libraries, and advancing the application of machine learning and distributed computing in data management. Rather than generic cloud environments, this method establishes a domain-aware foundation specifically designed for next-generation digital library infrastructures and academic computing centers.

2. Related Works

2.1. Application Scenarios and Challenges

In domain-specific data management, addressing the unique infrastructure needs of modern digital libraries is a critical challenge[10]. Today's libraries are complex ecosystems supporting intensive Research Data Management (RDM) and massive institutional repositories[11]. These library-specific workloads exhibit strong tidal characteristics, ranging from high-concurrency OPAC queries to large-scale background digital preservation[12]. Managing such dynamic, heterogeneous tasks requires systems with domain-aware, efficient resource scheduling and task allocation capabilities.

In this context, how to optimize resource utilization and improve system response speed has become a core issue. Especially in distributed database environments, resource scheduling and task allocation are not only related to efficiency but also involve real-time adjustment of task priorities and adaptive allocation of computational resources[13][14]. As mainstream datasets grow in scale and data types become more complex, scheduling systems require powerful computing capabilities and must adapt to changes in data volume and task demands[15].

Many studies have evaluated using standard datasets, which are characterized by high complexity and scale, covering a wide range of real-world scenarios[16][17]. However, despite these datasets driving technological progress, challenges remain in practical applications. Standard datasets typically focus on single metrics such as system efficiency and response time, lacking a comprehensive evaluation of factors such as load fluctuations and resource utilization balance. Furthermore, existing evaluation systems tend to focus on static testing, lacking consideration of dynamic load and task demand changes, which limits their effectiveness.

2.2. Overview of Mainstream Methods

Existing task scheduling and resource allocation methods can be broadly categorized into rule-based scheduling methods, optimization algorithm methods, and machine learning-based methods. Rule-based scheduling methods rely on static rules for task scheduling, such as Shortest Job First (SJF) or Round-Robin scheduling[18]. These methods perform well in small-scale or low-load environments, but as system scale expands and task demands diversify, they fail to address load fluctuations and resource contention, leading to performance degradation under high loads[19]. Optimization algorithm methods, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), solve resource scheduling problems by optimizing objective functions[20][21]. These methods improve resource allocation efficiency, but their computational complexity is high, making them difficult to apply in real-time environments, especially when task numbers are large and system load fluctuates frequently. In such cases, algorithm convergence speed and computational efficiency become bottlenecks.

In recent years, machine learning-based methods have gradually become a research hotspot for task scheduling and resource allocation[22]. These methods train models to predict future demands and optimize task scheduling and

resource allocation strategies. For example, reinforcement learning methods can adaptively adjust scheduling strategies based on system feedback, overcoming the static limitations of traditional methods. However, these generic ML methods typically treat workload prediction and task scheduling as isolated, open-loop pipelines. They rely on reactive state representations and struggle to proactively adapt to the highly tidal and bursty access patterns characteristic of digital libraries[23].

Unlike existing methods, the AML-TSCRA model combines deep learning for load prediction with reinforcement learning for task scheduling optimization, addressing the limitations of existing methods in dynamic load environments. AML-TSCRA further enhances resource utilization and system response speed by dynamically adjusting computational resource allocation.

2.3. Closest Related Research

Related research includes task scheduling methods based on optimization algorithms and machine learning. Some studies focus on optimizing task scheduling algorithms, using genetic algorithms and other methods for task allocation, achieving good performance[24][25]. However, these methods have high computational complexity and are difficult to apply in real-time environments with high performance requirements. Additionally, they fail to effectively address the issue of adaptive computational resource allocation.

Another category of research uses reinforcement learning and deep learning to optimize task scheduling, but these studies predominantly operate in a purely reactive manner and lack a closed-loop feedback mechanism to dynamically adjust task priorities based on real-time load imbalance[26]. In contrast, the AML-TSCRA model proposed in this paper combines deep learning with reinforcement learning to solve the dual problem of load prediction and computational resource allocation. It is capable of adaptively adjusting task scheduling and resource allocation in high-load and dynamic environments, offering significant advantages.

2.4. Summary and Research Gaps

Although existing research has made progress in task scheduling, current generic methods remain insufficient when faced with the highly dynamic, bursty access patterns typical of modern digital libraries and Research Data Management (RDM) systems. Existing research often treats prediction and allocation in isolation, relying on reactive scheduling policies evaluated primarily in static or generic cloud environments. Consequently, there is a noticeable lack of domain-aware, systematic research tailored to the unique tidal fluctuations of library infrastructures.

The AML-TSCRA model proposed in this paper explicitly fills this gap. Unlike prior works, this study introduces a genuine closed-loop optimization framework that seamlessly embeds LSTM-based proactive foresight into the RL agent's state space. By dynamically adjusting task priorities and resource allocations through feedback mechanisms, AML-TSCRA provides a highly adaptable, domain-specific solution for distributed library databases, overcoming the reactive limitations of traditional end-to-end RL approaches.

3. Methodology

3.1. Problem Formulation

In a distributed database environment, task loads and resource demands exhibit dynamic and unpredictable characteristics. To address this challenge, this study aims to construct an optimization model that adaptively schedules tasks and allocates computational resources to maximize overall system performance and resource utilization. First, the core elements of the system are formally defined as follows: the task set is denoted as $T = \{t_1, t_2, \ldots, t_n\}$, where each task$t_i$ has a computational demand w_i, dynamic priority $p_i(t)$, and state $s_i(t)$; the computational resource set is denoted as $R = \{r_1, r_2, \ldots, r_m\}$, where each resource r_j has a computational capacity c_j and real-time load $l_j(t)$.

The scheduling strategy π aims to allocate tasks t_i to appropriate resources $r_{\pi(i)}$ for execution. The following objective function is defined to systematically balance task priorities and resource load balancing:

$$\max_{\pi} \sum_{i=1}^{n} \sum_{t=0}^{T} \frac{w_i \cdot p_i(t)}{1+\alpha \cdot l_{\pi(i)}(t)} \quad (1)$$

where $l_{\pi(i)}(t)$ represents the load of the resource allocated to task t_i at time t, and α is a weight coefficient that adjusts the impact of resource load. This function encourages the system to schedule tasks with high computational demands and high priorities to resources with lower current loads, thus achieving load balancing while meeting task urgency.

To ensure the feasibility of the scheduling scheme, the following constraints must be satisfied: the computational demand of each task must not exceed the available capacity of the resource it is allocated to; each task can only be allocated to one computational resource at any given time; resource loads change dynamically over time, and the scheduling strategy must be adaptive to this dynamic nature. In summary, the core of this problem is to find the optimal scheduling strategy π^* that maximizes the objective function, ensuring global optimization of system throughput and resource utilization efficiency in a dynamic environment, while satisfying the above constraints.

3.2. Overall Framework

The AML-TSCRA model proposed in this paper aims to optimize task scheduling and computational resource allocation in distributed databases. The overall framework of the model consists of three core modules: the load prediction module, task scheduling module, and resource allocation module. These modules work collaboratively to achieve adaptive resource management under dynamic loads.

First, the load prediction module uses deep learning algorithms to predict future loads based on historical data and the system's real-time state. This module analyzes task demands and resource loads to generate load variation trend predictions, which serve as the basis for subsequent scheduling and allocation decisions, helping the system make optimization decisions in advance. Next, the task scheduling module, based on load predictions and task priorities, determines the task scheduling order. Through reinforcement learning algorithms, the scheduling module continuously adjusts the strategy based on system feedback to ensure rational task allocation while considering task dependencies and avoiding conflicts that may lead to performance bottlenecks. Finally, the resource allocation module dynamically adjusts computational resource allocation based on the scheduling results and load conditions. The core of this module is the adaptive computational resource allocation algorithm, which adjusts the computational resources for task allocation in real-time, optimizing resource utilization and avoiding waste or shortage.

Overall, the AML-TSCRA model achieves adaptive task scheduling and computational resource allocation in dynamic environments through the collaboration of these three modules. The load prediction module provides data support for scheduling and allocation, the task scheduling module ensures tasks are allocated according to priority, and the resource allocation module ensures system load balancing and maximizes resource utilization. The overall architecture is shown in Figure 1, which illustrates the data flow and functional relationships between the modules, clearly demonstrating the working mechanism of the model.

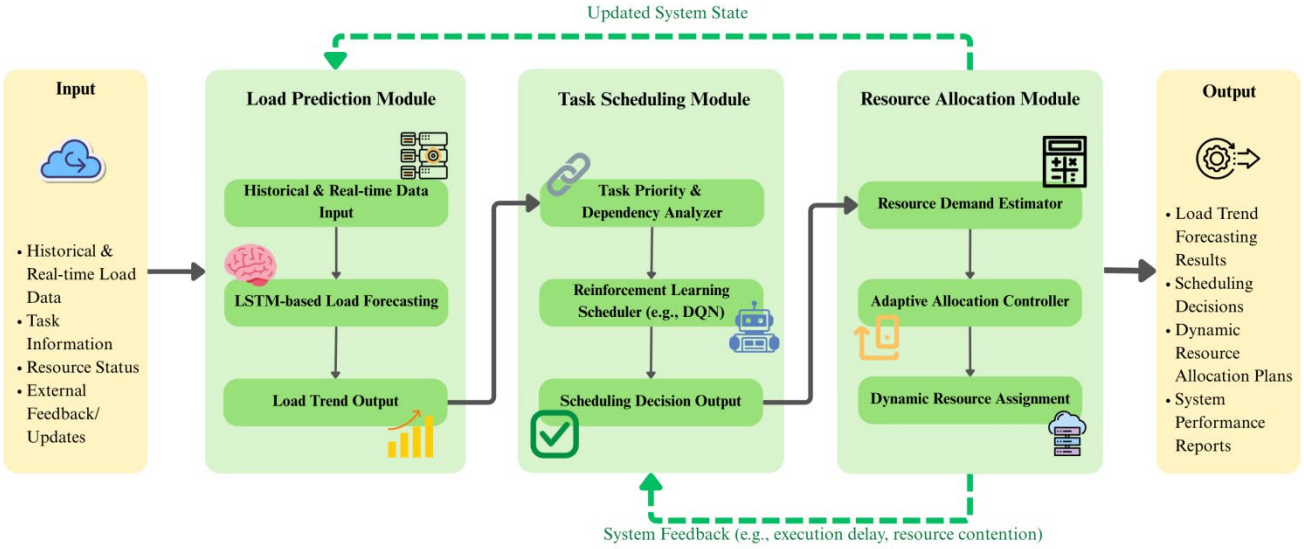

Figure 1. Overall architecture of the AML-TSCRA model

3.3. Detailed Module Descriptions

Load Prediction Module

Motivation:

In distributed databases, task loads and resource demands change rapidly and are difficult to predict. The lack of load prediction may prevent the system from adjusting resource allocation in advance, leading to performance bottlenecks or resource waste. Therefore, the introduction of the load prediction module helps the system optimize resource scheduling and allocation.

Principle:

The load prediction module employs Long Short-Term Memory (LSTM) networks to process historical usage data (e.g., CPU, memory) and resolve long-term dependencies via gating mechanisms. Rather than generic load forecasting, this module explicitly captures library-specific tidal patterns by encoding the stark contrast between high-frequency daytime queries (e.g., OPAC searches) and intensive nighttime background tasks (e.g., digital preservation and full-text indexing). This domain-aware feature design ensures the model accurately predicts both cyclical academic access and sudden bursty workloads.

Implementation:

The module is implemented through data collection, preprocessing, LSTM training, and prediction steps. Historical load data is standardized and input into the LSTM network for training. After training, real-time data is fed into the prediction model, which outputs predicted results for reference by the task scheduling and resource allocation modules. Figure 2 shows the structure of the load prediction module.

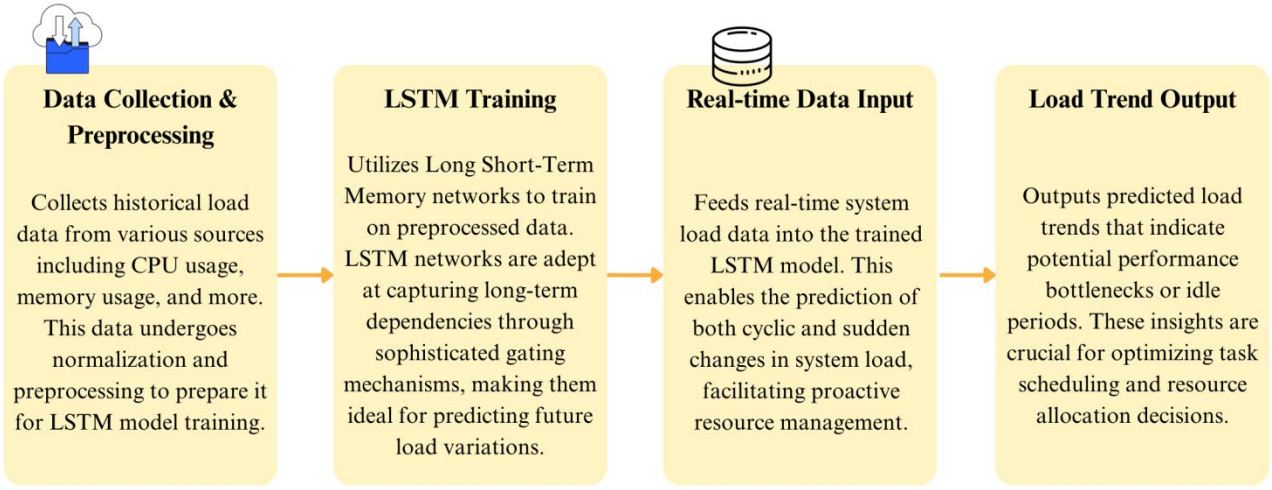

Figure 2. Schematic Diagram of the Load Prediction Module

Task Scheduling Module

Motivation:

The goal of task scheduling is to arrange the execution order of tasks based on their priorities and resource requirements. Traditional scheduling methods struggle to cope with dynamically changing task priorities and system loads. The task scheduling module optimizes task allocation through adaptive scheduling, reducing waiting and idle times while improving response speed and throughput.

Principle:

The task scheduling module is based on Reinforcement Learning (RL) algorithms, which learn the optimal strategy through interaction with system feedback. Task priority, resource demand, and system state are used as inputs, and reinforcement learning computes the reward value of scheduling actions to choose the optimal scheduling strategy.

Implementation:

Using Q-learning or Deep Q-Networks (DQN), the task scheduling module dynamically adjusts the scheduling strategy based on task status and system feedback. Reinforcement learning calculates rewards to select the best scheduling plan and optimize task allocation. Figure 3 shows the structure of the task scheduling module.

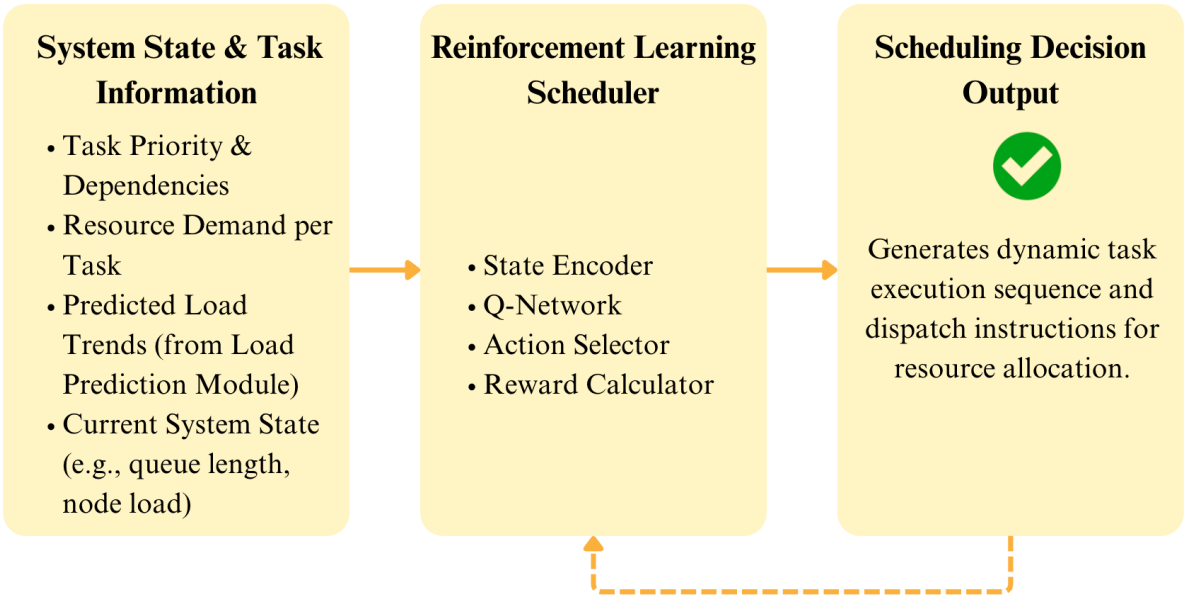

Figure 3. Schematic Diagram of the Task Scheduling Module

Resource Allocation Module

Motivation:

The resource allocation module dynamically allocates computational resources based on task demands and system load. Traditional static allocation methods cannot cope with rapid load changes. The resource allocation module uses adaptive algorithms to adjust resource allocation in real time, ensuring efficient use of computational resources.

Principle:

The resource allocation module is based on adaptive allocation algorithms, integrating deep learning and reinforcement learning. It dynamically adjusts resource allocation strategies based on task demands and load conditions, optimizing computational resource distribution to ensure high resource utilization.

Implementation:

This module receives load prediction results from the load prediction module and calculates the required resources for each task based on the task scheduling module's output. It dynamically adjusts resource allocation to ensure proper distribution of resources. Figure 4 shows the structure of the resource allocation module.

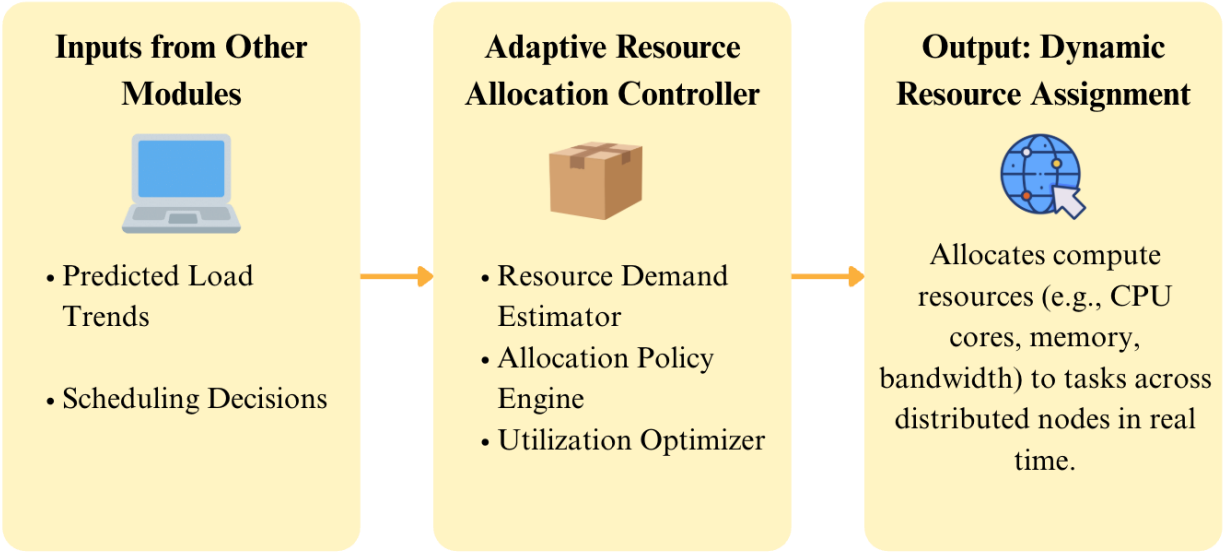

Figure 4. Schematic Diagram of the Resource Allocation Module

3.4. Objective Function & Optimization

This section builds a more refined, computable mathematical model based on the optimization problem defined in Section 3.1, to support the implementation of the AML-TSCRA framework. The core of the model is to decompose the intuitive performance objectives into quantifiable sub-goals of task efficiency and load balancing.

The overall system performance J aims to maximize task execution efficiency while minimizing resource load imbalance, defined as a weighted combination of the two:

$$J = \lambda_1 \cdot J_{task} - \lambda_2 \cdot J_{load} \tag{2}$$

where:

J_{task}: Task execution efficiency term.

J_{load}: Resource load imbalance penalty term.

λ_1,λ_2 (objective weight coefficients, with $\lambda_1, \lambda_2 > 0$) are used to adjust the balance between the two terms.

Furthermore, the reward shaping (reflected via the objective weight coefficients λ_1 and λ_2) is intrinsically tailored to library priorities. The formulation heavily penalizes latency for short, user-facing interactive queries (e.g., real-time catalog retrievals) to guarantee user experience, while tolerating higher scheduling flexibility for massive, latency-insensitive research data management (RDM) batch jobs, thereby distinguishing this reward mechanism from generic cloud schedulers.

To quantify scheduling decisions, we introduce binary decision variables $x_{ij}(t)$:

$$x_{ij}(t) = \begin{cases} 1, & if\ task\ t_i\ is\ allocated\ to\ resource\ r_j\ at\ time\ t, \\ 0, & otherwise. \end{cases} \tag{3}$$

To avoid action space explosion, the binary variables $x_{ij}(t)$ are not directly output by the Q-network. Instead, the RL agent outputs continuous allocation preference scores. A deterministic post-processing step (a capacity-bounded

greedy selection) is then applied to discretize these scores into the exact binary variables $x_{ij}(t)$, ensuring all hardware constraints are strictly met.

Task Execution Efficiency J_{task} encourages high-priority tasks to receive more computational resources and complete quickly, defined as the sum of the priority-weighted computational resources allocated to all tasks during the scheduling period:

$$J_{task} = \sum_{t=0}^{T} \sum_{i=1}^{n} p_i(t) \cdot \left(\sum_{j=1}^{m} x_{ij}(t) \cdot \eta_{ij}(t) \right) \quad (4)$$

where:

$p_i(t)$: Dynamic priority of task t_i at time t.

$\eta_{ij}(t)$: Effective computational resource allocation value for task t_i on resource r_j at time t.

Resource Load Imbalance J_{load} measures the discrepancy in load levels across all resources and aims to promote load balancing. First, we define the real-time load $L_j(t)$ of resource r_j at time t:

$$L_j(t) = \sum_{i=1}^{n} x_{ij}(t) \cdot w_i \quad (5)$$

where w_i is the baseline computational demand of task t_i, used here to measure its contribution to the resource load. Then, J_{load} is defined as the sum of the variance in resource loads over the entire period:

$$J_{load} = \sum_{t=0}^{T} \sum_{j=1}^{m} \left(L_j(t) - \bar{L}(t) \right)^2 \quad (6)$$

where $\bar{L}(t) = \frac{1}{m} \sum_{j=1}^{m} L_j(t)$ is the average load at time t.

To ensure scheduling feasibility, the following core constraints must be satisfied:

Task Exclusivity Constraint: Each task can only run on one resource at any given time slot:

$$\sum_{j=1}^{m} x_{ij}(t) \leq 1, \quad \forall i, \forall t \quad (7)$$

Resource Capacity Constraint: The total computational demand of all tasks on a single resource cannot exceed its maximum capacity:

$$\sum_{i=1}^{n} x_{ij}(t) \cdot \eta_{ij}(t) \leq C_{j\max}, \quad \forall j, \forall t \quad (8)$$

where $C_{j\max}$ is the maximum computational capacity of resource r_j.

Task Completion Status Constraint: Define the cumulative completion of task t_i at time t as $\Phi_i(t)$:

$$\Phi_i(t) = \frac{1}{w_i} \sum_{\tau=0}^{t} \sum_{j=1}^{m} x_{ij}(\tau) \cdot \eta_{ij}(\tau) \quad (9)$$

When the task completion degree $\Phi_i(t)$ first reaches or exceeds 1, the task is considered complete, and resources should be released. This is indirectly implemented through the constraint optimization process. The completion degree is used to evaluate task progress in the reinforcement learning agent's reward function and implicitly constrains the scheduling strategy: when $\Phi_i(t) \geq 1$, task t_i should be removed from the scheduling queue.

The adaptability of this model is reflected in the dynamic updates of key parameters, which depend on the system's real-time state and the output of the prediction module.

Dynamic Priority Update: Task priority $p_i(t)$ increases with waiting time to prevent task starvation:

$$p_i(t) = p_{base} + \beta \cdot W_i(t) \quad (10)$$

where:

p_{base}: Base priority of task t_i (static).

$W_i(t)$: Cumulative waiting time for task t_i until time t.

β (waiting penalty coefficient, $\beta > 0$) controls the strength of the influence of waiting on priority.

Adaptive Computational Resource Allocation: The effective computational resource $\eta_{ij}(t)$ that a task receives on a resource is not fixed but is dynamically adjusted based on the predicted load $\hat{L}_j(t)$ of the resource (from the LSTM module):

$$\eta_{ij}(t) = \kappa_{ij} \cdot A_j(t) \cdot c_j \quad (11)$$

where:

κ_{ij} (task-resource adaptation factor, $0 < \kappa_{ij} \leq 1$) reflects the execution efficiency of task t_i on resource r_j.

c_j: Peak computational capacity of resource r_j.

$A_j(t)$: Available computational capacity ratio of resource r_j at time t, calculated based on the predicted load:

$$A_j(t) = \max\left(0, 1 - \frac{\hat{L}_j(t)}{C_{j\max}}\right) \quad (12)$$

This equation ensures that when predicted load is high, the computational resources for newly allocated tasks are reduced accordingly, achieving forward-looking load smoothing.

In summary, the optimization problem solved by AML-TSCRA can be fully stated as:

Given the constraints, the objective is to optimize the binary decision variable set $\{x_{ij}(t)\}$ at each time point and determine the corresponding $p_i(t)$ and $\eta_{ij}(t)$ based on the dynamic update formulas (10), (11), and (12), to maximize the comprehensive objective function J defined in equation (2).

This problem is a large-scale dynamic mixed-integer programming problem with nonlinear characteristics. The innovation of the AML-TSCRA framework lies in not directly solving for the exact optimal solution but instead using a RL agent to learn a scheduling strategy π that approximates the optimal solution. The strategy takes system states (including task queues, predicted loads $\hat{L}_j(t)$, etc.) as inputs and directly outputs the binary scheduling decisions $\{x_{ij}(t)\}$ at each time point. By interacting with the environment, the strategy is continuously optimized, ultimately maximizing the objective defined in equation (2) over the long term. The deep learning (LSTM) module

provides accurate $\hat{L}_j(t)$ for equation (12), which is key to the efficient operation of the entire adaptive mechanism.

4. Experiment and Results

4.1. Experimental Setup

To verify the applicability of the AML-TSCRA model in data-intensive service environments such as distributed databases in libraries, we selected the following publicly available datasets with high concurrency and dynamic load characteristics for testing. The task scheduling and resource allocation challenges simulated by these datasets are highly similar to scenarios faced by library systems, such as concurrent query response, peak access to digital resources, and batch processing of metadata. The experiments in this paper were validated using three representative traces: Google Cluster Data [27], SLURM workload logs from the Parallel Workloads Archive [28], and anonymized Amazon EC2 operational logs collected from an internal production environment (see Table 1). These datasets cover task scheduling and resource allocation issues in large-scale distributed computing environments, making them suitable for comparative validation of the AML-TSCRA model proposed in this paper.

Table 1. Dataset Overview

Dataset Name	Sample Size	Feature Dimensions	Main Application Scenarios	Data Collection Method	Data Type
Amazon EC2 Logs	Large-scale	Resource usage, task execution time	Cloud computing resource scheduling and task management	Cloud computing platform logs	Resource scheduling, performance analysis
Google Cluster Data	Large-scale	Compute tasks, resource allocation, load	Distributed computing task scheduling	Google data center logs	Task scheduling, resource allocation
SLURM Workload Manager	Medium-scale	Task status, resource scheduling, node load	High-performance computing scheduling and resource management	High-performance computing job scheduling logs	Task scheduling, resource load management

The Google Cluster Data originates from Google's production data centers and includes detailed records of compute tasks, resource requests, and execution timelines, ideal for studying large-scale distributed scheduling. The SLURM Workload Manager logs are sourced from real high-performance computing (HPC) clusters archived in the Parallel Workloads Archive, capturing job submission times, resource allocations, and node loads. The Amazon EC2 Logs reflect virtual machine lifecycle events and resource utilization patterns from a cloud platform; while not part of a standard public benchmark, they were collected internally under strict anonymization protocols to preserve privacy. Compared to Google Cluster Data, which emphasizes coarse-grained task orchestration, EC2 logs focus on fine-grained VM-level scheduling, whereas SLURM traces target HPC batch job management. Together, they provide complementary validation across cloud, distributed, and HPC environments. Since large-scale, open-source datasets specific to library distributed clusters are extremely scarce, we utilize these foundational surrogate datasets by strictly mapping their workload distributions to genuine library scenarios. In our experimental evaluation, short, bursty tasks are mapped to represent user-facing library portal queries and PDF downloads, whereas long-duration, computationally heavy tasks simulate institutional repository migrations and massive RDM processing.

The hardware and software configurations used in this experiment are shown in Table 2. The hardware configuration provides powerful computational capabilities, supporting large-scale data processing and task scheduling optimization. The combination of an Intel Xeon Gold 6248 CPU and an NVIDIA Tesla V100 GPU efficiently handles deep learning training and parallel computing. With 256GB of memory ensuring system stability for large-scale computations, and a 2TB SSD providing fast data reading and storage, the setup ensures reproducibility of the experiments. The operating system and deep learning framework versions are also optimized for efficient task scheduling and resource allocation.

Table 2. Hardware and Software Configuration

Configuration Category	Device/Software	Model/Version	Description
Hardware	Processor (CPU)	Intel Xeon Gold 6248	20 cores, 3.0 GHz, suitable for multi-task parallel

			processing.
	Graphics Card (GPU)	NVIDIA Tesla V100	16GB HBM2 memory, optimized for deep learning training and large-scale data processing.
	Memory (RAM)	256GB DDR4	High-performance memory, supports large-scale data processing and parallel computing.
	Storage Device	2TB SSD	High-speed storage for datasets, models, and experiment logs.
Software	Operating System	Ubuntu 20.04	Stable Linux system, supports deep learning frameworks and large-scale data processing.
	Deep Learning Framework	TensorFlow 2.4	Supports GPU acceleration, ideal for training deep learning models and reinforcement learning algorithms.
	Key Libraries	NumPy 1.19.5, Pandas 1.1.5, Keras 2.4.3, scikit-learn 0.24.1	Data processing and machine learning libraries, supports large-scale dataset processing and model training.
	GPU Acceleration Environment	CUDA 11.1	Used for accelerating deep learning training and inference, improving computational efficiency.

The evaluation metrics in Table 3 are derived from the operational logic of AML-TSCRA and directly reflect its dual-loop architecture: LSTM-based load forecasting informs resource provisioning, while the reinforcement learning agent optimizes scheduling actions. Each metric quantifies a specific aspect of system behavior under adaptive control, task completion rate validates policy robustness, resource utilization measures alignment between predicted and actual demand, response time captures latency sensitivity to dynamic scheduling, and throughput reflects aggregate efficiency gains from coordinated allocation.

Table 3. Evaluation Metrics

Metric Name	Formula	Core Interpretation	Numerical Interpretation Reference
Task Completion Rate	$\frac{N_{completed}}{N_{submitted}} \times 100\%$	Proportion of tasks successfully executed without timeout or failure; indicates scheduler reliability under load	>90%: High reliability; <80%: Significant scheduling inefficiency
Resource Utilization	$\frac{\sum_t c(t)}{\sum_t C_{total}} \times 100\%$	Efficiency of allocated CPU/memory usage over time; reflects precision of predictive provisioning	>85%: Efficient allocation; <70%: Over-provisioning or poor prediction
Average Response Time	$\frac{1}{N_{completed}} \sum_{i=1}^{N_{completed}} (T_{finish,i} - T_{submit,i})$	End-to-end latency per task; sensitive to queuing delays and scheduling prioritization	<1.3 s: Low latency; >1.8 s: Noticeable user-perceived delay
System Throughput	$\frac{N_{completed}}{T_{total}}$ (task/ second)	Aggregate processing capacity under adaptive scheduling; measures scalability and concurrency handling	Higher is better; AML-TSCRA achieves ~1.2× baseline on EC2 Logs

4.2. Baselines

To validate the AML-TSCRA model proposed in this paper, we selected two classic methods (Classic) and two state-of-the-art (SOTA) methods as baselines, ensuring a comprehensive evaluation of its performance in distributed task scheduling and resource allocation.

One of the classic methods is Load Balancing Scheduling, which schedules tasks based on computational demand and node load to ensure balanced resource distribution[29]. This method relies on static rules and cannot adapt to dynamic task changes and load fluctuations, resulting in poor performance in dynamic environments. Another classic method is Shortest Job First (SJF), which prioritizes tasks with smaller computational demands to reduce average completion time[30]. SJF works effectively in low-load, stable environments, but in high-concurrency and fluctuating load environments, it fails to adjust task priorities and resource allocation efficiently, leading to low resource utilization.

Among the SOTA methods, RL-based Dynamic Task Scheduling uses system feedback to dynamically adjust the scheduling strategy[31]. Another SOTA method is Deep Q-Network (DQN) combined with adaptive resource allocation[32]. While these two baselines share architectural overlap with our RL agent, they operate in a purely reactive manner based on current system states. They lack the integrated closed-loop mechanism, treating prediction and scheduling as isolated steps. The two learning-based baselines (RL-based dynamic task scheduling and DQN with adaptive resource allocation) are included in the quantitative

comparison in Section 4.3, and are further analyzed in terms of convergence and robustness in Sections 4.3 and 4.5.

Compared with these baselines, the unique design choice of AML-TSCRA lies in embedding LSTM-based forecasting directly into the RL state space to form a closed-loop feedback mechanism. This proactive architecture is specifically tailored to absorb the bursty access patterns and dynamic fluctuations in resource demand typical of library services, providing new ideas for improving the responsiveness and resource utilization of digital library systems.

4.3. Quantitative Results

To evaluate AML-TSCRA, we compared it with four baseline methods on the Amazon EC2 Logs, Google Cluster Data, and SLURM Workload Manager datasets: two heuristic methods (SJF scheduling and load balancing scheduling) and two learning-based methods (RL-based dynamic task scheduling and DQN with adaptive resource allocation), as shown in Table 4. Figure 5 further presents the convergence behavior of the three learning-based methods. Overall, AML-TSCRA performed best on Amazon EC2 Logs, showed moderate gains on Google Cluster Data, and achieved smaller but still consistent advantages on the more heterogeneous SLURM workload.

Table 4. Performance Comparison of AML-TSCRA and Baseline Methods (mean ± std over 10 runs)

Method	Dataset	Task Completion Rate (%)	Resource Utilization (%)	Average Response Time (s)	System Throughput (tasks/s)	p-value (Task Completion Rate)	p-value (Resource Utilization)
AML-TSCRA	Amazon EC2 Logs	88.9 ± 1.6	81.8 ± 2.4	1.31 ± 0.10	23.7 ± 1.9	–	–
AML-TSCRA	Google Cluster Data	85.6 ± 2.3	78.6 ± 3.0	1.46 ± 0.13	22.4 ± 2.1	–	–
AML-TSCRA	SLURM Workload Manager	83.8 ± 2.8	76.9 ± 3.4	1.58 ± 0.16	21.6 ± 2.4	–	–
Load Balancing Scheduling	Amazon EC2 Logs	81.7 ± 2.7	75.2 ± 2.9	1.63 ± 0.14	20.9 ± 1.6	p = 0.004	p = 0.018
Load Balancing Scheduling	Google Cluster Data	79.8 ± 2.9	73.9 ± 3.2	1.71 ± 0.16	20.3 ± 1.7	p = 0.012	p = 0.041
Load Balancing Scheduling	SLURM Workload Manager	77.9 ± 3.3	72.5 ± 3.6	1.83 ± 0.19	19.8 ± 1.9	p = 0.031	p = 0.074
SJF Scheduling	Amazon EC2 Logs	82.9 ± 2.4	73.8 ± 3.0	1.52 ± 0.12	20.1 ± 1.8	p = 0.009	p = 0.011
SJF Scheduling	Google Cluster Data	80.7 ± 2.8	72.6 ± 3.3	1.60 ± 0.14	19.7 ± 1.9	p = 0.026	p = 0.037
SJF Scheduling	SLURM Workload Manager	78.6 ± 3.1	71.4 ± 3.5	1.76 ± 0.17	19.2 ± 2.0	p = 0.048	p = 0.083
RL-based Dynamic Task Scheduling	Amazon EC2 Logs	85.9 ± 2.2	77.6 ± 3.0	1.40 ± 0.12	22.4 ± 1.9	p = 0.038	p = 0.071
RL-based Dynamic Task Scheduling	Google Cluster Data	83.0 ± 2.7	75.0 ± 3.4	1.54 ± 0.15	21.2 ± 2.1	p = 0.081	p = 0.108
RL-based Dynamic Task Scheduling	SLURM Workload Manager	81.7 ± 3.1	73.6 ± 3.7	1.67 ± 0.18	20.6 ± 2.3	p = 0.104	p = 0.146
DQN +	Amazon EC2	85.1 ± 2.5	78.5 ± 2.8	1.44 ± 0.13	22.0 ± 2.0	p = 0.061	p = 0.089

Adaptive Resource Allocation	Logs						
DQN + Adaptive Resource Allocation	Google Cluster Data	82.8 ± 2.9	75.9 ± 3.1	1.57 ± 0.16	21.1 ± 2.2	p = 0.097	p = 0.094
DQN + Adaptive Resource Allocation	SLURM Workload Manager	80.8 ± 3.2	74.2 ± 3.5	1.69 ± 0.19	20.2 ± 2.4	p = 0.137	p = 0.121

Note: p-values are based on two-sample t-tests against AML-TSCRA on the same dataset. AML-TSCRA consistently outperformed the heuristic baselines and retained an overall advantage over the two learning-based baselines. SJF was relatively more competitive than load balancing scheduling in response time, while RL-based scheduling showed relatively stronger completion rate and throughput, and DQN showed relatively stronger resource utilization.

Figure 5 shows the training loss and task completion dynamics of the three learning-based methods across the three datasets. AML-TSCRA converged faster and more smoothly, with the clearest advantage on Amazon EC2 Logs and a smaller gap on the more heterogeneous SLURM workload.

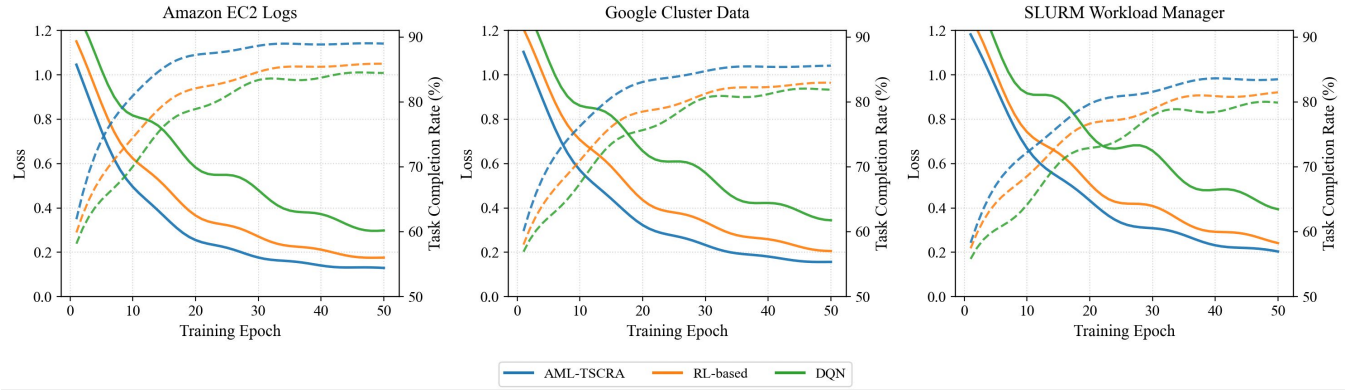

Figure 5. Training loss (solid lines, left axis) and task completion rate (dashed lines, right axis) of AML-TSCRA, RL-based Dynamic Task Scheduling, and DQN-based adaptive resource allocation across three datasets.

This advantage is mainly due to the staged design of AML-TSCRA. The LSTM-based load prediction module provides prior workload information through offline pretraining, narrowing the exploration space for reinforcement learning. Meanwhile, the joint optimization of scheduling and resource allocation makes the reward signal less sparse and improves learning efficiency. By contrast, the DQN method must explore a more dynamic state space directly, while the RL-based scheduling method lacks explicit predictive guidance under bursty workloads.

From an engineering perspective, faster convergence can shorten adaptation time, and smoother training behavior can reduce the risk of online adjustment in continuous-service library systems. Therefore, Figure 5 supports the practical value of AML-TSCRA, although further deployment-oriented validation is still needed.

4.4. Qualitative Results

To demonstrate the advantages and limitations of the AML-TSCRA model, we selected one successful case and one failure case for analysis, and provided an in-depth examination of the model's performance using data and visualizations.

In the successful case on the Amazon EC2 Logs dataset, the AML-TSCRA model successfully scheduled a high-priority task. The model accurately predicted the system load and allocated the task to a resource node with relatively lighter utilization at that moment. As a result, the task execution time was significantly reduced, and the target node's resource utilization increased moderately after allocation, consistent with efficient use of available capacity. Figure 6 (a) shows the resource allocation heatmap, which illustrates that the high-priority task was assigned to nodes in the darker regions, corresponding to lower loads at the time of scheduling. This directly reflects the effectiveness of the AML-TSCRA scheduling strategy, which jointly leverages load prediction and task priority to avoid resource contention and improve overall system responsiveness.

However, on the SLURM Workload Manager dataset, AML-TSCRA failed to effectively handle a low-priority large-scale computation task. The model misjudged the task's resource requirement and adopted an overly conservative scheduling strategy, distributing the task across several moderately loaded nodes rather than consolidating it on nodes with sufficient available capacity. As a result, the allocated computational resources were insufficient, leading to suboptimal execution performance and inefficient resource utilization. Figure 6 (b) shows the resource allocation heatmap for this case, where the large task is fragmented across multiple nodes with moderate loads (visible as star markers in regions of similar intensity). This pattern reflects a deviation in the load prediction module when handling large tasks, prioritizing load balancing over execution efficiency, and ultimately results in a scheduling decision that fails to meet the task's true resource demands.

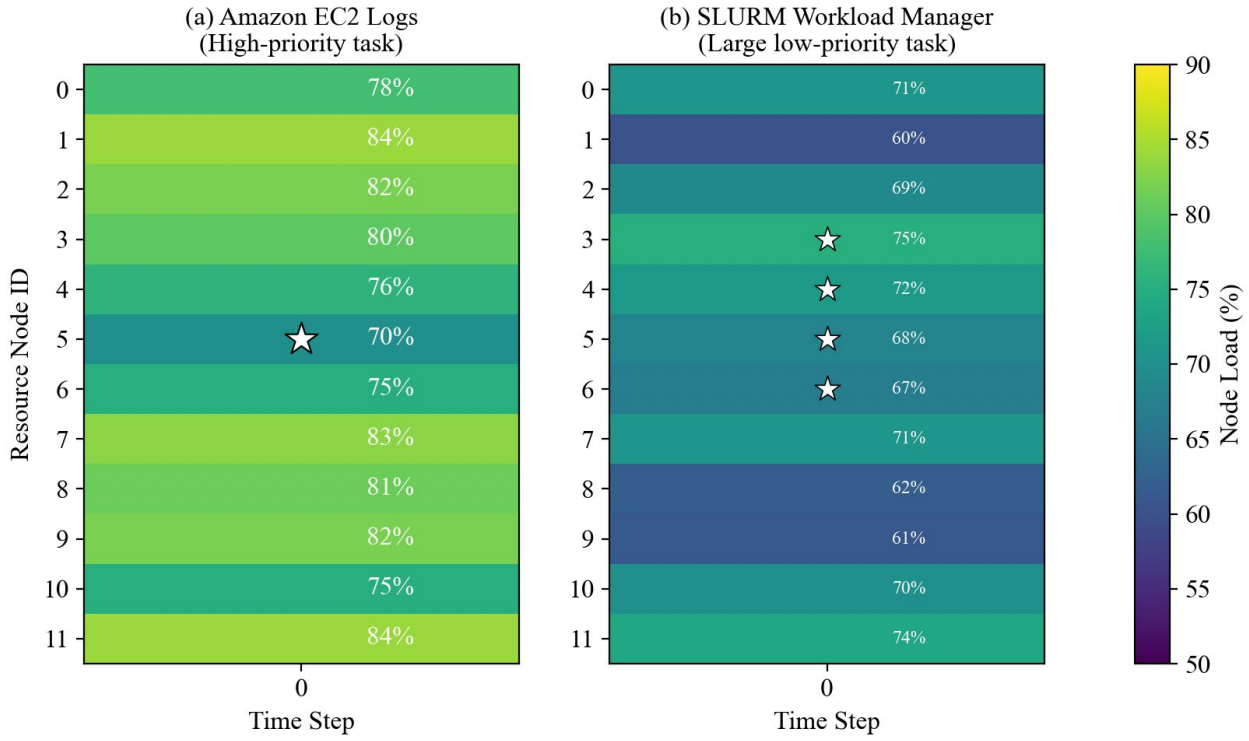

Figure 6. Resource allocation heatmaps of AML-TSCRA: (a) successful scheduling of a high-priority task on Amazon EC2 Logs; (b) fragmented allocation of a large low-priority task on SLURM Workload Manager.

Analysis indicates a significant prediction error for large tasks, with the predicted demand falling noticeably short of the actual demand. Technically, this systematic underestimation occurs because massive computational tasks represent rare, long-tail events in the training data. Consequently, the LSTM module, which is optimized to minimize average errors, inherently smooths out these sparse peak demands.

Despite this, the AML-TSCRA model performed well in most scenarios. matic error, possible architectural modifications for future work include: (1) integrating a Temporal Attention Mechanism to enhance the model's sensitivity to sudden workload bursts, and (2) employing an asymmetric loss function during LSTM training to penalize resource underestimation more heavily than overestimation.

4.5. Robustness Validation

To assess the adaptability of AML-TSCRA in dynamic, non-ideal environments, we conducted robustness testing by injecting Gaussian noise of varying intensities (σ = 0.1–0.5) into the input data and mixing tasks with different computational complexities, measured by the variance of task demand w_i. All tests were performed on the Amazon EC2 Logs, Google Cluster Data, and SLURM Workload Manager datasets.

Figure 7 shows the performance trends of AML-TSCRA and the learning-based baselines under increasing noise and task heterogeneity. Overall, AML-TSCRA remains relatively more stable under perturbation, although the degree of degradation differs across datasets. Under low-to-moderate noise, declines in task completion rate and resource utilization are limited, whereas under higher noise levels and more heterogeneous task mixtures, performance drops become more visible, especially on the SLURM workload.

This relative robustness is likely related to the model design. The LSTM-based load prediction module can

partially smooth short-term input fluctuations and provide more stable load estimates for downstream decisions. In addition, the reinforcement learning component retains a certain level of policy generalization under perturbed environments. The coordination between dynamic priority adjustment and adaptive resource allocation also helps reduce the effect of local prediction errors on overall system behavior.

By contrast, methods without explicit predictive guidance are generally more sensitive to instantaneous load disturbances and tend to degrade faster as noise increases. This difference becomes more apparent when task mixtures are more heterogeneous.

Overall, the robustness results support the effectiveness of the AML-TSCRA prediction-scheduling-allocation framework in noisy environments. Although performance still declines as perturbation intensifies, the model maintains comparatively better stability across datasets, indicating its potential value for dynamic service scenarios such as continuous-access digital library systems and data-intensive backend services.

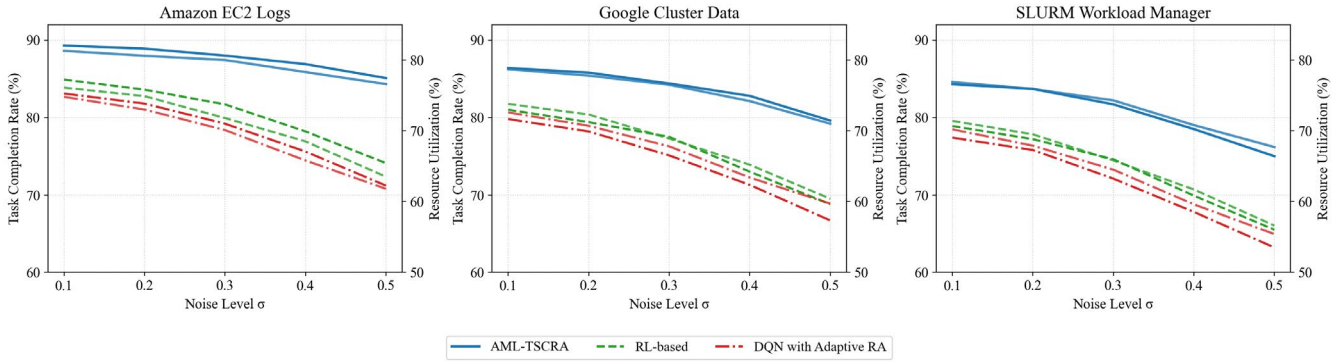

Figure 7. Performance robustness of AML-TSCRA and learning-based baselines under Gaussian input noise across three datasets.

4.6. Ablation Study

To verify the contribution of each core component of the AML-TSCRA model, we conducted an ablation study by sequentially removing the load prediction, task scheduling, and resource allocation modules. The experiments were conducted on the Amazon EC2 Logs and Google Cluster Data datasets, and the results are shown in Table 5.

Table 5. Ablation Study Results

Method	Dataset	Task Completion Rate (%)	Resource Utilization (%)	Average Response Time (s)	System Throughput (tasks/s)
AML-TSCRA	Amazon EC2 Logs	88.8 ± 1.8	81.6 ± 2.7	1.33 ± 0.11	23.5 ± 2.0
AML-TSCRA	Google Cluster Data	85.8 ± 2.5	78.9 ± 3.2	1.48 ± 0.14	22.2 ± 2.3
Remove Load Prediction	Amazon EC2 Logs	77.3 ± 3.5	70.6 ± 3.1	1.94 ± 0.17	18.5 ± 1.9
Remove Load Prediction	Google Cluster Data	76.2 ± 3.9	68.8 ± 3.8	2.01 ± 0.18	18.0 ± 2.1
Remove Task Scheduling	Amazon EC2 Logs	80.8 ± 2.9	74.1 ± 3.3	1.74 ± 0.13	19.6 ± 1.8
Remove Task Scheduling	Google Cluster Data	79.4 ± 3.1	72.9 ± 3.5	1.70 ± 0.12	19.8 ± 1.9
Remove Resource Allocation	Amazon EC2 Logs	84.5 ± 2.7	75.0 ± 3.4	1.44 ± 0.09	22.0 ± 1.9
Remove Resource	Google Cluster Data	82.7 ± 2.8	73.4 ± 3.7	1.55 ± 0.11	21.4 ± 2.0

Allocation					

The ablation results show that all three modules contribute to system performance, although their effects differ somewhat across datasets and metrics. Overall, the complete AML-TSCRA model achieved the best results, and removing any module led to declines in task completion rate, resource utilization, response time, and throughput.

Removing the load prediction module caused the largest overall degradation, particularly in response time and resource utilization. Without future workload estimation, the scheduler became more reactive to short-term load changes and less able to avoid upcoming peaks, resulting in slower response and lower allocation efficiency.

When the task scheduling module was replaced with a round-robin strategy, task completion rate and throughput dropped clearly. This indicates that dynamic priority-aware scheduling is important for maintaining processing efficiency under heterogeneous workloads, although its effect was not fully uniform across datasets and metrics.

Removing the resource allocation module mainly reduced resource utilization, while also causing moderate declines in the other indicators. Replacing the dynamic allocation formula $\eta_{ij}(t)$ with simple average allocation limited the system's ability to match resources to task characteristics and real-time system status, leading to lower allocation efficiency.

Overall, the advantage of AML-TSCRA comes from the interaction among load prediction, task scheduling, and resource allocation. Load prediction provides forward-looking state information, scheduling converts it into execution priorities, and resource allocation supports efficient execution of those decisions. These results suggest that the model's performance gain depends on both individual modules and their coordination.

The experiments further suggest that this modular collaborative architecture is suitable for dynamic service environments such as intelligent library systems, where forecasting, priority scheduling, and adaptive resource allocation can jointly improve service quality and backend efficiency.

5. Discussion and Conclusion

In this study, we addressed dynamic resource management challenges in modern digital libraries by developing AML-TSCRA. Unlike isolated scheduling models, the proposed framework establishes a proactive closed-loop mechanism that integrates LSTM-based load forecasting with adaptive reinforcement learning scheduling. Evaluated on three real-world datasets, Amazon EC2 Logs, Google Cluster Data, and SLURM Workload Manager, AML-TSCRA achieved its strongest performance on Amazon EC2 Logs, with a task completion rate of 88.9%, resource utilization of 81.8%, an average response time of 1.31 s, and system throughput of 23.7 tasks/s. It also showed moderate gains on Google Cluster Data and smaller but still consistent advantages on the more heterogeneous SLURM workload. The ablation results further confirm that load prediction, task scheduling, and resource allocation all contribute to the overall performance of the framework. In addition, the robustness experiments suggest that AML-TSCRA degrades more gradually than the learning-based baselines under noisy conditions, indicating a reasonable level of robustness.

The practical value of AML-TSCRA lies in its coordinated design. Offline pretraining of the load prediction module narrows the exploration space for reinforcement learning, while the joint design of scheduling and resource allocation improves decision consistency under dynamic workloads. For digital library systems and research data management environments, this architecture is particularly relevant because bursty user-facing access and latency-insensitive background processing often coexist. From an operational perspective, the observed gains in resource utilization suggest that proactive provisioning may reduce idle standby capacity and improve backend efficiency in medium-scale library data centers. In this sense, the framework provides not only algorithmic improvements but also a more deployment-oriented resource orchestration strategy. This practical emphasis is also consistent with the revised focus on deployment-oriented implications, including operational feasibility, cold-start mitigation, and resource-efficiency gains.

Nevertheless, several limitations remain. First, deploying the model in a new library environment may introduce a cold-start problem when historical logs are limited, which can reduce forecast accuracy at the initial stage. Second, although the RL and LSTM components require relatively high computational cost during offline training, the online inference stage remains lightweight enough for deployment on standard CPU servers once the model has been trained. In addition, rare large-scale tasks remain more difficult to model accurately than common workloads. Future work will therefore focus on transfer learning for cold-start mitigation, attention-based modeling for rare large tasks, explicit task dependency modeling under extreme workload shifts, and deeper integration with Library and Information Science practices.

In summary, AML-TSCRA provides a domain-aware framework for intelligent resource orchestration by combining foresight, adaptability, and efficiency. Rather than relying on purely reactive scheduling, it addresses the bursty and heterogeneous workload patterns common in digital library infrastructures. These results suggest that the proposed framework has practical potential for improving service stability and resource efficiency in modern academic computing environments.

Acknowledgements

Researching on Innovative Digital Literacy Education in Higher Vocational Colleges Empowered by Generative Artificial Intelligence(Approval No.145J111)

References

[1] Yakubu, A. S., Yagana, A. A., & Umar, S. I. Y. (2023). Investigating librarians ' intention to use artificial intelligence for effective library service delivery: A partial least square-structural equation modeling-based approach. Dutse Journal of Pure and Applied Sciences, 9(1b), 1-14.

[2] Dube, T. V. (2025). Research data management in academic libraries: institutional repositories as a reservoir for research data. Library Management, 46(5), 319-331.

[3] Fahimullah, M., Ahvar, S., Agarwal, M., & Trocan, M. (2024). Machine learning-based solutions for resource management in fog computing. Multimedia Tools and Applications, 83(8), 23019-23045.

[4] Kulkanjanapiban, P., Silwattananusarn, T., & Lambovska, M. (2025). Research on AI-driven innovations and services in academic libraries: A bibliometric and systematic literature review. Journal of Data and Information Science, 10(4), 146-196.

[5] Anjum, S., Upadhyay, D., Singh, K., & Upadhyay, P. (2024, March). Machine learning-based resource allocation algorithms for 6G networks. In 2024 2nd International Conference on Disruptive Technologies (ICDT) (pp. 1086-1091). IEEE.

[6] Zhou, H. (2023). A novel approach to cloud resource management: hybrid machine learning and task scheduling. Journal of Grid Computing, 21(4), 68.

[7] Li, Y., Zhang, X., Zeng, T., Duan, J., Wu, C., Wu, D., & Chen, X. (2023). Task placement and resource allocation for edge machine learning: A gnn-based multi-agent reinforcement learning paradigm. IEEE Transactions on Parallel and Distributed Systems, 34(12), 3073-3089.

[8] Zhou, G., Tian, W., Buyya, R., Xue, R., & Song, L. (2024). Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. Artificial Intelligence Review, 57(5), 124.

[9] Li, P., Xiao, Z., Wang, X., Huang, K., Huang, Y., & Gao, H. (2023). EPtask: Deep reinforcement learning based energy-efficient and priority-aware task scheduling for dynamic vehicular edge computing. IEEE Transactions on Intelligent Vehicles, 9(1), 1830-1846.

[10] Chahare, P. B. (2024). Cloud technology for enhanced academic library services: A comprehensive review. InSight Bulletin, 1(2), 1-5.

[11] Sinha, P., A, S., & Sinha, M. K. (2025). Research data management services in academic libraries: A comparative study of South Asia and Southeast Asia. Global Knowledge, Memory and Communication, 74(3-4), 777-793.

[12] Bhat, J. A., & Hasan, S. (2024). A comprehensive overview of digital preservation in the digital library landscape. Academic Libraries, 220.

[13] Saidi, K., & Bardou, D. (2023). Task scheduling and VM placement to resource allocation in Cloud computing: challenges and opportunities. Cluster Computing, 26(5), 3069-3087.

[14] Pal, S., Jhanjhi, N. Z., Abdulbaqi, A. S., Akila, D., Alsubaei, F. S., & Almazroi, A. A. (2023). An intelligent task scheduling model for hybrid internet of things and cloud environment for big data applications. Sustainability, 15(6), 5104.

[15] Tang, S., Yu, Y., Wang, H., Wang, G., Chen, W., Xu, Z., ... & Gao, W. (2023). A survey on scheduling techniques in computing and network convergence. IEEE Communications Surveys & Tutorials, 26(1), 160-195.

[16] Abraham, O. L., Ngadi, M. A. B., Sharif, J. B. M., & Sidik, M. K. M. (2024). Task scheduling in cloud environment–Techniques, applications, and tools: A systematic literature review. IEEE Access, 12, 138252-138279.

[17] Tang, S., Yu, Y., Wang, H., Wang, G., Chen, W., Xu, Z., ... & Gao, W. (2023). A survey on scheduling techniques in computing and network convergence. IEEE Communications Surveys & Tutorials, 26(1), 160-195.

[18] Banerjee, P., Roy, S., Sinha, A., Hassan, M. M., Burje, S., Agrawal, A., ... & El-Shafai, W. (2023). MTD-DHJS: makespan-optimized task scheduling algorithm for cloud computing with dynamic computational time prediction. IEEE Access, 11, 105578-105618.

[19] Sanjalawe, Y., Al-E'mari, S., Fraihat, S., & Makhadmeh, S. (2025). AI-driven job scheduling in cloud computing: a comprehensive review. Artificial Intelligence Review, 58(7), 197.

[20] Singh, G., & Chaturvedi, A. K. (2024). Hybrid modified particle swarm optimization with genetic algorithm (GA) based workflow scheduling in cloud-fog environment for multi-objective optimization. Cluster Computing, 27(2), 1947-1964.

[21] Kareem Awad, W., Zainol Ariffin, K. A., Nazri, M. Z. A., & Yassen, E. T. (2025). Resource allocation strategies and task scheduling algorithms for cloud computing: A systematic literature review. Journal of Intelligent Systems, 34(1), 20240441.

[22] Jalali Khalil Abadi, Z., Mansouri, N., & Javidi, M. M. (2024). Deep reinforcement learning-based scheduling in distributed systems: a critical review. Knowledge and Information Systems, 66(10), 5709-5782.

[23] Lu, X., Liu, C., Zhu, S., Mao, Y., Lio, P., & Hui, P. (2023). RLPTO: A reinforcement learning-based performance-time optimized task and resource scheduling mechanism for distributed machine learning. IEEE Transactions on Parallel and Distributed Systems, 34(12), 3266-3279.

[24] Agarwal, G., Gupta, S., Ahuja, R., & Rai, A. K. (2023). Multiprocessor task scheduling using multi-objective hybrid genetic Algorithm in Fog–cloud computing. Knowledge-Based Systems, 272, 110563.

[25] Prity, F. S., Uddin, K. A., & Nath, N. (2024). Exploring swarm intelligence optimization techniques for task scheduling in cloud computing: algorithms, performance analysis, and future prospects. Iran Journal of Computer Science, 7(2), 337-358.

[26] Khiat, A., Haddadi, M., & Bahnes, N. (2024). Genetic-based algorithm for task scheduling in fog–cloud environment. Journal of Network and Systems Management, 32(1), 3.

[27] Reiss, C., Wilkes, J., & Hellerstein, J. L. (2011). Google cluster-usage traces: format+ schema. Google Inc., White Paper, 1, 1-14.

[28] Feitelson, D. G., Tsafrir, D., & Krakov, D. (2014). Experience with using the parallel workloads archive. Journal of Parallel and Distributed Computing, 74(10), 2967-2982.

[29] Zhu, L., Zhang, Z., Liu, L., Feng, L., Lin, P., & Zhang, Y. (2023). Online distributed learning-based load-aware heterogeneous vehicular edge computing. IEEE Sensors Journal, 23(15), 17350-17365.

[30] Liu, W. X., Lu, J., Cai, J., Zhu, Y., Ling, S., & Chen, Q. (2021). DRL-PLink: Deep reinforcement learning with private link approach for mix-flow scheduling in software-defined data-center networks. IEEE Transactions on Network and Service Management, 19(2), 1049-1064.

[31] Khan, A. R. (2024). Dynamic load balancing in cloud computing: optimized RL-based clustering with multi-objective optimized task scheduling. Processes, 12(3), 519.

[32] Hurtado Sanchez, J. A., Casilimas, K., & Caicedo Rendon, O. M. (2022). Deep reinforcement learning for resource management on network slicing: A survey. Sensors, 22(8), 3031.