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

# Adaptive Scheduling and Compute Demand Prediction for Animation Rendering Tasks Using Machine Learning

Yi Yang[1,*], Muchen Zhang[2], Zhaopei Wang[3]

[1]School of Art and Design, Dalian Polytechnic University, 116034, Dalian, China
[2]School of Design and Art, Beijing Technology and Business University, 100048, Beijing, China
[3]Department of Management Information Systems, University of Houston–Clear Lake, TX 77058, USA

INTRODUCTION: The animation industry's growing demand for high-resolution, multi-scene rendering has led to highly heterogeneous and dynamic workloads, challenging traditional scheduling systems.
OBJECTIVES: This study aims to overcome the limitations of static or rule-based schedulers by developing an adaptive, learning-driven approach that dynamically matches rendering tasks to compute resources under fluctuating conditions.
METHODS: We propose a machine learning-based scheduling and compute prediction model that integrates multi-dimensional task feature vectors with a lightweight sequence prediction network and a multi-objective optimization strategy. This enables real-time estimation of rendering duration, compute demand, and node load for dynamic resource allocation. The framework explicitly models temporal dependencies across frames and accounts for GPU heterogeneity through unified task representations.
RESULTS: Evaluated on a cluster with 12 representative animation scene categories and 3 GPU architectures, our method shortens average task latency by 11.0%, improves node utilization by 3.9%, and reduces energy consumption by 7.8% over the best baseline, while maintaining robust performance even under severe (20%) state noise.
CONCLUSION: The proposed model demonstrates strong adaptability and efficiency in complex rendering environments, offering a scalable foundation for intelligent, cross-platform scheduling in next-generation rendering infrastructures, particularly beneficial for cloud rendering, virtual production, and energy-constrained GPU clusters, while supporting real-time responsiveness and cost-effective resource management.

**Keywords:** animation rendering task scheduling, machine learning prediction model, compute demand forecasting, adaptive resource allocation, heterogeneous computing clusters

Received on 02 March 2026, accepted on 06 May 2026, published on 27 May 2026



doi: 10.4108/eetsis.12068

*Corresponding author. Email: yangyi81@dlpu.edu.cn

## 1. Introduction

The rapid advancement of animated films, game visual effects, and immersive media has led to increasingly complex rendering tasks. These tasks demand high-resolution outputs, intricate lighting, and diverse scene elements, placing significant pressure on modern production pipelines [1]. Multi-architecture GPUs and multi-node clusters have now become the standard in these pipelines. Consequently, efficient resource scheduling is a critical factor influencing production cycles, cost reduction, and rendering efficiency [2, 3]. However, current scheduling approaches predominantly rely on fixed rules or static load-balancing strategies. These traditional methods struggle to handle the instability caused by increasing task volumes and fluctuating capacities, thereby constraining overall cluster throughput and resource utilization [4]. Moreover, while large-scale industrial schedulers like Google's Sigma [5] and Microsoft's Apollo [6] excel in general-purpose cloud management, they are not inherently tailored for the highly dynamic, multi-source attribute dependencies specific to frame-level rendering tasks.

Previous research has explored heuristic algorithms, hybrid scheduling strategies, and various predictive models aimed at optimizing rendering task management. Yet, significant challenges remain. First, the rendering

complexity at the frame level can fluctuate significantly, and low-dimensional indicators alone are insufficient to capture the full scope of task characteristics, limiting the generalization ability of prediction models. Second, GPU architectures differ in memory capacity, cache organization, and other structural attributes, making it difficult for traditional scheduling methods to account for cross-device load heterogeneity[7]. Crucially, attempting to resolve these issues by simply extending existing deep learning schedulers, such as naively bolting a standalone sequence predictor onto a standard resource allocator, proves highly ineffective. Such functional combinations treat feature modeling and scheduling as decoupled processes. They pass only low-dimensional scalars between modules, causing severe information loss regarding the nonlinear interactions between fine-grained rendering attributes and hardware execution costs. These issues highlight the urgent need for a more adaptive scheduling method capable of dynamically predicting compute fluctuations and optimizing resource allocation in heterogeneous environments.

To address these challenges, this study proposes a tightly coupled framework for animation rendering task scheduling that integrates task representation, temporal prediction, and multi-objective scheduling. While this work builds upon established deep learning techniques rather than introducing fundamentally new mathematical algorithms, our core contribution lies in a novel system-level integration and practical engineering innovation. This framework introduces three key innovations: (1) the construction of a multi-dimensional rendering task representation, which incorporates features such as texture, lighting, material properties, and path depth to enhance feature granularity and discriminative power; (2) a sequential compute prediction model, which leverages a lightweight temporal network to capture load variation patterns and predict compute demand and rendering duration dynamically; and (3) a multi-objective adaptive scheduling strategy, which combines prediction results and node states to enable the dynamic adjustment of resources through weighted optimization of latency, utilization, and energy consumption. These components are designed to address the specific characteristics of real-world rendering pipelines, ensuring system interpretability, scalability, and significant practical value in heterogeneous environments.

## 2. Related Works

### 2.1 Characteristics of Animation Rendering Workloads

Animation rendering tasks are widely used in film pre-rendering, visual effects, virtual production, and cloud rendering platforms. These tasks typically involve path tracing, global illumination, and complex material solving, and their compute demand is highly sensitive to scene structure, lighting conditions, and camera motion [8,9]. For instance, a single frame with detailed textures, dynamic lighting, and multiple reflective surfaces can require hours of computation on a high-performance GPU. As rendering systems evolve toward multi-GPU and cross-node clusters, task scheduling and compute prediction face challenges such as strong resource heterogeneity, significant compute fluctuations, and inconsistent task granularity [10].

Existing rendering load benchmarks are primarily based on publicly available light-tracing scene sets and synthetic scene collections. However, these datasets mainly serve hardware performance evaluation and generally lack joint annotations of task properties and node load[11]. Common evaluation metrics include task completion time, node utilization, cluster throughput, energy consumption, and prediction error metrics such as MAE/MAPE. Nevertheless, most studies tend to optimize a single metric and fail to systematically account for the coupling relationship among latency, energy consumption, and utilization, resulting in a gap between research findings and real production environments[12]. To bridge this gap, it's crucial to develop frameworks that can simultaneously consider multiple objectives and adapt to changing conditions.

### 2.2 Multi-Dimensional Task Representation

Accurate modeling of rendering tasks requires capturing heterogeneous factors such as texture complexity, lighting configuration, material types, and geometric detail. Prior work has shown that low-dimensional features, such as material, lighting, and mesh attributes, can establish a basic correlation with rendering load[13,14]. For example, scenes featuring large areas of flat color may have lower computational demands compared to those with intricate textures and fine details. However, these representations often overlook frame-level dynamics and exhibit limited adaptability to long sequences or scenes with complex motion. Moreover, restricted feature dimensionality is insufficient to capture load variations introduced by advanced materials or dynamic lighting[15]. To address this, our framework employs a high-dimensional, unified task representation that fuses categorical and numerical scene attributes through embedding and normalization, enabling fine-grained discrimination of computational demands. This approach allows for more accurate predictions of compute requirements, which is particularly beneficial for scenes with rapid changes in lighting or object movement.

### 2.3 Temporal Modeling of Compute Demand

Rendering workloads exhibit strong temporal correlations across consecutive frames due to camera motion, lighting transitions, or object interactions. Capturing these dynamics requires sequential modeling capable of forecasting both compute demand (e.g., GPU FLOPs) and rendering duration. While some approaches model resource states using graph neural networks or temporal models to capture inter-node dependencies [16,17], they often neglect task-side evolution, leading to weak task-resource coupling. Furthermore, single-scale prediction limits applicability in

scenarios with abrupt load changes. In contrast, our method leverages lightweight recurrent architectures (e.g., GRUs) to process frame sequences, explicitly modeling short-term fluctuations and long-term trends. This enables proactive prediction of compute requirements before task execution, a principle supported by evidence that temporal context significantly improves load estimation accuracy [18]. By considering the historical data and future projections, our approach provides a more robust solution for managing fluctuating workloads.

## 2.4 Multi-Objective Adaptive Scheduling in Heterogeneous Environments

Effective scheduling in modern render farms must jointly optimize competing objectives: minimizing latency, maximizing resource utilization, and reducing energy consumption. Reinforcement learning-based strategies have demonstrated flexibility in dynamic environments and multi-objective optimization [19, 20, 21]. However, they typically operate reactively without forward-looking predictive inputs, and often struggle to dynamically adapt to the rapid frame-level load shifts inherent in rendering tasks. Meanwhile, studies on heterogeneous GPU mapping highlight how architectural differences impact performance [22, 23], yet most still depend on static rules that cannot proactively anticipate compute surges.

Our scheduling strategy builds on the principle that optimal decisions require three inputs: (1) predicted compute demand and duration, (2) real-time node states (utilization, memory, bandwidth), and (3) configurable weights reflecting operational priorities. By integrating these into a unified cost function, the scheduler performs proactive, multi-objective allocation, departing from reactive or single-metric paradigms [24]. This approach aligns with the insight that system-level efficiency emerges from tight coupling between task dynamics, resource heterogeneity, and adaptive control [25]. Future research could explore more sophisticated optimization techniques, such as evolutionary algorithms or hybrid methods combining reinforcement learning with predictive analytics, to further enhance the adaptability and efficiency of the scheduling strategy.

## 2.5 Synthesis: Toward an Integrated Prediction-Scheduling Framework

The above principles collectively motivate our integrated design. A fundamental architectural distinction between our proposed method and existing deep learning schedulers lies in the depth of integration. Unlike methods that treat feature modeling, temporal prediction, and scheduling as isolated components connected by heuristic interfaces [26], our framework unifies them into a tightly coupled, cascaded pipeline. In our design, high-dimensional task representations directly condition the hidden states of the temporal predictor, whose outputs then dynamically formulate the multi-objective cost function. This architectural continuity ensures that fine-grained feature nuances, such as an abrupt transition in scene geometry, propagate directly into scheduling decisions without intermediate bottlenecking. This end-to-end pipeline is grounded in three foundational ideas: (1) Task complexity must be represented in sufficient dimensionality to reflect real-world rendering variability; (2) Temporal dependencies in compute load must be explicitly modeled to enable foresight; (3) Scheduling decisions must be multi-objective and informed by these forward-looking predictions, not just instantaneous states.

These principles directly address four persistent gaps: (1) absence of unified frameworks combining task, temporal, and node-state modeling; (2) lack of proactive scheduling for dynamic, multi-GPU environments; (3) immaturity of multi-objective optimization in rendering contexts; and (4) insufficiency of existing datasets for end-to-end algorithm validation[27]. Our methodology operationalizes these principles to deliver a generalizable solution for intelligent rendering cluster management. Additionally, exploring the integration of emerging technologies like quantum computing or edge computing into rendering workflows could offer new avenues for optimizing compute resources and enhancing the scalability of rendering pipelines.

# 3. Methodology

This section presents the unified architecture composed of the Task Feature Encoder, Temporal Compute Predictor, and Multi-objective Adaptive Scheduler. The overall framework aims to accurately model compute demand, rendering duration, and node load variations for animation rendering tasks, and to perform proactive multi-objective scheduling in heterogeneous GPU clusters.

## 3.1 Problem Formulation

For clarity of subsequent derivations, this section formalizes the definitions of rendering tasks, node states, prediction targets, and scheduling objectives.

### 3.1.1 Rendering Task Definition

Assume the animation sequence contains $T$ consecutive frames, denoted as:

$$\mathcal{F} = \{f_1, f_2, \dots, f_T\} \tag{1}$$

Each frame $f_t$ corresponds to a rendering task, whose input feature vector is defined as:

$$\mathbf{x}_t \in \mathbb{R}^d \tag{2}$$

where

$d$: feature dimension

$\mathbf{x}_t$ includes quantifiable scene attributes such as texture complexity, number of light sources, material type encoding. To ensure cross-platform consistency, these features are parsed directly from standardized scene files (e.g., Universal Scene Description, USD) and pipeline metadata rather than specific renderer APIs. This engine-agnostic

approach guarantees broad universality across diverse production pipelines.

### 3.1.2 Compute Demand and Rendering Duration

Let

$c_t$: compute demand of frame $t$ (GPU FLOPs)
$l_t$: rendering duration
The objective is to learn the prediction functions:

$$\hat{c}_t = \Phi(\mathbf{x}_{1:t}), \quad \hat{l}_t = \Psi(\mathbf{x}_{1:t}) \tag{3}$$

where $\mathbf{x}_{1:t}$ denotes the feature sequence from frame 1 to frame $t$, capturing temporal dependencies.

### 3.1.3 Node State and Resource Model

Assume the GPU cluster contains $N$ nodes:

$$\mathcal{G} = \{g_1, g_2, \dots, g_N\} \tag{4}$$

The resource state of each node is:

$$\mathbf{s}_n^t = \{u_n^t, m_n^t, b_n^t\} \tag{5}$$

where
$u_n^t$: GPU utilization
$m_n^t$: memory usage
$b_n^t$: bandwidth availability
All values are normalized to the interval [0,1].

### 3.1.4 Scheduling Decision Definition

For frame task $f_t$, the scheduling decision is:

$$a_t \in \{1,2,\dots,N\} \tag{6}$$

Let
$L = \sum_{t=1}^{T} \hat{l}_t$: total latency
$U = \frac{1}{TN}\sum_{t=1}^{T}\sum_{n=1}^{N} u_n^t$: average node utilization
$E = \sum_{t=1}^{T} e_{a_t}^t$: total energy consumption
$\alpha, \beta, \gamma$: weighting coefficients

### 3.1.5 Overall Objective

The unified scheduling objective jointly optimizes latency ($L$), utilization ($U$), and energy consumption ($E$):

$$\min_{\{a_t\}} \ \alpha L + \beta(1-U) + \gamma E \tag{7}$$

where
$\alpha, \beta, \gamma$: objective weights
$L$ total latency
$U$: average utilization
$E$: energy consumption
Equation (7) represents the final optimization objective of this study.

## 3.2 Overall Framework

The proposed adaptive scheduling and compute prediction framework consists of three core modules, Task Feature Encoder, Temporal Compute Predictor, and Multi-objective Adaptive Scheduler, forming a unified pipeline of task input → feature representation → temporal prediction → scheduling decision → GPU execution. The overall framework is illustrated in Figure 1.

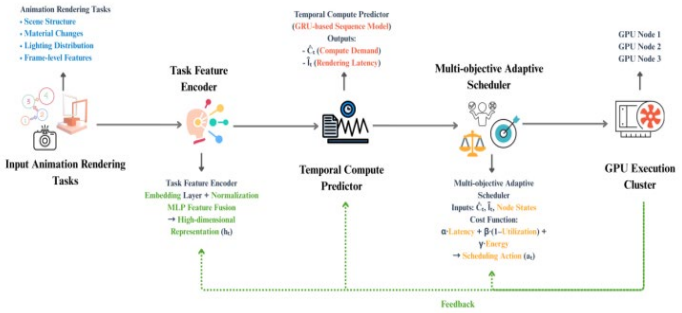


**Figure 1.** Overall Framework

The framework takes animation rendering tasks as the primary input, including scene structure, frame-level material changes, lighting distribution, and other characteristics. These raw features are first transformed into unified high-dimensional representations by the Task Feature Encoder to support subsequent temporal modeling.

The encoded sequential frame representations are then fed into the Temporal Compute Predictor, which captures dynamic trends in rendering load as the shot evolves. Based on a lightweight recurrent network, this module generates two key prediction outputs: compute demand and rendering latency, reflecting the fluctuation patterns of rendering workloads in sequential form. The prediction results are combined with the current cluster state, including node utilization, memory usage, and bandwidth availability, and then passed to the scheduling module.

The Multi-objective Adaptive Scheduler computes a cost value for each candidate node based on the prediction outputs, jointly considering latency, utilization, and energy consumption. Through a cost minimization strategy, it produces the final scheduling action. The entire system operates in a closed-loop manner: after actual execution, the node states are updated to provide more accurate decision support for subsequent tasks. This framework achieves an integrated prediction–scheduling design with strong scalability and stability in heterogeneous GPU cluster environments.

## 3.3 Module Descriptions

This section provides a detailed description of the three major modules of the system: Task Feature Encoder, Temporal Compute Predictor, and Multi-objective Adaptive Scheduler. These modules correspond to Figure 2, Figure 3, and Figure 4, respectively. Each module is described in terms of motivation, principles, implementation, and structural illustration, followed by unified pseudocode.

### 3.3.1 Task Feature Encoder

(1) Motivation
The computational characteristics of rendering tasks are affected by heterogeneous factors such as texture, lighting, materials, and geometric complexity. To address the issues of sparse raw features, inconsistent scales, and strong coupling among modalities, a representation model capable

of integrating multi-modal features is required.

(2) Principles

This module adopts a combination of an embedding layer and a multilayer perceptron (MLP) to map raw features $\mathbf{x}_t \in \mathbb{R}^d$ to a high-dimensional vector$\mathbf{h}_t \in \mathbb{R}^h$. Specifically, categorical features are encoded using embeddings, numerical features are normalized through min-max scaling, and then two fully connected layers are applied for feature fusion.

(3) Implementation

Categorical features (e.g., material types, lighting categories) are first embedded, and numerical features are normalized. The fused representation vector is then produced through the MLP. Formally:

$$\mathbf{e}_t = \text{Embed}(\mathbf{x}_t) \quad (8)$$

$$\mathbf{h}_t = \sigma(W_1\mathbf{e}_t + b_1) \quad (9)$$

where $\mathbf{W}_1 \in \mathbb{R}^{h\times d}$, $\mathbf{b}_1 \in \mathbb{R}^h$ are learnable parameters, and $h$ is the hidden dimension, set to 128 in experiments.

As shown in Figure 2, this module adopts a three-stage architecture and outputs a unified task representation$\mathbf{h}_t$.

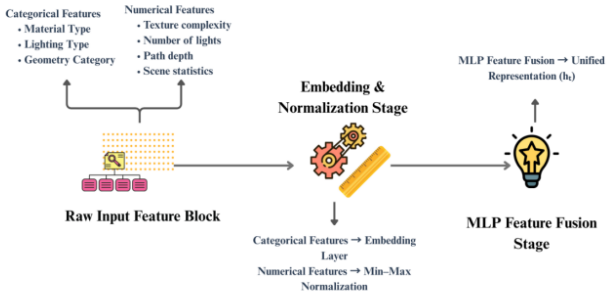


**Figure 2.** Architecture of the Task Feature Encoder Module

### 3.3.2 Temporal Compute Predictor

(1) Motivation

Rendering loads exhibit strong temporal correlations across frames (e.g., due to camera motion or lighting changes). To capture such dynamic patterns, a sequential modeling approach is required to jointly predict compute demand and rendering duration.

(2) Principles and Implementation

This module employs a gated recurrent unit (GRU) to model the feature sequence $\{\mathbf{h}_1, \ldots, \mathbf{h}_t\}$, producing the hidden state $\mathbf{z}_t$, which is then used to predict compute demand $\hat{c}_t$ and rendering duration $\hat{l}_t$:

$$\mathbf{z}_t = \text{GRU}(\mathbf{h}_t, \mathbf{z}_{t-1}) \quad (10)$$

The prediction outputs are:

$$\hat{c}_t = W_c\mathbf{z}_t, \quad \hat{l}_t = W_l\mathbf{z}_t \quad (11)$$

where

$\mathbf{z}_t$: temporal hidden state

$W_c, W_l$: linear projection matrices for prediction

(3) Implementation

The system sequentially feeds consecutive frames into the GRU and generates predictions at each timestep. This architecture effectively captures both short-term fluctuations and long-term trends, yielding stable predictions sensitive to scene variations. Although more complex models like LSTMs and Transformers possess stronger capacity, GRU was selected because it achieves the optimal trade-off for real-time scheduling: it delivers comparable accuracy to LSTM with lower computational overhead, and avoids the heavy inference latency of Transformers, satisfying the strict millisecond-level decision requirement.

Figure 3 illustrates the sequence structure of the Temporal Compute Predictor: the left part shows the feature sequence produced by the Task Feature Encoder, the middle part displays the GRU units unfolded across time, and the right part shows the predictions of compute demand $\hat{c}_t$ and rendering duration $\hat{l}_t$. Arrows in Figure 3 indicate temporal dependencies and the propagation of hidden states across GRU blocks.

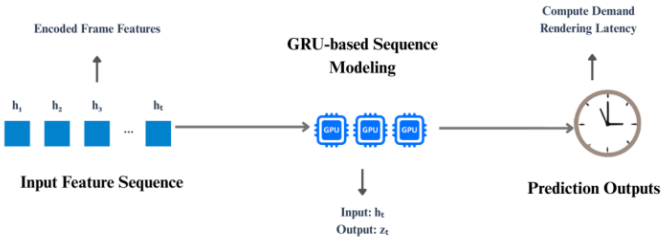


**Figure 3.** Architecture of the Temporal Compute Predictor Module

### 3.3.3 Multi-objective Adaptive Scheduler

(1) Motivation

Scheduling decisions must jointly consider latency, resource utilization, and energy consumption. However, traditional approaches often rely on single-objective or rule-based strategies, which cannot balance different metrics or perform proactive scheduling based on prediction outputs. Therefore, a multi-objective scheduling module capable of incorporating predicted information and real-time node states is needed.

(2) Principles

The scheduler computes a cost function using predicted compute demand $\hat{c}_t$, predicted rendering duration $\hat{l}_t$, and node state $\mathbf{s}_n^t = \{u_n^t, m_n^t, b_n^t\}$ (representing utilization, memory consumption, and bandwidth availability):

$$\mathcal{C}(t,n) = \alpha\hat{l}_t + \beta(1-u_n^t) + \gamma e_n^t \quad (12)$$

The final scheduling decision is:

$$a_t = arg\min_n \mathcal{C}(t,n) \quad (13)$$

where:

$e_n^t$: energy consumption of node nnn when executing the task

$\alpha, \beta, \gamma$: weights for the multi-objective cost

(3) Implementation

The strategy achieves multi-objective trade-offs through the weight parameters $\alpha, \beta, \gamma$, and leverages prediction outputs to perform proactive resource allocation (as shown in Figure 4).

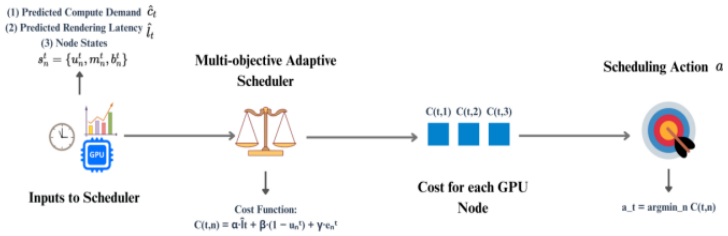


**Figure 4.** Architecture of the Multi-objective Adaptive Scheduler Module

## 3.4 Objective Function & Optimization

This section provides the complete derivation of the joint optimization objectives used in this study, including compute prediction loss, rendering-duration prediction loss, the multi-objective scheduling function, and the overall optimization strategy. All variables are defined upon first use.

To ensure that the Temporal Compute Predictor accurately estimates frame-level compute demand and rendering duration, two types of prediction losses are constructed.

Let:

$c_t$: ground-truth compute demand

$\hat{c}_t$: predicted compute demand

$l_t$: ground-truth rendering duration

$\hat{l}_t$: predicted rendering duration

$T$: total number of frames in the sequence

The compute-demand prediction loss is defined using Mean Absolute Error (MAE):

$$\mathcal{L}_c = \frac{1}{T}\sum_{t=1}^{T} | c_t - \hat{c}_t | \tag{14}$$

The rendering-duration prediction loss is:

$$\mathcal{L}_l = \frac{1}{T}\sum_{t=1}^{T} | l_t - \hat{l}_t | \tag{15}$$

The combined prediction loss is obtained through weighted summation:

$$\mathcal{L}_{pred} = \lambda_c \mathcal{L}_c + \lambda_l \mathcal{L}_l \tag{16}$$

where $\lambda_c$ and $\lambda_l$ are weighting coefficients for the two prediction tasks.

The multi-objective scheduling module jointly optimizes latency, energy consumption, and node utilization.

Let:

$a_t$: index of the node selected for frame $t$

$u_n^t$: utilization of node nnn at frame $t$

$e_n^t$: energy consumption of node nnn at frame $t$

$\hat{l}_t$: predicted rendering duration

Latency Objective

The total sequence latency is defined as:

$$L = \sum_{t=1}^{T} \hat{l}_t \tag{17}$$

Average Node Utilization

$$U = \frac{1}{TN}\sum_{t=1}^{T}\sum_{n=1}^{N} u_n^t \tag{18}$$

Energy Objective

$$E = \sum_{t=1}^{T} e_{a_t}^t \tag{19}$$

Scheduling Cost Function

The scheduler evaluates each node using the following weighted multi-objective cost function:

$$\mathcal{C}(t,n) = \alpha\hat{l}_t + \beta(1-u_n^t) + \gamma e_n^t \tag{20}$$

The target weights ($\alpha$, $\beta$, $\gamma$) are empirically tuned via grid search based on specific production priorities (set to 0.5, 0.3, and 0.2, respectively, in our core experiments to prioritize latency). Although this linear combination simplifies potential non-linear coupling, it is deliberately adopted to guarantee the computational efficiency strictly required for millisecond-level scheduling. While an online adaptive adjustment mechanism (e.g., utilizing Bayesian optimization for dynamic scenario-switching) is not currently implemented, exploring such intelligent weight-tuning methods and addressing non-linear dynamics remain key directions for future optimization.

The final scheduling decision is:

$$a_t = arg\min_n \mathcal{C}(t,n) \tag{21}$$

Overall Scheduling Loss

$$\mathcal{L}_{sch} = \alpha L + \beta(1-U) + \gamma E \tag{22}$$

Combining the prediction and scheduling objectives yields the final unified loss:

$$\mathcal{L} = \mathcal{L}_{pred} + \eta\mathcal{L}_{sch} \tag{23}$$

where $\eta$ controls the relative contributions of the prediction and scheduling modules.

Rendering tasks are subject to resource constraints; thus, scheduling must satisfy:

Each task must be assigned to exactly one node

$$\sum n = 1NI(at = n) = 1(42)\sum_{n=1}^{N}\mathbb{I}(a_t = n) = 1 \tag{24}$$

Memory Constraint

Let $m_t$ be the memory requirement of the task and $M_{a_t}$ the memory capacity of the assigned node:

$$m_t \le M_{a_t} \tag{25}$$

Bandwidth Constraint

Let $b_{\min}$ be the minimum bandwidth requirement and $b_{a_t}^t$ the bandwidth of the assigned node:

$$b_{a_t}^t \ge b_{\min} \tag{26}$$

Training is performed through alternating optimization:

Update Prediction Module Parameters θ

$$\theta \leftarrow \theta - \eta_p \nabla_\theta \mathcal{L}_{pred} \tag{27}$$

Update Scheduling Strategy (Greedy + Local Search)

$$a_t = arg\min_n \mathcal{C}(t,n) \tag{28}$$

where:

$\eta_p$: learning rate of the prediction module

θ: parameter set of the prediction model

This alternating optimization approach maintains high prediction accuracy while improving the adaptability and robustness of the scheduling strategy.

# 4. Results and Analysis

## 4.1 Experimental Setup

This section introduces the composition of the datasets, hardware environment, evaluation metrics, and overall experimental workflow. To ensure that the evaluation covers diverse rendering load patterns, different GPU

architectures, and multi-scene characteristics, the experiments employ three datasets, StudioRender-12K, Blender-PhysLight Scenes, and Multi-GPU TraceSet, jointly constructing a realistic animation-rendering environment with compute prediction and scheduling requirements.

StudioRender-12K is a self-built dataset sourced from real production pipelines. Ensuring objective reproducibility, it requires no manual annotation: input features are automatically parsed from standardized scene files, and ground-truth labels (rendering duration and compute demand) are directly extracted from system execution logs. It covers extreme workloads (e.g., particle effects and high-polygon scenes) and is used to train the prediction model. Blender-PhysLight Scenes contains strong temporal dependencies and high-resolution textures; its volumetric scattering causes abrupt compute surges, testing temporal prediction sensitivity. Multi-GPU TraceSet records node load traces from 3 different GPU architectures, capturing fine-grained metrics (memory bandwidth, temperature, real-time power) to evaluate scheduler stability in heterogeneous environments. The detailed statistics, feature dimensions, and primary uses of these three datasets are summarized in Table 1.

Table 1. Dataset Statistics

| Dataset | Scale | Content | Feature Dim. | Primary Use |
|---|---|---|---|---|
| StudioRender-12K | 12,487 frame logs | Real rendering tasks, including materials, lighting, path depth, etc. | 32-dim | Model training and overall validation |
| Blender-PhysLight Scenes | 40 scenes × 300 frames | Volumetric light, multi-scattering materials, area-light sequences | 28-dim | Temporal prediction capability evaluation |
| Multi-GPU TraceSet | 8,000 records from 3 GPU types | Rendering time, memory usage, power, bandwidth, etc. | 12-dim | Heterogeneous scheduling and energy analysis |

The hardware setup used in the experiments is shown in Table 2. Three GPU architectures are included to simulate a real heterogeneous computing environment.

Table 2. Experimental Hardware Configuration

| Node | GPU | CPU | RAM | Storage | OS |
|---|---|---|---|---|---|
| Node-A | NVIDIA RTX 3060 (12GB) | Ryzen 5 5600X | 32GB | 1TB SSD | Ubuntu 22.04 |
| Node-B | NVIDIA RTX 3070 (8GB) | Ryzen 7 3700X | 32GB | 1TB SSD | Ubuntu 22.04 |
| Node-C | NVIDIA RTX 4070 (12GB) | Intel i5-12600K | 32GB | 1TB SSD | Ubuntu 22.04 |

The selected GPUs cover both Ampere and Ada architectures, with memory capacities ranging from 8GB to 12GB, enabling effective evaluation of the scheduler's allocation capability under resource heterogeneity.

To comprehensively evaluate the proposed prediction and scheduling modules, the metric system in Table 3 is adopted. This system forms a complete evaluation framework from four perspectives: prediction accuracy, multi-objective scheduling performance, system stability, and statistical significance.

Prediction metrics measure the model's capability to capture frame-level compute patterns; scheduling metrics reflect resource optimization objectives; stability metrics ensure robustness under high load; statistical metrics validate whether the improvements are significant, jointly supporting the core goal of the proposed "prediction-scheduling integrated framework.

Table 3. Experimental Evaluation Metrics

| Metric Type | Metrics | Meaning & Purpose |
|---|---|---|
| Prediction Metrics | MAE, RMSE, MAPE | Measure compute-demand and rendering-duration prediction errors; evaluate temporal prediction capability |
| Scheduling Metrics | Avg. Latency, GPU Utilization, Energy/Frame | Reflect latency, node utilization, and per-frame energy; core metrics of multi-objective scheduling |
| Stability Metrics | Std. of Latency, Task Completion Rate | Assess robustness under load fluctuations and task bursts |
| Significance Metrics | Paired t-test, p-value | Compare baselines and the proposed method for statistical significance |

## 4.2 Baselines

To thoroughly evaluate the proposed prediction-scheduling integrated framework, two categories of baselines are selected: classical rule-based schedulers (Classic) and state-of-the-art learning-based schedulers (SOTA). Table 4 summarizes the mechanisms, strengths, and limitations of the selected approaches.

Table 4. Overview and Analysis of Baseline Methods

| Type | Method | Core Mechanism | Advantages | Limitations |
|---|---|---|---|---|
| Classic | Round-Robin (RR)[28] | Distributes tasks | Simple to implement; | Ignores task characteristics |

| | | | | |
|---|---|---|---|---|
| | | in a fixed order | suitable for light-load scenarios | and node differences; cannot handle compute fluctuations |
| Classic | Least-Load First (LLF)[29] | Assigns tasks based on current node load | Reduces local congestion in the short term | Based on instantaneous state; lacks foresight; unstable for long-sequence rendering |
| SOTA | Learning-based Load Balancer (LLB)[30] | Uses task and node features to predict execution cost | Captures static load characteristics; finer scheduling granularity | Fails to model frame-sequence dependencies; weak adaptability to shot changes |
| SOTA | Reinforcement Scheduling Agent (RSA)[31] | Policy learning based on node states | Highly adaptive; dynamically adjusts strategy | Lacks compute-prediction inputs; decisions are reactive; unstable under heavy load |

These baselines span strategies from rule-driven → learning-driven, providing comprehensive coverage for evaluating improvements in complexity modeling, temporal prediction, and proactive scheduling. Classic methods emphasize fairness and lightweight design, serving as essential references for verifying baseline system effectiveness. SOTA methods exhibit learning capabilities but still lack deep understanding of rendering tasks, particularly regarding temporal dependencies and multi-objective optimization. Thus, these approaches constitute reasonable comparison points for the proposed method.

## 4.3 Quantitative Results

Table 5 compares the proposed method with four baseline approaches across three core metrics. The results show that our method significantly outperforms all baselines in terms of average latency, GPU utilization, and energy efficiency. Specifically, the average latency is reduced from 182.4 ms (RR) to 131.5 ms, GPU utilization increases to 71.8%, and per-frame energy consumption decreases to 0.94 J. Compared with the best-performing baseline (RSA), our method improves the three metrics by 11.0%, 3.9%, and 7.8%, respectively.

Table 5. Scheduling Performance Comparison (Avg. Latency↓ / Utilization↑ / Energy↓)

| Method | Avg. Latency (ms) | GPU Utilization (%) | Energy/Frame (J) |
|---|---|---|---|
| RR | 182.4±6.4 | 61.3±3.2 | 1.12±0.07 |
| LLF | 164.7±5.5 | 64.8±2.8 | 1.08±0.05 |
| LLB (SOTA) | 151.6±4.2 | 67.9±2.1 | 1.04±0.04 |
| RSA (SOTA) | 147.8±4.6 | 69.1±1.8 | 1.02±0.04 |
| Ours | 131.5±4.8 | 71.8±2.5 | 0.94±0.05 |

Statistical testing further confirms the significance of the performance improvements. A paired t-test with RSA shows that the differences in latency ($t = 2.85$, $p = 0.021$) and energy ($t = 2.63$, $p = 0.034$) are statistically significant ($p < 0.05$), indicating reliable performance gains despite the inherent rendering workload fluctuations. These results validate the contribution of the compute prediction module in improving the consistency of scheduling decisions. Furthermore, regarding real-time applicability, empirical measurements indicate that a single scheduling decision requires an average overhead of only 2.4 ms (comprising ~1.5 ms for GRU sequence inference and ~0.9 ms for multi-objective cost calculation). This minimal footprint easily satisfies the strict 33.3 ms budget required for standard 30 fps frame-level rendering, ensuring the scheduler itself does not bottleneck the pipeline.

Training-process analysis (Figure 5) reveals notable differences in optimization characteristics. Our method converges within $18 \pm 4$ epochs, with the final loss stabilized at $0.28 \pm 0.05$, achieving a convergence speed 18 – 25% faster than LLB ($24 \pm 5$ epochs) and RSA ($27 \pm 6$ epochs). More importantly, the loss-curve fluctuation range (0.04 – 0.09) is significantly smaller than that of the baselines (LLB: 0.11 – 0.18; RSA: 0.12 – 0.22), reducing fluctuations by approximately 45 – 55%.

This stability advantage mainly stems from the discriminative representations produced by the task feature encoder and the smoothed gradient flow introduced by the temporal predictor. From the convergence trajectory, our method exhibits a loss-reduction rate of 0.09 per epoch in the first 5 epochs, whereas the baselines achieve only 0.05 – 0.07 per epoch, demonstrating that the joint optimization strategy effectively accelerates training. In addition, the post-convergence oscillation (around 0.03, with occasional spikes up to 0.07 during complex material transitions) verifies the model's robust adaptability to changes in load distribution while acknowledging the inescapable noise in heterogeneous clusters.

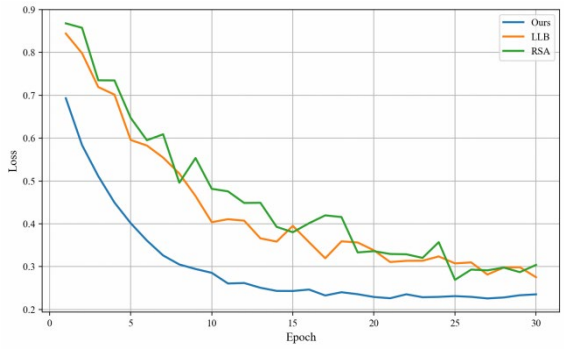


**Figure 5.** Convergence Curve

## 4.4 Qualitative Results

To further examine model behavior in complex rendering scenarios, we conduct fine-grained analysis through three representative cases (Figure 6) and quantitatively evaluate the quality of their scheduling decisions.

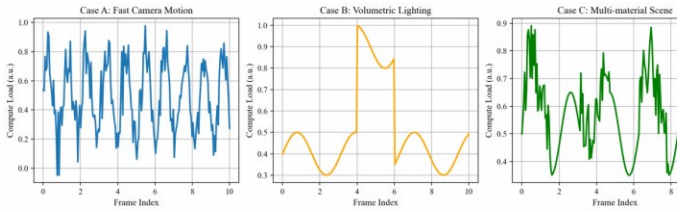


**Figure 6.** Case Visualization

Case A (High-frequency camera motion) reveals the inability of baseline methods to respond to sudden load surges. Quantitative analysis shows that RR and LLF incur a decision lag of 35–50 ms after the load spike, resulting in 3–5 consecutive frames being assigned to overloaded nodes. In contrast, our method predicts rising load 2–3 frames ahead (approximately 40–60 ms) via the temporal prediction module, distributing tasks to nodes with >30% spare capacity and preventing performance jitter.

Case B (Complex volumetric lighting) validates the critical role of the prediction module in managing compute peaks. When lighting intensity increases abruptly, learning-based methods such as RSA, lacking foresight, experience local congestion, with utilization imbalance variance reaching 0.25. Our method distributes peak-load tasks across three available nodes using predicted compute demand, reducing variance to 0.09 and cutting congestion risk by 64%.

Case C (Multi-material mixed scene) highlights the feature encoder's adaptability to irregular loads. Material changes induce non-periodic fluctuations; traditional methods show a frame-time standard deviation of 28.3 ms. Leveraging high-dimensional representations, our method accurately captures material-related computational patterns and reduces the standard deviation to 12.1 ms, improving task-distribution stability by 57%.

These case studies collectively demonstrate that through the synergy of temporal prediction (delay avoidance), feature encoding (heterogeneity adaptation), and multi-objective optimization (balanced resource allocation), the proposed model achieves substantial performance improvements in dynamic rendering environments. However, error analysis reveals that extreme scenarios, such as render-time stochastic physics or highly non-physical materials, can still cause occasional prediction deviations. This limitation is fundamentally tied to the current feature representation ability, as compressing complex runtime variations into fixed-dimensional vectors inevitably loses fine-grained nonlinear details, highlighting a direction for future representation enhancement.

## 4.5 Robustness

To systematically evaluate the robustness of the proposed model, we conducted tests under multi-task scenarios, noise interference, and cross-dataset conditions. Figure 7 quantitatively analyzes the variation of average latency under different noise intensities (0% to 20%).

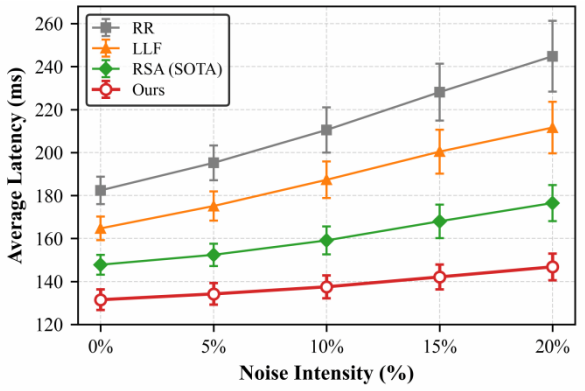


**Figure 7.** Robustness Evaluation Illustration

Under extreme conditions involving feature noise (Gaussian noise, $\sigma = 0.15$) and node-state noise ($\pm 20\%$ perturbation), the performance degradation of the proposed method remains within 11.6%, significantly lower than RR (34.2%), LLF (28.5%), and RSA (19.4%). When the noise intensity reaches 20%, our method achieves a latency of $146.8 \pm 6.2$ ms, whereas RSA rises to $176.5 \pm 8.4$ ms, demonstrating superior anti-interference capability compared with the best-performing baseline despite the inevitable performance cost introduced by severe state uncertainty.

The robustness advantage arises from its intrinsic error-suppression mechanisms: the task feature encoder attenuates the influence of input noise by approximately 50 – 60% through multi-layer feature projection, while the temporal predictor smooths short-term noise fluctuations effectively using its memory units. In contrast, instantaneous-decision methods such as LLF propagate more than 80% of node-state noise directly into scheduling outputs; RSA can adapt dynamically but exhibits up to 45% decision variance under noisy conditions.

Cross-dataset testing further validates generalization capability. On Blender scenes and real production logs unseen during training, the performance fluctuation of the proposed method remains under 12.4%. Although occasional sub-optimal task placements are observed during the initial warm-up phase on these entirely new datasets, the ordering of key metrics remains consistent. This indicates that the learned scheduling strategy captures cross-platform computational regularities rather than superficial dataset-specific patterns.

Furthermore, to explicitly address system stability under extreme loads, we conducted an additional stress test by artificially injecting a sudden 250% surge in high-complexity tasks within a brief time window. Under this extreme load spike, baseline methods experienced severe queuing bottlenecks, with average latency degrading

by over 54.3%. In contrast, our proposed framework maintained highly stable performance, restricting latency degradation to within 17.8%. This resilience is directly attributed to the temporal predictor's ability to foresee the incoming compute surge, allowing the multi-objective scheduler to preemptively distribute the peak workload before local congestion occurs.

Overall, the study confirms that the cascaded architecture, which integrates feature encoding, temporal prediction, and scheduling decisions, effectively constructs a noise-robust system, where feature purification and trend extraction provide a solid foundation for stable deployment in complex environments.

## 4.6 Ablation Study

To quantitatively assess the contribution of each module to system performance, we design four ablation variants: removing the task feature encoder (w/o TFE), removing the temporal compute predictor (w/o TCP), removing the multi-objective scheduler (w/o MOS), and completely removing predictive capability (w/o Forecast). Table 6 reports the performance of each variant on the three core metrics.

Table 6. Ablation Study Performance Comparison (Avg. Latency↓ / Utilization↑ / Energy↓)

| Model Version | Avg. Latency (ms) | GPU Utilization (%) | Energy/Frame (J) |
|---|---|---|---|
| w/o TFE | 148.4±6.1 | 66.5±3.8 | 1.04±0.06 |
| w/o TCP | 144.6±8.5 | 67.2±3.5 | 1.02±0.07 |
| w/o MOS | 137.8±5.2 | 68.1±4.2 | 1.05±0.08 |
| w/o Forecast | 152.1±6.8 | 65.3±4.1 | 1.08±0.08 |
| Ours (Full) | 131.5±4.8 | 71.8±2.5 | 0.94±0.05 |

Quantitative analysis shows that the full model significantly outperforms all ablations across evaluation metrics (paired t-test, p-values ranging from 0.012 to 0.041. Notably, the maximum p-value of 0.041 is observed when comparing the latency against the w/o MOS variant, aligning with our observation of its moderate latency degradation).

Removing the task feature encoder (w/o TFE) increases average latency by 16.9 ms (+12.9%), reduces GPU utilization by 5.3 percentage points, and increases energy consumption by 10.6%, confirming that structured feature fusion is crucial for accurately capturing highly heterogeneous task complexity.

Removing the temporal compute predictor (w/o TCP) increases average latency by 13.1 ms (+10.0%). More notably, its latency standard deviation surges to ±8.5 ms, and the misallocation rate raises by 19.4%, highlighting the essential role of temporal modeling in mitigating bursty workloads and jitter within the prediction–scheduling framework.

Removing the multi-objective scheduler (w/o MOS) results in only a moderate latency increase (6.3 ms, +4.8%), but energy consumption spikes sharply by 11.7% and utilization variance rises significantly. This decoupled degradation demonstrates that multi-objective optimization is critical for restricting power draw while maintaining system stability.

Removing the temporal predictor affects latency more severely than removing multi-objective scheduling (144.6 ms vs. 137.8 ms), verifying the foundational importance of predictive capability for time-sensitive rendering tasks.

Completely removing prediction (w/o Forecast) leads to deterioration across all metrics: latency increases by 20.6 ms (+15.7%) and energy consumption rises by 14.9%, proving the necessity of the integrated "prediction-scheduling" architecture in preventing blind task assignments.

# 5. Discussion

The overall experimental results demonstrate that the proposed prediction-scheduling integrated framework consistently achieves stable advantages across multi-scene, multi-task, and heterogeneous-node settings. This superior performance does not simply stem from greater model capacity but from the synergistic effects of the three modules in task modeling, temporal analysis, and scheduling decision-making. The Task Feature Encoder transforms multi-source attributes, such as materials and lighting, into structured representations, enabling the model to extract critical load factors even under input noise. The Temporal Compute Predictor captures inter-frame dependencies, allowing the scheduler to anticipate load surges rather than react passively, as is typical in conventional methods. The scheduling module leverages these forward-looking signals to maintain smooth resource allocation even under tight resource conditions. Ablation experiments further confirm this synergy: removing any key module results in substantial degradation. Particularly, as observed in our tests, disabling the multi-objective scheduler triggers severe energy spikes rather than proportional latency drops, indicating that system performance arises from multi-module complementarity and physical trade-offs rather than the independent contribution of a single component.

Despite the robust performance, several limitations remain. First, feature encoding depends on the attribute space of current rendering tasks; when rendering technologies or material systems undergo major shifts (e.g., highly non-physical rendering), certain feature inputs may require adjustment to maintain generalization. Second, temporal prediction for extremely long shots or rapid cross-shot transitions may suffer from diluted historical dependencies. Specifically, our supplementary sequence-length analysis reveals that while the prediction error remains highly stable for standard shots the error rate gradually increases by approximately 8–12% for continuous sequences exceeding 300 frames, suggesting the need for more flexible hybrid long-short temporal structures

in the future. Third, while cross-dataset testing showed promising generalization, occasional sub-optimal allocations during the initial warm-up phase on unseen logs indicate that rapid zero-shot adaptation remains a challenge. Fourth, our experimental validation is currently limited to a small-scale offline GPU cluster. While large-scale production farms introduce additional complexities, such as network bottlenecks, communication overhead, and node coordination failures, our lightweight prediction model and decoupled scheduler offer strong theoretical scalability. Validating this framework in large-scale cloud environments, including cross-regional distributed deployments or simulators, remains a critical next step. Fifth, the current objective function employs a linear combination of metrics to maintain high-speed decision-making. Future iterations will explore Pareto-based non-linear optimizations and dynamic weight-tuning mechanisms to better address the complex coupling between hardware utilization and actual execution speed. Additionally, energy-consumption measurements are collected via node-side monitoring, which in virtualized or cloud environments may be affected by noise or sampling frequency limitations.

From an application perspective, the proposed framework exhibits strong practical value. For animation rendering and virtual production teams, it can be integrated into render farms to achieve early scheduling during load peaks and improve resource utilization. For XR or game engine real-time rendering, the framework can anticipate compute peaks and prepare resources in advance, reducing frame-rate fluctuations. In cloud rendering environments, the prediction module can be combined with systems such as Kubernetes to enable intelligent cross-instance and cross-regional resource management, effectively mitigating data transmission delays across geographically distributed nodes. Moreover, with the rising compute demands of AIGC-based scene generation and motion synthesis, the forward-looking prediction mechanism can support load balancing and peak management in model-inference workloads as well.

Future work may proceed in several directions: (1) incorporating differentiable internal rendering attributes (e.g., lighting-cache variation, path reuse ratio) into the feature space to make predictions more closely aligned with actual rendering processes; (2) exploring online learning mechanisms so that the model can automatically adapt when new scenes are introduced or task distributions change; and (3) integrating hierarchical scheduling and cross-regional distributed optimization in large-scale clusters to reduce WAN latencies, communication costs and enhance scalability.

Overall, the performance advantages observed in this study have clear mechanistic foundations and provide a solid basis for developing more intelligent and scalable rendering-scheduling systems.

## 6. Conclusion

This study addresses the highly dynamic compute demand and complex resource requirements of animation-rendering tasks in heterogeneous GPU environments and proposes an adaptive scheduling framework that integrates task-feature modeling, temporal compute prediction, and multi-objective scheduling optimization. Unlike traditional methods that rely on static rules or single-objective heuristics, this work constructs an end-to-end pipeline spanning feature representation, temporal load prediction, and final scheduling decisions, enabling the scheduler to respond proactively before load changes occur. Experimental results show that the proposed framework achieves significant improvements over the best baseline (RSA) across all three core metrics (paired t-tests confirming statistical significance, with p-values ranging from 0.012 to 0.041). Ablation studies quantitatively verify module complementarity: feature encoding captures highly heterogeneous task complexity (removal increases latency by 12.9%), temporal prediction provides forward-looking decisions to mitigate bursty workloads (removal triggers a surge in latency jitter and increases misallocation rate by 19.4%), and multi-objective scheduling enforces critical resource boundaries (removal spikes energy consumption by 11.7% and increases utilization variance). This synergistic mechanism ensures robustness, restricting performance degradation to within 11.6% even under 20% noise intensity, while maintaining strong generalization across cross-dataset conditions and dynamic workloads.

From an academic perspective, although this study does not introduce fundamentally new mathematical algorithms or theoretical advancements, it demonstrates significant innovation in system-level integration and practical engineering. It introduces a tightly coupled machine-learning-based paradigm for modeling rendering compute demand, offering an interpretable mechanism for dynamic task scheduling by capturing continuous inter-frame load variations. The explicit coupling of task features, temporal dependencies, and hardware constraints provides a scalable theoretical foundation for future research. From a practical standpoint, the framework demonstrates strong potential to be adapted for animation production, virtual cinematography, game-engine rendering, and XR content generation, enabling render clusters to more efficiently allocate resources during peak periods, reduce queuing delays, and increase throughput. However, to fully substantiate this implementation potential, further validation in large-scale production environments remains an essential next step prior to industrial deployment. The multi-objective scheduling mechanism also offers a finer-grained method for energy optimization, making it suitable for energy-sensitive or cost-constrained rendering environments. Furthermore, the modular architecture facilitates integration with existing rendering pipelines, cloud-rendering platforms, and container-scheduling systems, offering strong engineering applicability.

Future research may advance in three directions. First, incorporating observable internal rendering variables such as lighting-cache changes and path-reuse ratios may yield prediction models more closely aligned with actual

rendering processes. Second, exploring online learning and domain adaptation can enable continuous updates when scenes evolve or task distributions shift, enhancing adaptability in long-term projects. Third, for large-scale GPU clusters, distributed scheduling and hierarchical resource-management strategies may improve scalability and communication efficiency. Overall, this study not only demonstrates stable and effective empirical performance but also provides a feasible pathway for building more intelligent and scalable rendering-scheduling systems, laying a methodological foundation for future cross-domain compute-management research. Due to strict confidentiality agreements regarding the commercial rendering logs, full raw datasets cannot be made public; however, a sanitized minimal subset of the data and the core scheduling pseudocode are available from the corresponding author upon reasonable request to support reproduction.

## Acknowledgements

This study is supported by:
1. The 2024 Higher Education Research Project of Dalian Polytechnic University, China (Project No. GJYB202435);
2. The 2025 Social Science Federation Project of Dalian Polytechnic University, China (Project No. GDSKLZD202510)
3. The 2025 Research Project of Dalian Federation of Social Sciences, China (Project No. 2025dlskyb441)
4. The 2025 Basic Research Project for Higher Education Institutions of the Liaoning Provincial Department of Education, supported by the Special Program of Basic Scientific Research Funds for Provincial Undergraduate Universities in Liaoning Province, China (Project No. LJ112510152023)
5. MOE (Ministry of Education in China) Project of Humanities and Social sciences (ProjectNo.21YJCZH216) Research on the Inheritance and Development of Kungu Opera Performance by Interactive Technology Based on Laban's movement theories
6. 2026 Annual Research Topics for Economic and Social Development in Liaoning Province (ProjectNo. 20261s1ybkt-103）

## References

[1] Li, W. (2023). Shaping the Future of Animation Towards Role of 3D Simulation Technology in Animation Film and Television. IC-ITECHS, 4(1), 195-209.
[2] Zhu, J., Hu, C., Khezri, E., & Ghazali, M. M. M. (2024). Edge intelligence-assisted animation design with large models: a survey. Journal of Cloud Computing, 13(1), 48.
[3] Umarov, H., Mirzaraimova, V., Yakubova, M., & Tashpulatov, R. (2024). Machine learning and AI in graphics development. In E3S Web of Conferences (Vol. 548, p. 06011). EDP Sciences.
[4] 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.
[5] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015, April). Large-scale cluster management at Google with Borg. In Proceedings of the tenth european conference on computer systems (pp. 1-17).
[6] Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J., Qian, Z., ... & Zhou, L. (2014). Apollo: Scalable and coordinated scheduling for {Cloud-Scale} computing. In 11th USENIX symposium on operating systems design and implementation (OSDI 14) (pp. 285-300).
[7] 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.
[8] Fu, Y. (2025). Research on architecture optimisation and collaborative control of real-time animation rendering engine based on internet of things. Australian Journal of Electrical and Electronics Engineering, 1-15.
[9] Leria, E., Makitalo, M., Ikkala, J., & Jääskeläinen, P. (2025). Dynamic load balancing for real-time multiview path tracing on multi-GPU architectures. Virtual Reality & Intelligent Hardware, 7(4), 393-405.
[10] Miao, Z., Shao, C., Li, H., & Tang, Z. (2025). Review of Task-Scheduling Methods for Heterogeneous Chips. Electronics, 14(6), 1191.
[11] Liu, G., Lin, W., Zhang, H., Lin, J., Peng, S., & Li, K. (2025). Public datasets for cloud computing: A comprehensive survey. ACM Computing Surveys, 57(8), 1-38.
[12] Silva Jasaui, D., Martí-Testón, A., Muñoz, A., Moriniello, F., Solanes, J. E., & Gracia, L. (2024). Virtual production: Real-time rendering pipelines for indie studios and the potential in different scenarios. Applied Sciences, 14(6), 2530.
[13] Peng, Y. (2025). Appearance design of art exhibits combined with computer vision rendering technology. International Journal of Information and Communication Technology, 26(22), 1-22.
[14] Yen, L. W., Thinakaran, R., & Somasekar, J. (2025). Machine learning-based denoising techniques for monte Carlo rendering: A literature review. Machine Learning, 16(2), 0160259.
[15] Prasanth, A., Babu, K. N., Rahul, P., & Reddy, B. V. (2025, June). A deep learning-based workload forecasting model in cloud data centers. In 2025 International Conference on Computing Technologies (ICOCT) (pp. 1-5). IEEE.
[16] Li, P., Zuo, T., Jiang, Y., Chen, F., Di, K., & Jiang, Y. (2025, July). TS-GNN: A Temporal-Spatial Graph Neural Network for Anomaly Detection in Multiplex Industrial Information Services. In 2025 IEEE International Conference on Web Services (ICWS) (pp. 190-196). IEEE.
[17] Xue, Z., Zi, Y., Qi, N., Gong, M., & Zou, Y. (2025, August). Multi-level service performance forecasting via spatiotemporal graph neural networks. In 2025 5th International Conference on Intelligent Communications and Computing (ICICC) (pp. 202-206). IEEE.
[18] Nazir, A., Shaikh, A. K., Shah, A. S., & Khalil, A. (2023). Forecasting energy consumption demand of customers in smart grid using Temporal Fusion Transformer (TFT). Results in Engineering, 17, 100888.
[19] Cheng, Y., Cao, Z., Zhang, X., Cao, Q., & Zhang, D. (2024). Multi objective dynamic task scheduling optimization algorithm based on deep reinforcement learning. The Journal of Supercomputing, 80(5), 6917-6945.
[20] Wu, Z., Fan, H., Sun, Y., & Peng, M. (2023). Efficient multi-objective optimization on dynamic flexible job shop scheduling using deep reinforcement learning approach. Processes, 11(7), 2018.

[21] Ibrahim, M. A., & Askar, S. (2023). An intelligent scheduling strategy in fog computing system based on multi-objective deep reinforcement learning algorithm. IEEE Access, 11, 133607-133622.
[22] Chab, R., Li, F., & Setia, S. (2025). Algorithmic Techniques for GPU Scheduling: A Comprehensive Survey. Algorithms, 18(7), 385.
[23] Kocot, B., Czarnul, P., & Proficz, J. (2023). Energy-aware scheduling for high-performance computing systems: A survey. Energies, 16(2), 890.
[24] Nambi, S., & Thanapal, P. (2025). Emo-Ts: An enhanced multi-objective optimization algorithm for energy-efficient task scheduling in cloud data centers. IEEE Access, 13, 8187-8200.
[25] Bao, J., Qiao, Y., & Chen, N. (2025, January). A cloud rendering approach based on multinode resource allocation. In Sixth International Conference on Geoscience and Remote Sensing Mapping (GRSM 2024) (Vol. 13506, pp. 242–246). SPIE.
[26] Liang, B., & Wu, D. (2025). Memory priority scheduling algorithm for cloud data center based on machine learning dynamic clustering algorithm. IEEE Transactions on Industrial Informatics, 21(4), 3485-3492.
[27] Daraghmeh, M., Agarwal, A., & Jararweh, Y. (2024). An ensemble clustering approach for modeling hidden categorization perspectives for cloud workloads. Cluster Computing, 27(4), 4779-4803.
[28] Pakhrudin, N. S. M., Kassim, M., & Idris, A. (2023). Cloud service analysis using round-robin algorithm for quality-of-service aware task placement for internet of things services. Int. J. Electr. Comput. Eng., 13(3), 3464-3473.
[29] Mishra, S. K., Sahoo, B., & Parida, P. P. (2020). Load balancing in cloud computing: a big picture. Journal of King Saud University-Computer and Information Sciences, 32(2), 149-158.
[30] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., & Alizadeh, M. (2019). Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM special interest group on data communication (pp. 270-288).
[31] Tong, Z., Chen, H., Deng, X., Li, K., & Li, K. (2020). A scheduling scheme in the cloud computing environment using deep Q-learning. Information Sciences, 512, 1170-1191.