JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 30, No. 3: 453~460, Mar. 2026

LLM 기반 GPU 클러스터 스케줄러

황훈순[1] · 김명선[2*]

LLM-based GPU Cluster Scheduler

Hunsun Hwang[1] · Myungsun Kim[2*]

[1]Graduate Student, Department of Applied AI, Hansung University, Seoul 02876, Korea
[2*]Associate Professor, Department of Applied AI, Hansung University, Seoul 02876, Korea

요 약

기존 First-In First-Out(FIFO), Shortest Job First(SJF) 등의 규칙 기반 GPU 스케줄러는 전문가의 도메인 지식에 의존하며, 복잡한 작업 구성이나 비정형적 시스템 환경 변화에 유연하게 대응하기 어렵다. 본 연구에서는 이러한 한계를 극복하기 위해, 명시적 규칙 설계 없이 스케줄링 데이터로부터 정책을 학습하는 Large Language Model(LLM) 기반 GPU 클러스터 스케줄러를 제안한다. 제안 방법은 LlamaRec의 LLMRanker(Llama-2-7B, LoRA 기반 PEFT)를 Job Sorter로 활용하며, 알리바바 GPU 작업 트레이스 기반 시뮬레이터에서 생성된 SJF 정책 데이터로 학습한다. 이 접근법은 규칙을 재설계할 필요 없이 데이터 추가 및 교체를 통한 재학습만으로 환경 변화에 대응할 수 있으며, 데이터 품질과 양에 따라 성능 향상도 기대할 수 있다. 실험 결과, LLMRanker는 Normalized Discounted Cumulative Gain(NDCG) 및 Kendall's tau 지표에서 높은 순위 예측 성능을 보였으며, GPU 클러스터 시뮬레이션에서 Job Completion Time(JCT), 대기 시간, makespan, GPU 활용률 모두 SJF와 동등한 수준을 달성하였다.

ABSTRACT

Rule-based GPU schedulers such as First-In First-Out (FIFO) and Shortest Job First (SJF) rely on expert domain knowledge and struggle to adapt to complex job configurations or dynamic environments. To overcome these limitations, this study proposes a Large Language Model (LLM)-based GPU cluster scheduler that learns scheduling policies from data without explicit rule design. The proposed method employs LLMRanker from LlamaRec (Llama-2-7B with LoRA-based PEFT) as a job sorter, trained on SJF policy data from an Alibaba GPU job trace-based simulator. This approach adapts to environmental changes through retraining with additional or updated data, eliminating the need for rule redesign, with further performance gains expected as data quality and quantity improve. Experimental results show that LLMRanker achieves high ranking accuracy in Normalized Discounted Cumulative Gain (NDCG) and Kendall's tau, and demonstrates performance comparable to SJF in GPU cluster simulations across Job Completion Time (JCT), waiting time, makespan, and GPU utilization.

키워드 : GPU 클러스터 스케줄링, 대형 언어 모델, 작업 순위화, 규칙 기반 스케줄링
Keywords : GPU Cluster Scheduling, Large Language Model (LLM), Job Ranking, Rule-based Scheduling

Received 25 December 2026, **Revised** 22 January 2026, **Accepted** 27 February 2026

*** Corresponding Author** Myungsun Kim (E-mail : kmsjames@hansung.ac.kr Tel +82-02 −760-4045)
Department of Applied AI, Hansung University, Seoul 02876, Korea

Open Access **http://doi.org/10.6109/jkiice.2026.30.3.453** pISSN:2234-4772

Ⅰ. 서 론

GPU 클러스터 환경에서 효율적인 작업 스케줄링은 시스템 성능과 자원 활용률을 결정하는 핵심 요소이다[1,2]. 기존의 First-In, First Out(FIFO), Shortest Job First(SJF), Priority-based 등과 같은 규칙 기반 정책을 활용한 시스템 자원 스케줄링 방식은 각각의 시스템 특징들에 최적화된 방법으로 동작한다[3-5]. 즉, 오프라인에서 전문가 집단에 의해 시스템 각각의 특성 및 요구사항들을 분석하고 이에 가장 적절한 정책을 각각의 시스템에 적용한다. 이러한 방식은 설계 과정에서 전문가의 도메인 지식과 경험에 크게 의존하며, 고정된 규칙에 따라 동작하기 때문에 복잡한 작업 구성이나 장애 패턴, 비정형적인 시스템 상황에서 세밀한 조정이 어렵다는 한계가 있다[6,7].

대형 언어 모델(LLM)은 복잡하고 비정형적인 정보를 구조화하고 예측하는 능력을 갖추고 있어, 이를 시스템 스케줄링에 적용하면 기존 규칙 기반 정책의 한계를 극복할 수 있는 잠재력을 가진다[8-11]. LLM 기반 스케줄러는 과거 실행 데이터를 학습하여 다양한 후보 작업 구성과 워크로드 환경에서도 일반화된 스케줄링 결정을 수행할 수 있다[12,13]. 또한, 전문가가 직접 정의하기 어려운 복잡한 규칙이나 장애 패턴, 성능 저하 상황을 자연어 기반 지침으로 제어할 수 있어, 시스템마다 서로 다른 고정 규칙에 의존하는 기존 방식보다 정교한 스케줄링이 가능하다.

본 연구에서는 LlamaRec에서 제안된 LLMRanker를 활용하여 SJF 정책을 모방하도록 파인튜닝하고, 이를 기반으로 하는 LLM 기반 스케줄러를 제안한다[14]. 제안하는 스케줄러는 환경이나 정책이 변화하더라도 규칙을 재설계할 필요 없이 데이터 추가 및 교체를 통한 재학습만으로 대응할 수 있으며, 데이터의 품질과 양에 따라 스케줄링 성능의 추가적인 향상도 기대할 수 있다. 성능은 전통적인 규칙 기반 스케줄러(FIFO, SJF)와 비교 평가되며, Job Completion Time(JCT), 대기 시간, makespan, GPU 활용률 등의 지표를 통해 LLM 기반 접근법이 규칙 기반 스케줄러의 동작을 효과적으로 재현할 수 있음을 확인한다.

Ⅱ. 연구 배경

2.1 규칙 기반 GPU 작업 스케줄러

대부분의 GPU 작업 스케줄러는 제한된 GPU 자원을 효율적으로 활용하기 위해 규칙 기반(rule-based) 접근법을 사용하여 작업의 실행 순서를 결정한다[1,4]. 이러한 접근법은 FIFO(First-In First-Out), SJF (Shortest Job First), Priority-based와 같은 규칙 기반 정책으로 작업의 우선순위를 판단하는 방식으로, 설계가 단순하고 해석이 용이하다는 장점을 가진다. 반면, 정책 설계 과정에서 도메인 지식과 수작업 튜닝에 대한 의존도가 높으며, 워크로드 특성이나 시스템 상태 변화에 따라 유연하게 대응하기 어렵다는 한계를 갖는다[6,7].

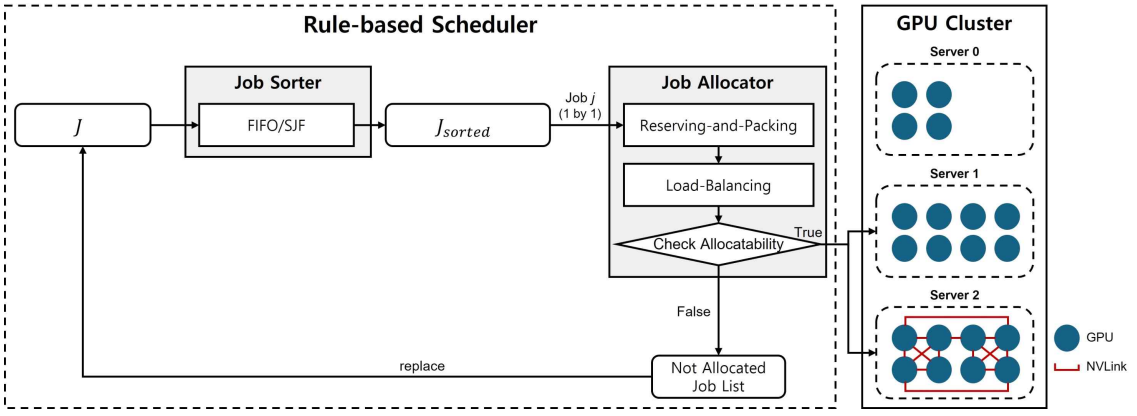
Alibaba의 대규모 생산 환경 GPU 클러스터에서도 이러한 규칙 기반 스케줄링 방식이 적용된다[15,16]. 그림 1은 Alibaba GPU cluster trace(v2020)에 공개된 시스템을 기반으로 한 규칙 기반 GPU 작업 스케줄러의 구조를 나타낸다. 해당 스케줄러는 머신러닝 학습 및 추론 작업을 대상으로 하며, FIFO 및 SJF와 같은 규칙 기반 정책으로 작업에 GPU 자원을 할당한다[15].

본 스케줄러는 크게 Job Sorter와 Job Allocator의 두 구성요소로 이루어진다. Job Sorter는 대기 작업 리스트를 입력으로 받아 사전에 정의된 규칙 기반 정책에 따라 작업의 우선순위를 결정하고, 정렬된 대기 작업 리스트를 출력한다. 이후 Job Allocator는 정렬된 작업을 순차적으로 처리하며 GPU 자원 할당을 시도한다. 이 과정에서 Reserving-and-Packing과 Load-Balancing 정책이 적용된다. Reserving-and-Packing은 GPU 성능 특성을 고려하여 작업의 요구 자원 수준에 따라 적절한 GPU에 우선 배치함으로써 자원 활용률과 대기 지연을 동시에 완화하는 것을 목표로 한다. Load-Balancing은 서버별 자원 사용률을 기준으로 작업을 분산 배치하여 노드 간 부하 불균형으로 인한 성능 저하를 완화한다. 자원 할당이 불가능한 작업은 이후 스케줄링 주기에서 다시 고려된다.

이와 같이 Alibaba의 GPU 작업 스케줄러는 전통적인 규칙 기반 스케줄링 방식을 대규모 GPU 클러스터 환경에 적용한 대표적인 사례로 볼 수 있으며, 규칙 기반 정책이 갖는 일반적인 한계 또한 동일하게 가진다.

2.2 LlamaRec

LlamaRec는 LLM 기반 추천에서 효율적이고 확장



가능한 순위화를 수행하기 위해 제안된 두 단계 추천 프레임워크로[14], LRURec[17]와 LLMRanker로 구성된다.

첫 번째 단계에서는 사용자 상호작용 히스토리를 입력으로 받아 LRURec 기반의 경량 순차 추천기를 사용하여 후보 아이템을 추출한다. LRURec는 선형 recurrent unit을 사용해 self-attention 기반 모델 대비 훨씬 빠른 추론 속도와 낮은 계산 비용을 제공하며, 긴 시퀀스에서도 점진적(incremental) 추론이 가능하다. 이러한 구조적 장점 덕분에 LRURec는 매우 큰 아이템 풀(예: 10,000개 이상)을 전체 계산하지 않고도, 사용자 히스토리에 대한 점수(score)를 빠르게 산출한 뒤 상위 Top-K 아이템만을 후보로 선택하는 효율적인 candidate retrieval을 수행한다. 즉, LRURec는 전체 아이템 분포에서 사용자가 클릭할 가능성이 높은 소수의 후보만을 선별해 LLMRanker로 전달함으로써, 전체 시스템의 추론 비용을 크게 절감한다.

두 번째 단계에서는 LLMRanker가 후보 아이템에 대한 랭킹을 수행한다. LLMRanker는 LLM과 verbalizer로 구성되며, LLM은 Llama-2 7B를 기반으로 하였다. LLM은 사용자 히스토리와 후보 아이템 목록, 추천 지침을 포함한 프롬프트를 입력으로 받는다. 이때 각 후보 아이템은 프롬프트 생성 과정에서 영문 알파벳 'A'부터 시작하여 순서대로 인덱스 문자(index letter)가 매핑된다. 즉, 첫 번째 후보는 'A', 두 번째 후보는 'B', 세 번째 후보는 'C'와 같이 순차적으로 대응된다. LLM은 프롬프트를 입력으로 받아 각 후보에 대한 logits을 출력하고, verbalizer는 logits에서 후보 아이템에 매핑된 인덱스 문자의 logit만을 추출하여 후보 집합에 대한 logits로 변환한다. 이러한 접근 방식은 AR(Auto-Regressive) 기반 순위화와 달리, 단일 forward pass로 전체 후보를 랭킹하여 추론 효율성을 크게 향상시킨다.

Ⅲ. LLM 기반 GPU 클러스터 스케줄링

기존 규칙 기반 스케줄러의 Job Sorter는 전문가가 직접 설계한 고정된 규칙에 따라 작업 우선순위를 결정하므로, 복잡한 작업 구성이나 비정형적 환경 변화에 유연하게 대응하기 어렵다. 반면 LLM은 복잡하고 비정형적인 정보를 구조화하고 예측하는 능력을 갖추고 있어 이러한 한계를 극복하기에 적합하므로, 본 연구에서는 LLM 기반 스케줄러를 제안한다. 구체적으로, LlamaRec의 LLMRanker를 Job Sorter로 활용하며, 평균 대기 시간을 최소화하는 SJF 정책을 따르도록 파인튜닝한 뒤 기존 Job Sorter로 대체하여 구현되었다.

3.1 알리바바 스케줄링 데이터 기반 데이터셋 구축

먼저 LLM 학습을 위해 알리바바 GPU 작업 스케줄링 시뮬레이터로 생성한 스케줄링 데이터를 기반으로 데이터셋을 구축하였다. 해당 시뮬레이터는 그림 1의 스케줄러 구조를 기반으로 구현되었다.

스케줄링 데이터 생성을 위해 알리바바에서 제공한 작업 구성 파일의 총 100,000개의 작업 중 50,000개를 선택하여 GPU 워크로드를 구성하였다. 각 작업의 요구 GPU 수, 요구 CPU 수와 실행 시간은 사전에 정의된 범위에서 단일 값으로 샘플링 하였다. 요구 GPU 수의

범위는 0~255, 요구 CPU 수의 범위는 1~4,800, 작업 실행 시간의 범위는 3초~535,585초로 설정하였다. 실행 환경은 단일 서버로 구성된 GPU 클러스터로 총 5000개의 GPU와 116,100개의 CPU로 구성하였다. 구성된 GPU 워크로드와 실험 환경에서 시뮬레이터로 스케줄링을 수행하였으며, 스케줄링 정책은 가장 짧은 작업을 우선적으로 처리하여 평균 대기 시간을 최소화하는 SJF를 사용하였다. 스케줄링 과정에서 작업 할당 이벤트가 발생할 때마다 스냅샷을 저장하여 스케줄링 데이터를 구성하였다. 스냅샷에는 SJF로 정렬된 대기 작업 리스트 J_{sorted}, 정렬되지 않은 대기 작업 리스트 J로 구성되어있다. J는 L개의 대기 작업 $[j_1, ..., j_L]$으로 구성되어 있으며, 여기서 j_i는 i-번째 대기 작업, L은 대기 작업 수를 의미한다.

스냅샷으로 구성된 스케줄링 데이터는 전처리 과정을 통해 각 스냅샷이 프롬프트-레이블 쌍으로 변환된다. 그림 2는 전처리 후 구성되는 프롬프트-레이블 쌍의 예시를 보여준다. 프롬프트는 작업 수행 지침과 J가 작성된 입력으로 구성된다. 레이블은 대기 작업들의 순위를 기반으로 계산된 관련도 점수(relevance score) y_i를 모아 구성된 관련도 점수 벡터(relevance score vector) y로 정의된다. 관련도(relevance) r이란 스케줄러 관점에서 특정 작업이 얼마나 우선적으로 고려되어야 하는지를 나타내는 값으로, 작업의 순위를 수치화한 값을 의미한다. j_i의 관련도 r_i는 순위 $rank_i$로 계산되며, 수식 (1)과 같이 정의된다. $rank_i$는 j_i의 순위로 K_{sorted}에서 j_i의 위치(index)에 1을 더한 값으로 정의된다. y_i는 각 작업의 관련도 r_i를 softmax로 정규화한 값으로, 수식 (2)와 같이 정의되며. 이를 모아 관련도 점수 벡터(relevance score vector) y를 구성한다.

$$r_i = L - rank_i + 1,\ i = 1, ..., L \tag{1}$$

$$y_i = \frac{\exp(r_i)}{\sum_{k}^{L} \exp(r_k)},\ y = [y_1, y_2, ..., y_L] \tag{2}$$

본 데이터셋 구축 방식은 실제 GPU 클러스터에서 관측된 이벤트 흐름과 자원 상태를 그대로 반영하여, 현실적이고 신뢰성 높은 스케줄링 데이터셋을 제공한다. 이를 통해, LLM은 데이터셋의 클러스터 상태 정보와 작업 정보를 기반으로 스케줄링 정책을 학습할 수 있도록 한다.

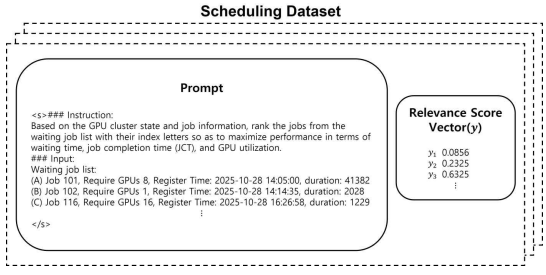


Fig. 2 Example prompt-label pair from the scheduling dataset

3.2 LLMRanker 파인튜닝

LLMRec를 구성하는 모델 중 파인튜닝 대상은 LLMRanker로, 후보 작업 간 상대적 순위를 정확히 학습하는 것을 목표로 한다. 파인튜닝 과정에서는 예측된 후보 작업 점수와 relevance score vector 간의 차이를 최소화하며, 손실 함수는 수식 (3)과 같이 정의된다.

$$\min_{\theta} E_{(x,y)\sim X}[-\sum_{i=1}^{L} y_i \log p_\theta(j_i|x)] \tag{3}$$

x는 프롬프트, y는 레이블, X는 데이터셋, $p_\theta(j_i|x)$는 모델이 예측한 j_i의 점수이다. 이 과정을 통해 LLMRanker는 프롬프트에 작성된 정보를 바탕으로 단일 forward pass만으로도 FIFO 또는 SJF와 같은 규칙 기반 정책의 결정 방식을 모방하여 작업 순서를 예측하도록 파인튜닝 된다.

3.3 LLM 기반 스케줄러 구조 및 동작

그림 3은 본 연구에서 제안하는 LLM 기반 스케줄러의 전체 구조를 나타낸다. 제안하는 스케줄러는 기존 규칙 기반 스케줄러의 Job Sorter를 LLMRanker로 대체한 형태로, LLMRanker는 기존 스케줄링 정책을 모사하도록 파인튜닝 된 모델이다. LLMRanker는 J를 입력받아 정렬된 대기 작업 리스트 J_{ranked}를 반환하며, Job Allocator는 이를 입력받아 2.1절에서 언급한 방식과 동일하게 작업을 GPU 클러스터에 할당한다. 그림 4는 GPU 작업 스케줄링을 위한 LLMRanker의 세부 동작을 나타낸다.

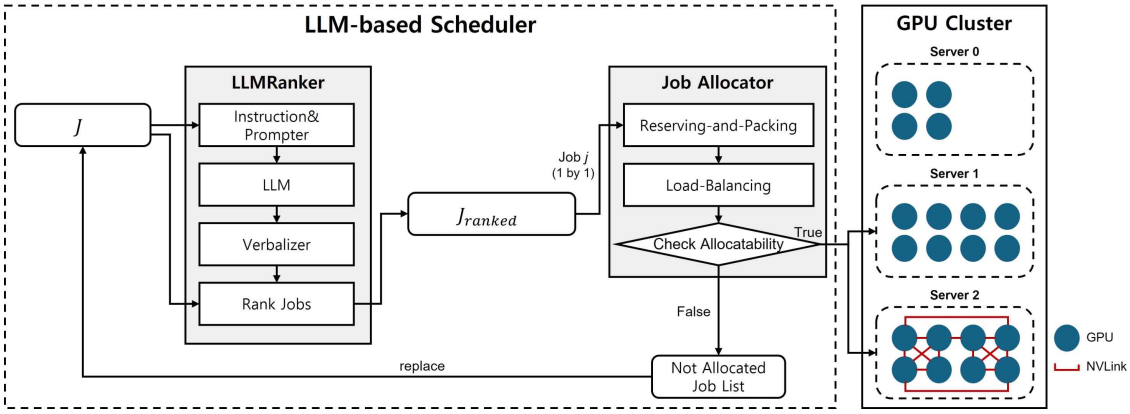


Fig. 3 LLM-based GPU cluster scheduler

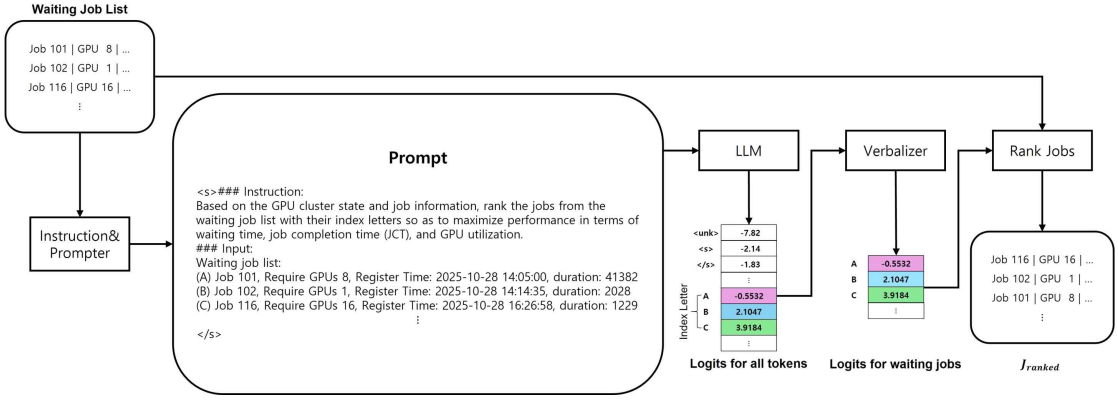


Fig. 4 LLMRanker

Instruction&Prompter단계에서 스케줄링 지침과 J로 구성된 프롬프트를 생성한다. 프롬프트 생성 과정에서 J에 포함된 각 작업은, 리스트의 순서에 따라 영문 알파벳 'A'부터 차례대로 인덱스 문자로 매핑된다. LLM은 생성된 프롬프트를 입력으로 받아 모든 토큰에 대한 logits을 출력한다. verbalizer는 이 logits을 입력받아 후보 작업에 매핑된 인덱스 문자에 해당하는 logit을 추출하고, J에 대한 logits을 변환한다. Rank Jobs 단계에서는 J에 대한 logits를 기반으로 J를 랭킹하여 최종적으로 랭킹된 대기 작업 리스트 J_{ranked}를 생성한다.

Ⅳ. 실험

4.1 실험 환경 설정

LLMRanker는 사전 학습된 LLaMA-2-7B를 기반으로 파인튜닝되었다. 학습 데이터는 3.1장에서 제안한 스케줄링 데이터셋을 사용하였다. 효율적인 파인튜닝을 위해 LoRA 기반의 parameter-efficient fine-tuning 기법을 적용하였다[18]. LoRA 어댑터는 transformer의 query 및 value projection layer에 적용되었으며, rank r=8, scaling factor α=32로 설정하였다. 모델은 AdamW 옵티마이저를 사용하여 learning rate 1×10^{-4}로 1 epoch 동안 학습되었다. 대상 시스템은 16코어 AMD Ryzen Threadripper PRO 3955WX CPU와 48GB 메모리를 갖춘 NVIDIA RTX A6000 GPU 2개

로 구성된다. LLMRanker의 랭킹 성능 평가는 Normalized Discounted Cumulative Gain(NDCG)[19]와 Kendall's tau 상관계수[20]를 사용하며, NDCG는 상위 랭크 정확도를, Kendall's tau는 전체 순위 일관성을 평가한다.

스케줄링 성능 평가는 GPU 클러스터 환경을 모사한 시뮬레이터를 통해 수행되었다. 시뮬레이션 환경은 단일 서버로 구성된 GPU 클러스터로 총 64개의 GPU와 1,486개의 CPU로 구성하였다. GPU 워크로드는 Alibaba에서 제공한 작업 구성 파일에서 총 100,000개의 작업 중 스케줄링 데이터 생성에 사용되지 않은 50,000개의 작업에서 300개를 무작위로 샘플링하여 구성했다. 또한, 서로 다른 시드(seed)를 적용하여 10개의 독립적인 워크로드 세트를 생성하였다. 각 워크로드 세트에 대해 성능을 측정하고, 결과값의 평균과 분산을 계산하여 제시한다. 실제 시스템에서 스케줄러의 성능은 자원이 집중적으로 사용되는 고부하 상황에서 더욱 중요하므로, 본 연구에서는 전체 실행 시간 중 GPU 활용률이 비피크 구간 대비 상대적으로 높게 유지되는 연속 구간을 피크 타임으로 정의하였다. 피크 타임은 규칙 기반 스케줄러 중 SJF 정책의 실행 결과를 기준으로 결정되며, 동일한 피크 타임을 모든 비교 스케줄러에 공통적으로 적용하여 공정한 성능 비교를 수행하였다. 이러한 실험 설정하에, FIFO 및 SJF 정책을 사용하는 규칙 기반 스케줄러와 LLMRanker를 사용하는 LLM 기반 스케줄러를 대상으로 비교하였다. 스케줄링 성능 평가 지표로 Job Completion Time (JCT), 대기 시간, makespan, 그리고 GPU 활용률을 사용하였다.

4.2 LLMRanker 작업 순위 정확도 평가

표 1는 LLMRanker의 작업 순위 예측 성능을 나타낸다. 상위 1개 작업의 NDCG@1은 0.505, 상위 10개 작업의 NDCG@10은 0.549로, 모델이 상위 작업 순위를 비교적 정확하게 예측함을 보여준다. 전체 순위 일관성을 나타내는 Kendall's tau-b는 0.342로, 상위 작업 중심뿐 아니라 전체 순위에서도 일정 수준의 일관성을 달성함을 확인할 수 있다.

Table. 1 Performance evaluation of LLMRanker on job ranking accuracy

Metrics	LLMRanker
NDCG@1	0.5049
NDCG@5	0.5258
NDCG@10	0.5485
Kendall's tau-b(τ_b)	0.3715

4.3 LLM 기반 스케줄러 성능 평가

표. 2는 FIFO 및 SJF 정책을 사용하는 규칙 기반 스케줄러와 LLMRanker를 사용하는 LLM 기반 스케줄러의 성능 비교 결과를 나타낸다. 평균 JCT는 SJF와 LLMRanker가 동일하게 2.07시간으로 가장 낮았으며, FIFO는 2.34시간으로 상대적으로 높았다. 평균 대기 시간은 SJF가 0.63시간, LLMRanker가 0.64시간으로 FIFO의 0.90시간에 비해 낮게 나타났으며, makespan 및 GPU 활용률은 정책 간 큰 차이를 보이지 않았다. 이러한 결과는 제안한 LLMRanker가 규칙 기반 정책과 유사한 시스템 수준 성능을 달성함을 시사한다.

Table. 2 Performance comparison of schedulers with different scheduling strategies on GPU workloads

Scheduling policy	Avg. JCT(h)	Avg. Waiting Time(h)	Makespan(h)	Avg. GPU Util.(%)
SJF	2.07 ± 0.34	0.63 ± 0.27	35.69 ± 11.87	43.42 ± 15.75
FIFO	2.34 ± 0.48	0.9 ± 0.45	35.03 ± 11.43	43.39 ± 16.76
LLMRanker	2.07 ± 0.35	0.64 ± 0.28	35.47 ± 11.99	42.95 ± 16.52

그림. 5는 FIFO 및 SJF 정책을 사용하는 규칙 기반 스케줄러와 제안하는 LLMRanker 기반 스케줄러에 대해, 전체 makespan 동안의 GPU 활용률 변화를 시간에 따라 나타낸 것이다. 모든 스케줄러에서 초기 구간에는 대기 작업이 충분히 존재하여 GPU 자원이 포화 상태로 활용되며, 이후 실행 중인 작업 수가 감소함에 따라 GPU 활용률이 점진적으로 하락하는 공통적인 경향이 관측된다. 정책별로 살펴보면, LLMRanker는 FIFO 및 SJF와 비교하여 전체 makespan 동안 전반적으로 유사한 GPU 활용률 추이를 보이며, 평균 GPU 활용률 또한 큰 차이를 보이지 않는다.

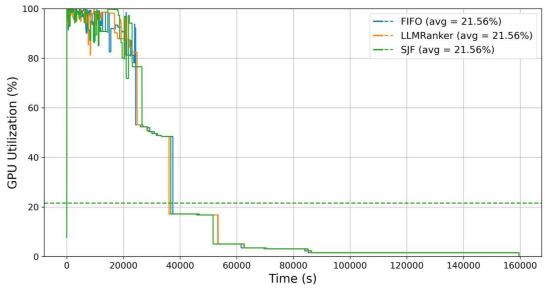


Fig. 5 GPU utilization timeline over makespan

그림. 6은 그림. 5에서 제시된 전체 실행 구간 중, GPU 활용률이 상대적으로 높게 유지되는 구간인 피크 타임에서의 GPU 활용률 변화를 나타낸 것이다. 피크 타임은 4.1절에서 정의한 기준에 따라 SJF 정책 실행 결과 중 0초부터 27,000초까지로 설정하였으며, 모든 비교 스케줄러에 동일하게 적용하였다. 해당 구간에서 세 스케줄러 모두 높은 GPU 활용률을 달성하며, 정책 간 전반적인 활용 패턴은 유사하게 관측된다. LLMRanker는 일부 시점에서 소폭의 변동을 보이지만, 평균 GPU 활용률 기준으로는 기존 규칙 기반 스케줄러와 동등한 수준의 자원 활용도를 달성한다.

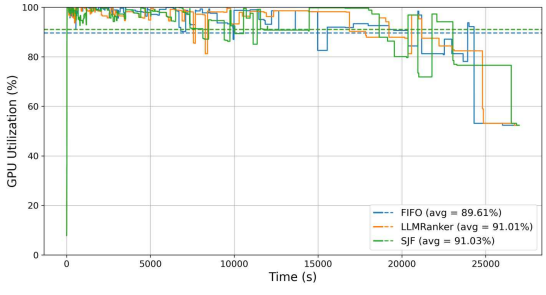


Fig. 6 GPU utilization timeline over peak time

Ⅴ. 결 론

본 연구에서는 LlamaRec의 LLMRanker를 Job Sorter로 활용하는 LLM 기반 GPU 클러스터 스케줄러를 제안하였다. 실험 결과, LLMRanker는 NDCG 및 Kendall's tau 지표에서 높은 순위 예측 성능을 보였으며, JCT, 대기 시간, makespan, GPU 활용률 모두 SJF와 동등한 수준을 달성하였다. 이를 통해 명시적 규칙 설계 없이 데이터 기반 재학습만으로 규칙 기반 스케줄러의 동작을 효과적으로 대체할 수 있음을 확인하였으며, 향후 더 다양한 정책 및 워크로드 환경으로의 확장을 검토할 예정이다.

ACKNOWLEDGEMENTS

This research was financially supported by Hansung University.

REFERENCES

[1] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at Google with Borg,” in *Proceedings of the Tenth European Conference on Computer Systems (EuroSys ’15)*, Bordeaux: FR, pp. 1-17, 2015. DOI: 10.1145/2741948.2741964.

[2] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, “Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning,” *arXiv preprint arXiv: 2008.12260*, Aug. 2020. DOI: 10.48550/arXiv.2008.12260.

[3] Y. Wang, C. Liu, D. Wong, and H. Kim, “GCAPS: GPU context-aware preemptive priority-based scheduling for real-time tasks,” *arXiv preprint arXiv:* 2406.05221, Jun. 2024. DOI: 10.48550/arXiv.2406.05221.

[4] G. Todd, A. Padula, M. Stephenson, E. Piette, D. J. N. J. Soemers, and J. Togelius, “GAVEL: Generating games via evolution and language models,” *arXiv preprint arXiv:* 2407.09388, Jul. 2024. DOI: 10.48550/arXiv.2407.09388.

[5] M. Siavashi, F. K. Dindarloo, D. Kostic, and M. Chiesa, “Priority-aware preemptive scheduling for mixed-priority workloads in MoE inference,” in *Proceedings of the 5th Workshop on Machine Learning and Systems (EuroMLSys ’25)*, Rotterdam: NL, pp. 132-138, 2025. DOI: 10.1145/3721146.3721956.

[6] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, “Learning scheduling algorithms for data processing clusters,” in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM ’19)*, Beijing: CN, pp. 270-288, 2019. DOI: 10.1145/3341302.3342080.

[7] F. Zhang, J. Cao, K. Hwang, K. Li, and S. U. Khan, "Adaptive workflow scheduling on cloud computing platforms with iterative ordinal optimization," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 156-168, Apr.-Jun., 2015. DOI: 10.1109/TCC.2015.2405790.

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," *arXiv preprint arXiv: 2201.11903*, Jan. 2022. DOI: 10.48550/arXiv.2201.11903.

[9] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *arXiv preprint arXiv: 2205.11916*, May 2022. DOI: 10.48550/arXiv.2205.11916.

[10] C. Sun, S. Huang, and D. Pompili, "LLM-based multi-agent decision-making: Challenges and future directions," *IEEE Robotics and Automation Letters*, vol. 10, no. 6, pp. 5681-5688, Jun. 2025. DOI: 10.1109/LRA.2025.3562371.

[11] H. Pei, Y. Gu, Y. Sun, Q. Wang, C. Liu, X. Chen, and L. Cheng, "LLM-based cost-aware task scheduling for cloud computing systems," *Journal of Cloud Computing*, vol. 14, no. 1, Dec. 2025. DOI: 10.1186/s13677-025-00822-0.

[12] Q. Xu, L. Yao, Z. Jiang, G. Jiang, W. Chu, W. Han, W. Zhang, C. Wang, and Y. Tai, "DIRL: Domain-invariant representation learning for generalizable semantic segmentation," *AAAI Conference on Artificial Intelligence*, vol. 36, no. 3, pp. 2884-2892, Jun. 2022. DOI: 10.1609/aaai.v36i3.20193.

[13] Z. Wang, Q. Hu, A. Klimovic, T. Zhang, Y. Wen, P. Sun, and D. Lin, "Semantic-aware scheduling for GPU clusters with large language models," *arXiv preprint arXiv*: 2510.03334, Oct. 2025. DOI: 10.48550/arXiv.2510.03334.

[14] Z. Yue, S. Rabhi, G. de Souza Pereira Moreira, D. Wang, and E. Oldridge, "LlamaRec: Two-stage recommendation using large language models for ranking," *arXiv preprint arXiv*: 2311.02089, Oct. 2023. DOI: 10.48550/arXiv.2311.02089.

[15] Github. alibaba/clusterdata [Internet]. Available: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-gpu-v2020.

[16] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI '22)*, Renton: USA, pp. 1-17, 2022.

[17] Z. Yue, Y. Wang, Z. He, H. Zeng, J. M. Auley, and D. Wang, "Linear recurrent units for sequential recommendation," *arXiv preprint arXiv: 2310.02367*, Oct. 2023. DOI: 10.48550/arXiv.2310.02367.

[18] E. J. Hu, Y. Shen, P. Wallis, Z. A. Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," *arXiv preprint arXiv:* 2106.09685, Jun. 2021. DOI: 10.48550/arXiv.2106.09685.

[19] K. Jarvelin and J. Kekalainen, "Cumulated gain-based evaluation of IR techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422-446, Oct. 2002. DOI: 10.1145/582415.582418.

[20] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81-93, Jun. 1938. DOI: 10.2307/2332226.

황훈순(Hun-Sun Hwang)

2019 - 현재 한성대학교 IT융합공학부
※관심분야 : Graph neural network, 강화학습, 인공지능 컴퓨팅

김명선(Myung-Sun Kim)

2000년 - 2002년 LG전자 전송연구소 주임연구원
2002년 - 2011년 삼성전자 DMC 연구소 책임연구원
2016년 서울대학교 전기컴퓨터공학부 박사 졸업.
2016년 - 2019년 삼성전자 SR연구소 수석연구원
2019 - 현재 한성대학교 AI응용학과 부교수
※ 관심분야 : NPU(인공지능 프로세서), Linux kernel, HW/SW Co-design, 객체인식, 딥러닝 가속 및 병렬 컴퓨팅