JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 29, No. 7: 919~928, Jul. 2025

강화학습 기반 온디바이스 다중 딥러닝 모델 스케줄링

박현빈[1] · 김명선[2*]

Reinforcement Learning-based On-Device Scheduling of Multiple Deep Learning Models

Hyeonbeen Park[1] · Myungsun Kim[2*]

[1]Undergraduate Student, Department of IT Convergence Engineering, Hansung University, Seoul 02876, Korea
[2]*Associate Professor, Department of Applied AI, Hansung University, Seoul 02876, Korea

요 약

최근 개인정보 보호 및 보안 이슈로 인하여 온디바이스 딥러닝 모델 실행의 중요성이 증가하고 있다. 하지만 클라우드 환경 대비 제한된 메모리와 임베디드 GPU와 같은 가속기의 성능 이슈로 인하여, 대규모 모델 또는 다중 모델 추론 실행에 있어서 제약이 발생한다. 본 논문은 온디바이스 환경에서 이러한 제약사항을 극복하고 높은 추론 성능을 달성하기 위한 강화학습 기반 동적 딥러닝 모델 스케줄링 기법을 제안한다. 먼저 오프라인에서 GPU 연산과 CPU와 GPU 간 데이터 송수신을 겹치게 하는 에이전트를 강화학습하고, 온라인에서는 에이전트가 최소 지연의 실행 시간을 보장하는 스케줄링 정책을 제공한다. 또한 딥러닝 모델을 태스크 단위로 분할하고 필요시에만 메모리에 로딩하여 메모리 제약사항을 극복한다. 실제 온디바이스 환경에서 실험 결과, 메모리 제약 조건 하에서 최대 메모리 사용량을 효과적으로 줄이고 효율적인 스케줄링을 통해 평균 추론 시간을 약 30% 단축하였다.

ABSTRACT

In recent years, the importance of running deep learning models on-device has increased due to privacy and security concerns. However, the limited memory and performance of accelerators such as embedded GPUs, compared to cloud environments, limit the execution of large-scale models or multi-model inference. In this paper, we propose a reinforcement learning-based dynamic deep learning model scheduling technique to overcome these limitations and achieve high inference performance in an on-device environment. First, we use reinforcement learning to teach an agent to overlap GPU computation and data transfer between the CPU and GPU offline. This provides the agent with a scheduling policy to ensure minimum latency execution time online. This approach also overcomes memory constraints by partitioning deep learning models into tasks and loading them into memory only when required. Experiments conducted in real on-device environments demonstrate that this approach effectively reduces peak memory usage under constrained conditions and decreases average inference time by approximately 30% through efficient scheduling.

키워드 : 온디바이스 AI, 태스크 스케줄링, GPU 스케줄링, 메모리 제약, 심층 강화학습

Keywords: On-Device AI, Task Scheduling, GPU scheduling, Memory Constraint, Deep Reinforcement Learning

Received 3 June 2025, **Revised** 18 June 2025, **Accepted** 23 June 2025

*** Corresponding Author** Myung-sun 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.2025.29.7.919 pISSN:2234-4772

Ⅰ. 서 론

최근 딥러닝 추론 모델의 빠른 발전과 적용 범위가 스마트 기기, 헬스 케어, 자동차 등 다양한 종류의 임베디드 시스템 또는 온디바이스 환경으로 빠르게 확장되고 있다[1-4]. 온디바이스 환경은 하드웨어 크기, 무게, 예산, 전력 소비, 발열 제한 등의 엄격한 제약 조건을 가진다. 또한 높은 추론 정확도 및 다양한 응용에 대응하기 위하여 딥러닝 모델의 복잡도는 갈수록 높아지고, 다중 모델을 활용하는 사례 역시 늘어나고 있다. 이는 높은 연산량과 큰 메모리 사용량으로 바로 연결된다[5-9]. 대규모 모델 또는 다중 모델 활용 시 필요한 메모리 요구량이 디바이스 내 GPU 메모리의 한계를 넘어서면, 모델 전체를 GPU 메모리에 상주시키는 방식으로는 실행 자체가 불가능하다. 제한된 온디바이스 환경 GPU 메모리에서 대규모 모델 혹은 다중 모델 실행을 가능하게 하려면 모델을 작은 단위로 나누어 필요한 부분만 GPU 메모리에 위치시키는 방법이 필요하다. 이때 분할된 모델의 GPU 메모리로 로딩/오프로딩하는 과정은 CPU-GPU 간의 데이터 송수신 오버헤드를 필연적으로 수반하며, 이는 전체 추론 실행 성능을 저하시킨다[10].

본 논문은 먼저 한정된 GPU 메모리에 전체 딥러닝 모델을 상주시키지 않고 시스템 메모리로부터 GPU에 필요한 딥러닝 모델 연산을 On-Demand 형태로 로딩할 수 있는 기법을 제안한다. 더불어 모델의 로딩 및 오프로딩 과정에서 반드시 발생하는 데이터 송수신에 따른 실행 시간 지연을 최소화한다. 이를 위하여 강화학습을 적용하고 강화학습의 에이전트는 GPU 연산과 모델의 로딩 및 오프로딩 시간을 중첩시킬 수 있게 학습된다. 결과적으로 오프라인에서 실행되는 강화학습은 최적의 스케줄링 정책을 출력하고 온라인에서는 이 정책을 기반으로 딥러닝 모델들은 최소 지연으로 스케줄링된다.

Ⅱ. 연구배경 및 문제정의

2.1 연구 배경

온디바이스 AI 시장은 빠르게 성장하고 있으며, 2025년에는 약 115.9억달러, 한화로 약 16조원 규모에 이를 것으로 예상된다[11]. 병원 의료기기, 스마트 교통 시스템, 산업 자동화, 자율 주행 자동차 등 다양한 분야에서 딥러닝 모델이 온디바이스 형태로 활용되고 있으며[1-4], 사용자와 가까운 엣지에서의 추론은 데이터 발생 즉시 처리를 진행한다는 특징으로 낮은 지연 시간, 데이터 프라이버시 보호, 네트워크 대역폭 비용 절감 등의 장점들을 제공한다[12].

데스크톱 혹은 서버 환경과 비교했을 때, 온디바이스 환경은 작은 메모리 용량 및 낮은 대역폭을 갖는다[13]. 동시에, 딥러닝 모델은 정확도 향상 및 복잡한 태스크 해결을 위해 점차 대형화되는 추세이며, 여러 모델을 조합하는 다중 모델 추론의 사용도 증가하고 있다[7-9]. 모델이 크다는 점과 다중 모델 실행 환경은 온디바이스 환경의 GPU가 가지는 제한된 메모리 용량 및 대역폭과 정면으로 충돌하는 문제점이며, 온디바이스 AI 환경에서 성능 하락의 원인이 된다.

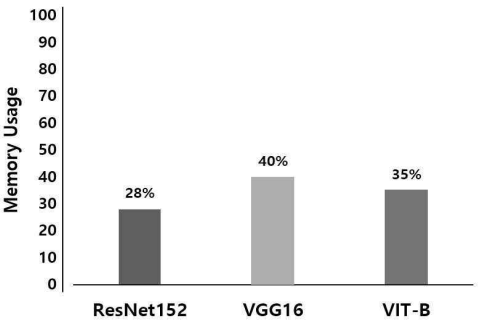

Fig. 1 Example of GPU memory utilization for representative deep learning models.

2.2 문제 정의

온디바이스 환경 GPU의 제한된 메모리 용량은 대규모 모델 또는 다중 모델 추론 파이프라인의 실행을 어렵게 만든다. 그림 1은 GPU 메모리 제한 환경에서 세 가지의 딥러닝 모델(ResNet152, VGG16, VIT-B)의 추론을 수행하여 측정한 최대 메모리 사용량이다. 세 모델을 전부 사용하는 작업이 존재하여 동시에 모든 모델을 GPU에 상주시키려 한다면 가용 GPU 메모리 용량을 초과하여 실행이 불가하다. 메모리 용량을 추가하는 메모리의 증설은 비용, 전력 제약 등으로 인해 항상 실현이 가능한 해결책이 될 수 없으므로[13], 모델 전체를 GPU 메모리에 로딩하지 않고 메모리 제약을 극복하기 위한 새로운 모델 실행 방법이 요구된다.

이 문제를 해결하기 위해 본 논문에서는 모델을 실행

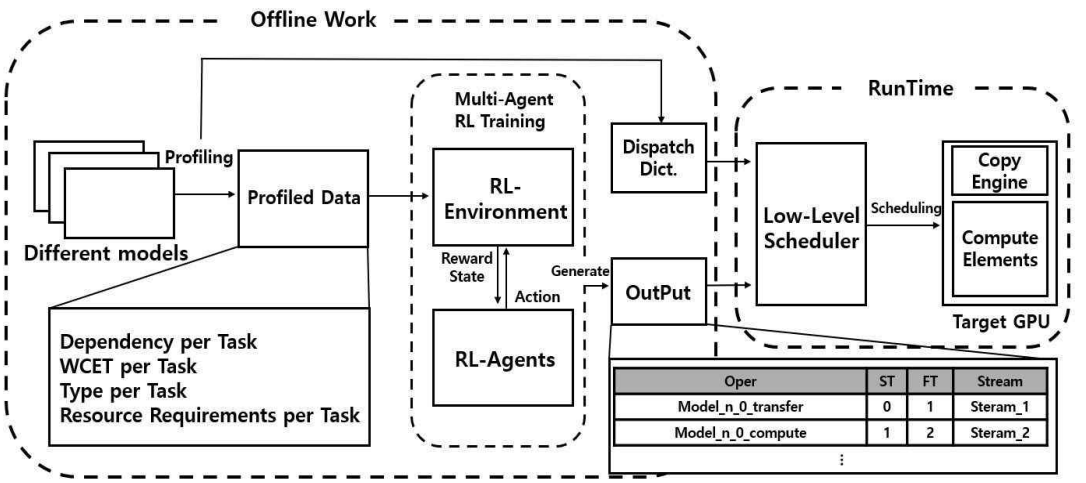

Fig. 2 Solution architecture overview.

가능한 작은 단위로 분할하여, 필요 시에만 분할한 모델을 GPU 메모리에 로딩하고, 사용 완료 혹은 메모리가 부족할 것이라고 예상되는 시점에서 오프로딩을 진행한다. 이 방법을 통해 모델 전체의 크기가 GPU 메모리 용량을 초과하더라도 실행이 가능하다. 그러나 모델의 분할 및 로딩/오프로딩 과정은 CPU-GPU 간의 빈번한 데이터 전송을 유발하며, 이때 소요되는 오버헤드는 전체 추론 시간을 크게 증가시키는 문제를 발생시킨다[10].

본 연구에서는 디바이스 내 GPU 메모리 용량 제약 상황에서도 다중 딥러닝 모델의 실행 가능성을 확보하고, 모델들의 GPU 메모리로 로딩/오프로딩 때 발생하는 데이터 전송 오버헤드를 최소화하여 전체 추론 실행 시간을 단축하는 최적의 스케줄링 기법을 제안한다.

Ⅲ. 강화학습 기반 동적 스케줄링

3.1 전체 아키텍처 개관

그림 2는 제안하는 솔루션의 전체 아키텍처를 보여준다. 제안하는 방법론은 크게 오프라인 단계와 온라인 단계로 나뉜다. 오프라인 단계에서는 대상 딥러닝 모델에 대한 프로파일링을 수행한다. 분석된 정보를 기반으로 강화학습 에이전트들을 학습시켜 최적의 스케줄을 생성한다. 온라인 단계에서는 생성된 스케줄과 프로파일링 정보를 이용하여 로우-레벨 스케줄러가 GPU 연산 및 데이터 전송 작업을 제어하도록 한다.

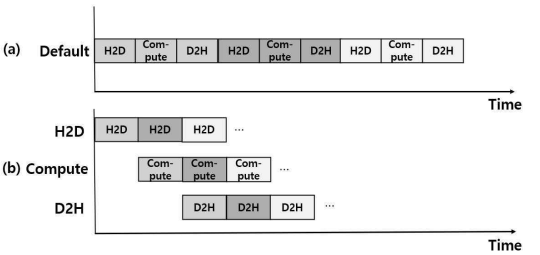

Fig. 3 Example of nesting data transfer and computation using CUDA Streams.

3.1.1 GPU 스트림과 연산의 중첩

GPU는 독립적으로 연산 코어와 메모리 복사 엔진을 갖추고 있다[6]. CUDA와 같은 GPU 프로그래밍 모델에서 스트림은 GPU 상에서 비동기적으로 실행될 커널 및 메모리 작업의 시퀀스를 나타낸다[14]. 서로 다른 스트림에 속한 작업들은 의존성이 없는 한 GPU 하드웨어 자원을 독립적으로 사용하여 병렬적으로 실행될 수 있다. 그림 3-(a)에서는 하나의 스트림(Default stream)에서 메모리 이동 작업(H2D: Host to Device, D2H: Device to Host)과 GPU의 Compute 연산이 모두 진행되는 동작 방식을 보여준다. 그림 3-(b)에서는 이전 모델의 메모리 이동 작업이 진행되는 동안, 다른 스트림에서는 현재 GPU 메모리에 로딩이 완료된 모델의 Compute 연산이 진행되어 연산의 중첩과 실행 속도 증가를 예시한다. 이러한

중첩은 CPU-GPU 간의 데이터 전송 오버헤드를 효과적으로 가릴 수 있어 전체 추론 시간을 단축한다. 제안하는 강화학습 기반 스케줄러는 이러한 GPU 스트림의 특성을 고려하여 최적의 태스크-스트림 할당 및 실행 순서를 결정한다.

3.2 오프라인 프로파일링

본 연구에서는 오프라인 프로파일링 단계를 통해 단일로 실행 가능하며 스케줄링이 가능한 최소 단위인 태스크(Task)들로 분할한 후, 각 태스크의 특성을 분석한다. 태스크는 단일로 실행 가능한 GPU 계산 작업(Compute) 또는 CPU-GPU 간 메모리 이동 작업으로 이루어져있다. H2D, Compute, D2H 중 한 종류의 작업으로만 이루어져 있어야 하며, 태스크 분할은 PyTorch[15]와 같은 딥러닝 프레임워크의 연산 단위 또는 모델 내 정의된 레이어 그룹을 기준으로 수행한다.

프로파일링 과정에서는 각 분할된 태스크에 대해 다음과 같은 정보들을 측정하고 기록한다. 첫째, 해당 태스크의 태스크 타입(Compute, H2D, D2H 중 해당 타입)을 식별한다. 둘째, 해당 태스크를 실행하기 위해 먼저 완료되어야 하는 다른 태스크들의 목록인 의존성 정보를 파악한다. 셋째, 태스크의 안정적인 실행을 보장하기 위한 최악 실행 시간(WCET, Worst-Case Execution Time)을 측정한다. WCET는 각 태스크를 솔루션을 적용할 실제 온디바이스 환경에서 여러 번 반복 실행하여 측정된 실행 시간 중 최댓값을 사용하여 결정하며, torch.cuda.Event[14]와 같은 GPU 이벤트 객체를 활용하여 측정한다. 넷째, 태스크 실행 시 GPU 메모리에 추가적으로 로딩되는 데이터의 크기를 측정하고 기록한다. 오프라인 프로파일링을 통해 수집된 이러한 상세 정보들은 이후 강화학습 에이전트들을 학습하기 위한 환경 입력 데이터로 활용되며, 온라인 단계의 로우-레벨 스케줄러에서 태스크를 빠르게 디스패치하기 위한 디스패치 메타데이터로도 활용된다.

이 디스패치 메타데이터는 로우-레벨 스케줄러가 스케줄에 정의된 태스크 정보에 신속하게 접근하여 해당 태스크를 호출할 수 있도록 딕셔너리 형태로 구성된다. 디스패치 메타데이터는 태스크 분할하며 정의한 이름을 키로 하고 태스크 실행에 필요한 포인터나 실행 함수 정보 등을 값으로 포함하여, 스케줄러가 스케줄에 따라 해당 태스크를 지연 없이 신속하게 호출하고 실행할 수 있도록 지원한다.

3.3 강화학습 기반 스케줄링 학습

3.3.1 DRL 프레임워크 개요

본 연구에서는 다수의 에이전트가 각각 태스크 할당 결정을 내려 보상을 극대화하는 멀티 에이전트 DRL 프레임워크를 설계하였다[16-17]. 그림 4는 논문에서 제안하는 강화학습 기반 스케줄링 프레임워크의 전체 구조를 나타낸다.

프레임워크는 강화학습 에이전트와 스케줄링 환경으로 구성된다. 환경은 OpenAI Gym[18] 규격을 준수하며, 3.2절에서 프로파일링된 딥러닝 모델의 태스크 정보(태스크 타입, 의존성 정보, 실행 시간, 메모리 요구량)를 바탕으로 구성된다. 에이전트는 현재 환경 상태를 관측하고, 다음 태스크의 GPU 스트림 할당 여부에 대한 행동을 결정한다. 환경은 에이전트의 행동에 따라 상태를 업데이트하고, 사전에 정의된 보상 함수에 따라 보상을 계산하여 에이전트에게 전달한다. 에이전트는 이 보상을 통해 각각의 누적 보상을 최대화하는 방향으로 스케줄링 정책을 학습한다.

3.3.2 DRL 에이전트 설계

스케줄링 대상 태스크의 특성에 따라 세 가지 유형의 DRL 에이전트를 설계하였다. 각 에이전트는 특정 타입의 태스크를 책임지고 해당 태스크를 에이전트가 맡은 스트림에 할당할지를 결정한다.

Compute 에이전트는 GPU 연산 태스크의 스케줄링을 담당하며 어떤 연산 스트림에 할당할지를 결정한다. H2D 에이전트는 CPU 메모리에서 GPU 메모리로 데이터를 전송하는 H2D(Host to Device) 태스크를 H2D 전용 스트림(h2d)에 할당할지를 결정한다. D2H 에이전트는 GPU 메모리에서 CPU 메모리로 데이터를 전송하는 D2H(Device to Host) 태스크를 D2H 전용 스트림(d2h)에 할당할지를 결정한다. 각각의 에이전트들은 독립적으로 행동을 결정하지만, 환경을 통해 서로의 행동 결과를 공유하고 협력적으로 전체 시스템의 성능을 최적화하도록 학습된다.

3.3.3 환경 설계

DRL 에이전트의 환경은 3.2절에서 수집한 프로파일링 정보를 기반으로 구성된다. 환경은 스케줄링 준비가

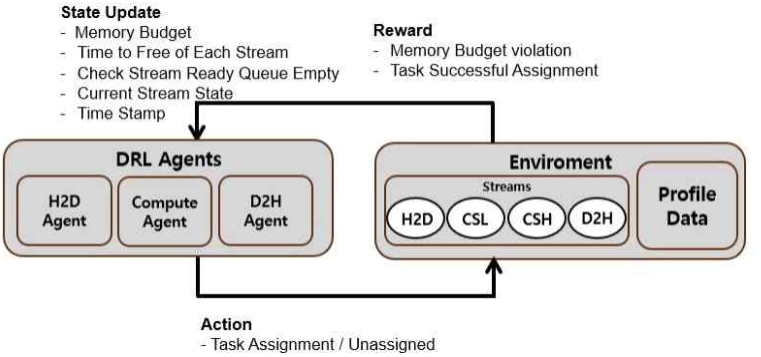

Fig. 4 Workflow of the proposed reinforcement learning.

된 태스크들의 상태, 현재 스트림들의 사용 상태, 현재 스트림들의 남은 사용 시간, GPU 메모리 사용량의 정보를 관리한다. 환경에서는 에이전트들의 행동 결정(스트림에 작업을 할당/미할당)을 받아 현재 상태를 업데이트하고, 보상 함수에 따라 보상을 계산하여 에이전트에게 전달함으로써 강화학습 루프를 진행한다.

h2d_fs	csh_fs	csl_fs	d2h_fs	h2d_ttf	csh_ttf
csl_ttf	d2h_ttf	MB	h2d_QE	cs_QE	d2h_QE

Fig. 5 State space of reinforcement learning.

3.3.4 상태 공간 설계

DRL 에이전트의 상태 공간은 에이전트들이 환경의 현재 상황을 정확하게 인지하고 이를 바탕으로 합리적인 스케줄링 결정을 내리는 데 필수적인 정보들로 구성된다. 본 연구에서 에이전트의 상태 공간에는 그림 5와 같은 핵심 정보들을 포함한다. 첫째, 각 스트림의 stream_fs(free_state)가 존재한다. 스트림의 현재 사용 가능 여부를 0과 1로 나타내며 각각 사용 불가와 가능을 나타낸다. 둘째, 각 스트림의 stream_ttf(time to free) 공간들이 존재하며, 스트림을 사용중인 태스크가 완료될 때까지 남은 시간정보를 기록한다. 셋째, 현재 메모리 사용 정보를 기록하는 MB(memory_budget) 공간이 있다. 마지막으로, 작업 타입마다 존재하는 대기큐의 상태를 확인하기 위한 type_QE(Queue_Empty) 공간이 존재한다. 0과 1 두 개의 값을 가지며 0은 대기큐에 작업이 존재한다는 것을 나타내고, 1은 작업이 없음을 나타낸다[16].

이러한 상태 정보들을 종합적으로 파악함으로써, 에이전트들은 현재 시스템 자원의 가용 상태, 메모리 제약 조건, 그리고 스케줄링 대상 태스크들의 준비 상태 등을 인지하고, 상태 공간을 기반으로 특정 태스크를 해당 스트림 할당 할지, 미할당시켜 태스크를 대기시킬지 최적의 행동을 결정하게 된다.

3.3.5 에이전트 행동 설계

각 DRL 에이전트는 현재 상태 공간과 스케줄링 대상 태스크가 주어졌을 때, 사전에 정의된 행동 공간 내에서 가장 높은 보상을 가지는 행동을 선택한다. 에이전트는 행동을 결정해야하는 시점마다 상태 공간을 관측한다. 만일 h2d 작업의 대기큐에 태스크가 존재하고 행동을 결정해야할 때, 먼저 H2D 에이전트는 상태 공간의 h2d_fs 와 h2d_ttf 공간을 관측하고 h2d_fs의 값이 현재 1이라면 에이전트는 대기 중인 태스크를 스트림에 할당하는 행동을 결정한다. h2d_fs가 0이라면 스트림 미할당을 결정한다.

상태 공간을 관측하며 각 에이전트가 결정하는 행동들은 다음과 같다. 먼저 Compute 에이전트의 행동은 세 가지 값으로 정의된다. 행동 0은 현재 스케줄링 대상 태스크를 어떤 스트림에도 할당하지 않고 스케줄링을 다음 시점으로 대기(Hold)시키는 것을 의미한다. 행동 1은 해당 태스크를 csh(Compute stream-High)에 할당하는 행동이며, 행동 2는 csl(Compute stream-Low)에 할당하는 행동이다. 본 연구에서 Compute 스트림의 High, Low는 단순히 스트림을 구분하기 위한 명칭이다.

H2D 에이전트의 행동은 두 가지 값으로 정의된다. 행동 0은 현재 스케줄링 대상 H2D 태스크를 할당하지

않고 대기시키는 것을 의미하며, 행동 1은 해당 H2D 태스크를 h2d 스트림에 할당하는 행동이다.

D2H 에이전트의 행동 역시 두 가지 값으로 정의된다. 행동 0은 현재 스케줄링 대상 D2H 태스크를 할당하지 않고 대기시키는 것을 의미하며, 행동 1은 해당 D2H 태스크를 d2h 스트림에 할당하는 행동이다.

여기서 행동 0에 해당하는 미할당/대기 행동은 에이전트가 현재 시점에서 특정 태스크의 스케줄링을 의도적으로 미루는 전략적 행동이다. 미할당/대기 행동은 주로 다음과 같은 상황에서 활용될 수 있다. 태스크가 대기큐에서 대기중이고, 할당 스트림이 비어있을 때 에이전트가 태스크를 스트림에 무조건 할당한다면 GPU 메모리를 초과할 수 있다. 에이전트는 태스크의 할당 행동 결정 전, MB 상태 공간을 확인한다. 현 시점에서 작업이 할당되었을 때 메모리 제한을 초과하게 된다면 에이전트는 메모리 제한을 회피하는 미할당 행동을 결정한다. 이처럼, 시스템 요구사항을 만족시키기 위해 미할당/대기 행동을 활용하게 된다.

3.3.6 보상 함수 설계

DRL 에이전트의 각 행동에 대한 환경의 피드백은 보상 값의 형태로 제공되며, 이는 에이전트가 목표 달성에 기여하는 행동의 가치를 학습하게 한다. 본 연구에서 사용된 주요 보상 및 페널티 조건은 다음과 같은 상황에 따라 결정된다.

첫째, 스케줄링 대상 태스크를 적절한 스트림에 성공적으로 할당했을 경우 에이전트는 +1의 보상을 받는다. 이는 에이전트가 태스크 스케줄링을 적극적으로 수행하도록 장려한다. 둘째, 현재 스케줄링 대상 태스크가 해당 에이전트 타입의 대기 큐에 존재하지 않음에도 불구하고 스트림에 할당하는 행동을 결정하려 할 경우, -1의 페널티를 부여한다. 셋째, 스케줄링 대상 태스크가 존재하지만, 해당 태스크를 할당하려는 스트림이 현재 다른 작업으로 사용 중일 때 할당하는 행동을 결정하려 할 경우, -1의 페널티를 부여한다. 넷째, 태스크가 스케줄링될 준비가 되었고 해당 스트림이 사용 가능함에도 불구하고, 에이전트가 태스크를 스트림에 할당하지 않고 불필요하게 대기하는 행동을 선택할 경우, -1의 페널티를 부여한다. 이 페널티는 에이전트가 자원을 효율적으로 사용하도록 유도한다. 다섯째, 메모리 용량 초과를 회피하기 위해 태스크를 스트림에 할당하지 않고 대기(Hold)하는 행동을 결정할 경우, +0의 보상을 부여한다. 메모리 용량을 초과할 경우 바로 학습 루프가 종료되기 때문에 메모리 할당을 피하도록 학습을 유도한다. 여섯째, 스케줄링 대상 태스크가 없어 미할당/대기 결정 시에도 +0의 보상을 부여한다. 이는 메모리 제약을 준수하기 위한 전략적인 대기 행동이나 태스크 부재 시의 대기에 대해 페널티를 부여하지 않음을 의미한다. 마지막으로, 스케줄 실행 중 어느 시점에서라도 실제 메모리 사용량이 사전에 설정된 메모리 예산을 초과할 경우, 즉각적으로 해당 학습 루프를 종료하고 다음 루프로 재진행한다. 이는 메모리 예산 초과가 허용되지 않는 환경임을 에이전트에게 알리는 역할을 한다.

각 DRL 에이전트가 최종적으로 받는 보상은 해당 에이전트의 행동 결과에 따라 위에 정의된 조건들 중 해당하는 보상 값을 합산하여 결정된다.

Compute 에이전트는 성공적인 태스크 할당 시 보상(+1), 대상 태스크 부재 및 스트림 사용 중 부적절한 할당 시 페널티(-1), 불필요한 대기 시 페널티(-1)를 받는다. H2D 에이전트는 Compute 에이전트와 동일한 보상 조건에 더해, 메모리 용량 초과 회피를 위한 대기 시 +0의 보상을 받으며 메모리 예산 초과 시 학습 종료 제약을 따른다. D2H 에이전트는 Compute 에이전트와 동일한 보상 조건을 적용받는다. 보상 함수 설계를 통해, DRL 에이전트들은 단순히 태스크를 빠르게 끝내도록 학습될 뿐만 아니라, 부적절하거나 비효율적인 자원 할당을 피한다. 특히, H2D 에이전트의 학습에서는 메모리 제약 조건을 적극적으로 고려하여 이를 준수하는 스케줄링 전략을 학습하게 된다.

3.3.7 학습 알고리즘

태스크 스케줄링 문제는 상태 공간과 행동 공간이 크고 복잡하여, 전통적인 Q-러닝 기법으로는 효율적인 학습이 어렵다. 본 연구에서는 심층 신경망을 사용하여 특정 상태에서 특정 행동을 취했을 때 얻을 수 있는 예상 누적 보상 값을 근사화함으로써 고차원 문제를 효과적으로 다룰 수 있는 DQN(Deep Q-Network) 알고리즘을 실험적으로 채택하였다[19-20].

3.4 스케줄 생성

학습이 완료된 강화학습 에이전트의 정책을 활용하여 실제 태스크 스케줄을 생성한다. 이 스케줄은 딥러

닝 모델을 구성하는 각 태스크(레이어 연산 및 메모리 이동)의 실행 순서 및 시점을 구체적으로 정의하는 테이블 형태로 표현된다. 스케줄 테이블은 다음과 같은 핵심 항목들을 포함한다.

첫째, 각 태스크를 고유하게 식별하기 위한 태스크 ID가 기록된다. 둘째, 해당 태스크가 할당되어 실행될 GPU 할당 스트림(h2d, csh, csl, d2h 중 하나) 정보가 포함된다. 셋째, 강화학습 에이전트의 결정에 따른 해당 태스크의 스케줄된 ST(Start Time)가 명시된다. 마지막으로, 태스크의 FT(Finish Time)가 기록되는데, 이는 스케줄된 시작 시간에 해당 태스크의 프로파일링된 WCET 를 더하여 계산된 예상 완료 시점이다. 이 스케줄된 완료 시간은 이후 태스크들의 스케줄링 시, 해당 태스크가 의존하는 선행 태스크의 완료 시점을 판단하는 기준으로 활용되어 종속성 관계를 보장한다.

생성된 실행 스케줄은 각 태스크가 GPU의 어떤 스트림에서 언제 시작하여 언제 완료될 것인지에 대한 상세 계획을 담고 있다. 강화학습 에이전트는 이러한 스케줄 생성 과정에서 GPU 스트림의 비동기 실행 특성을 고려하여 H2D, Compute, D2H 등 다양한 타입의 태스크들 간의 중첩 실행을 최대화하는 스케줄을 탐색하고 결정한다.

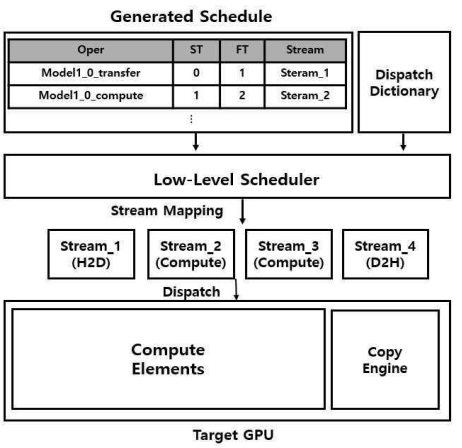

Fig. 6 Low-level scheduler.

3.5 온라인 로우-레벨 스케줄러

그림 6은 로우-레벨 스케줄러의 동작 구조를 보여준다. 로우-레벨 스케줄러는 오프라인에서 생성된 실행 스케줄과 디스패치 메타데이터를 기반으로 작성된 디스패치 딕셔너리를 입력으로 받아 GPU를 제어한다.

스케줄러의 기본 동작 원리는 시간 기반 디스패치이다. 스케줄러는 시스템의 현재 시간을 타이머를 통해 지속적으로 모니터링한다. 생성된 스케줄 테이블에는 각 태스크의 스케줄 시작 시간이 명시되어 있으며 디스패치 딕셔너리를 사용하여 시작 시간에 도달하거나 초과한 시점에 해당 작업을 실행 준비 상태로 인식하고 디스패치를 진행한다.

실행 준비 상태가 된 태스크는 미리 스케줄에 명시된 CUDA 스트림으로 디스패치된다. 작업이 디스패치 되면 해당 스트림 내에서 작업을 비동기적으로 실행 큐에 추가하고 순차적으로 처리한다.

태스크간의 선후관계와 같은 의존성은 스케줄 생성 단계에서 반영되었으며, 로우-레벨 스케줄러는 스케줄된 순서대로 스케줄을 실행하되, 필요에 따라 cudaStreamWaitEvent 와 같은 CUDA API를 사용하여 선행 태스크의 완료를 기다리는 동기화 작업을 수행할 수 있다[14]. 실제 실행 시간이 WCET 보다 짧을 경우 스케줄러는 다음 태스크를 시작 시간에 맞추어 실행하거나, 의존성이 충족되는 즉시 유연하게 다음 태스크를 시작하도록 확장 구현할 수 있다.

시간 모니터링, 스케줄 탐색, 스트림 기반 디스패치, 그리고 필요한 동기화 매커니즘이 로우-레벨 스케줄러의 핵심 기능을 구성하며, 이는 오프라인에서 생성된 실행 스케줄을 GPU 상에서 재현한다.

Ⅳ. 실 험

실험을 위한 하드웨어 환경은 GPU GTX 1650, CPU i7-10510U, RAM 16GB로 구성된다. 소프트웨어 환경으로는 Python 3.9.21, PyTorch 2.5.1, CUDA 11.8, OpenAI Gym 0.26.2 버전을 사용하였다. 가용 GPU 메모리(VRAM)는 2GB이다. 실험에 사용한 대상 딥러닝 모델은 ResNet-152, VGG16, ViT-B이다[7,8,21]. 이 세 모델이 동시에 GPU 메모리에 적재되어 추론이 실행되어야 하는 상황을 주요 실험 시나리오로 설정하여, 제안하는 레이어 단위 스케줄링 기법이 2GB 메모리 제약을 극복하고 성능 손실을 최소화하며 효율적인 추론 속도를 달성하는지를 평가하였다.

4.1 성능 평가

세 가지 모델(ResNet-152, VGG16, ViT-B)이 동시에 추론되는 시나리오에서, 제안된 솔루션 적용 전(Org.)과 적용 후(Sol.)의 평균 추론 완료 시간을 측정하였다.

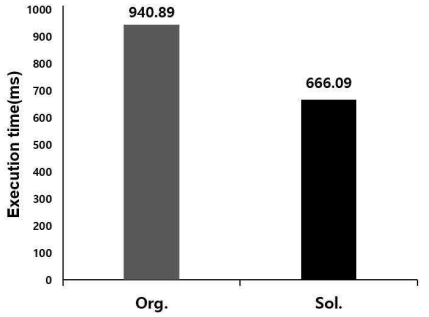

Fig. 7 Average inference performance.

4.1.1 평균 추론 시간 비교

그림 7은 솔루션을 적용한 시스템의 전후 추론시간을 측정한 것이다. 측정 결과 제안된 솔루션을 적용하여 약 30%의 성능 향상을 보였다. 강화학습 스케줄러가 생성한 스케줄 표에 기반한 예상 완료 시간은 622ms였다. 실제 측정된 평균 추론 시간인 666.09ms와 근접한 결과를 보여, 제안하는 스케줄링 기법이 효과적으로 추론 시간을 단축시켰음을 확인할 수 있다.

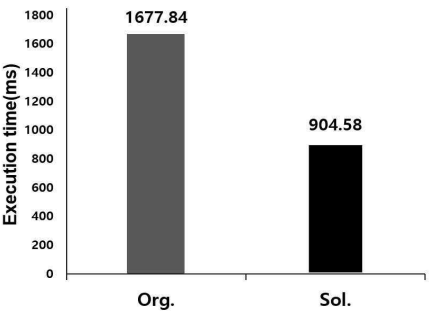

Fig. 8 Average execution time for first inferences

4.1.2 첫 번째 추론 실행 시간 분석

GPU에서 딥러닝 모델을 가장 처음 실행할 때 발생하는 프로세스 초기화, 스레드 생성, GPU 캐시 워밍업, 라이브러리 로딩 등 다양한 초기 오버헤드는 전체 실행 시간에 영향을 줄 수 있다. 특히 WCET 기반으로 스케줄을 생성하는 방식에서는 이러한 초기 오버헤드가 성능에 영향을 미칠 가능성이 있다.

그림 8은 그림 7의 실험과 동일한 환경에서 가장 맨 처음 모델 추론 시간을 비교한 결과이다. 솔루션 적용 전은 1677.84ms이며, 솔루션 적용 후는 904.58ms이다. 초기 오버헤드가 포함된 상황에서도 제안 기법이 유의미한 성능 향상을 달성하였다. 이는 메모리 이동 연산과 GPU 계산 연산의 중첩 스케줄링 기법이 런타임 환경의 다양한 초기 오버헤드 속에서도 충분히 효과적이며, 단순히 모델 전체를 로딩하여 계산하는 것보다 효율적임을 입증한다.

또한, 그림 8에서 솔루션을 적용한 모델의 평균 실행 시간을 측정할 시 첫 실행을 제외하고 이후 실행 차수들의 평균을 측정한다면 평균 추론 시간은 632ms이며 더더욱 스케줄 표와 근접한 결과를 보여준다.

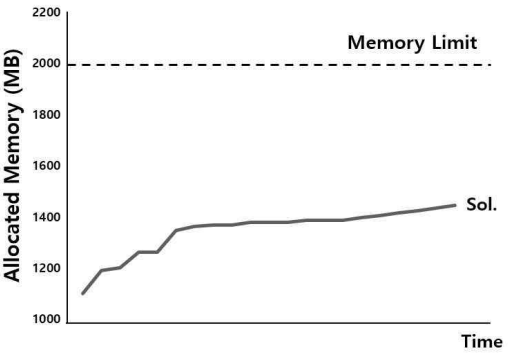

Fig. 9 GPU memory usage over time

4.2 메모리 사용량 분석

본 실험에서는 일정 이상의 메모리 사용량을 초과하는 경우 로우-레벨 스케줄러의 스케줄링 단계에서 실행이 완료된 레이어들을 동적으로 할당 해제하게 하였다.

그림 9는 제안 솔루션 적용 전후 최대 메모리 사용량을 측정한 것이다. 실험에 사용한 세 가지 모델 전체를 2GB로 제한된 GPU 메모리에 동시에 로딩하려 할 경우, 모델들의 총 메모리 요구량이 2GB를 초과하여 실행 자체가 불가능했다. 반면, 제안 솔루션을 적용한 후에는 모델 레이어들이 필요에 따라 로딩/오프로딩되면서, 연산량이 가장 많이 겹치는 시점의 최대 메모리 사용량이 약 1.45GB 수준으로 측정되었다. 이는 2GB의 GPU 메모리 용량 제약을 회피할 수 있음을 보여준다.

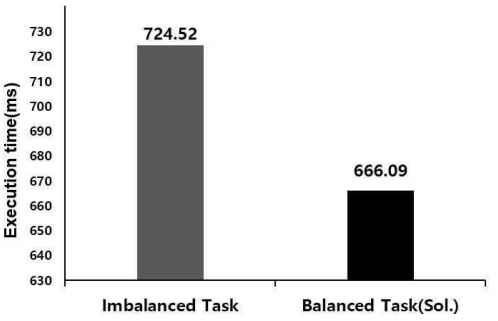

Fig. 10 Execution time for imbalanced and balanced task-time scenarios.

4.3 태스크 크기 불균형이 성능에 끼치는 영향

본 절에서는 딥러닝 태스크 분할 시 태스크 간 크기 불균형이 전체 추론 성능에 미치는 영향을 분석하기 위한 실험을 수행하였다. 이를 위해 두 가지 시나리오를 구성하였다. 먼저, 프로파일링에서 각 분할된 태스크의 WCET를 고려하지 않고 최대한 세분화하여, 태스크 간 WCET가 100배 이상 차이 나는 불균형 시나리오를 구성하였고, 태스크들의 임의의 집합을 구성하여 태스크 간의 WCET 불균형이 20배를 넘지 않도록 조정하여 균형 시나리오를 구성하였다. 각 시나리오에서 분할, 정의된 태스크 정보를 강화학습 모델의 입력 데이터로 사용하여 학습을 진행한 후 스케줄을 생성하였고, 동일한 로우-레벨 스케줄러로 평균 추론 시간을 측정하였다. 그림 10에 제시된 평균 추론 시간을 살펴보면, 균형 시나리오의 스케줄은 666.09ms가 측정되었고, 불균형 시나리오의 스케줄은 724.52ms로 더 긴 시간이 소요되었다. 이는 태스크 간 실행 시간의 차이가 클수록 스케줄 생성 시 GPU 연산과 메모리 이동 간의 중첩 효율이 감소한다는 것을 명확히 보여준다.

강화학습을 통해 생성된 스케줄을 분석한 결과, 디스패치되는 각 태스크는 의존성을 가지므로 이전 태스크가 완료되어야 다음 태스크가 실행될 수 있다. 이 과정에서 태스크의 실행 시간 불균형이 커질 경우, 스케줄 내에서 GPU가 메모리 이동 작업만 수행하며 유휴 상태로 대기하는 시간이 늘어나는 경향을 보였다. 메모리 이동에 총 소요되는 WCET는 고정되어 있으므로 태스크의 불균형이 GPU 연산의 중첩 효율 및 성능에 부정적인 영향을 미치는 것으로 분석된다.

Ⅴ. 결 론

본 논문에서는 온디바이스 환경에서 GPU 메모리 용량을 초과하는 다중 딥러닝 모델의 효율적인 실행 기법을 제안하였다. 이 기법은 강화학습을 기반으로 모델들을 On-Demand 형태로 동적 스케줄링한다. 모델들의 로딩 및 오프로딩에서 발생하는 지연 시간을 최소화하기 위해, 오프라인 프로파일링으로 데이터를 수집하고 메모리 이동과 GPU의 연산 작업을 효과적으로 중첩시키는 스케줄링 정책을 출력하도록 강화학습 에이전트를 학습하였다. 세 종류의 다중 딥러닝 모델을 동시에 추론한 결과, 강화학습으로 출력된 스케줄링 정책 반영 후 온라인에서 실행 시간이 30% 감소했다.

ACKNOWLEDGEMENTS

This research was financially supported by Hansung University.

REFERENCES

[1] K. Tan, X. Zhang, and D. Wang, “Deep learning based real-time speech enhancement for dual-microphone mobile phones,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 1853-1863, 2021. DOI: 10.1109/TASLP.2021.3082318.

[2] B. Kisacanin, “Deep learning for autonomous vehicles,” in *Proceedings of the* 47th International Symposium on Multiple-Valued Logic (ISMVL), Novi Sad: RS, pp. 142-142, 2017. DOI: 10.1109/ISMVL.2017.49.

[3] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, “Deep learning sensor fusion for autonomous vehicle perception and localization: A review,” *Sensors*, vol. 20, no. 15, pp. 4220, Jul. 2020. DOI: 10.3390/s20154220.

[4] B. Norgeot, B. S. Glicksberg, and A. J. Butte, “A call for deep-learning healthcare,” *Nature Medicine*, vol. 25, no. 1, pp. 14-15, Jan. 2019. DOI: 10.1038/s41591-018-0320-3.

[5] Google Cloud. Introducing Cloud TPU [Internet]. Available: https://cloud.google.com/edge-tpu/.

[6] NVIDIA. The Ultimate Platform for Physical AI and Robotics [Internet].

Available: https://www.nvidia.com/en-sg/autonomous-machines/embedded-systems/.

[7] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, Sep. 2014. DOI: 10.48550/arXiv.1409.1556.

[8] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas: USA, pp. 770-778, 2016. DOI: 10.1109/CVPR.2016.90.

[9] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of* IEEE Conference on Computer Vision and Pattern Recognition, Honolulu: USA, pp. 2261-2269, 2017. DOI: 10.1109/CVPR.2017.243.

[10] S. Chen, Z. Wang, Z. Guan, Y. Liu, and P. B. Gibbons, “Practical Offloading for Fine-Tuning LLM on Commodity GPU via Learned Sparse Projectors,” *arXiv preprint arXiv:2406.10181*, Jun. 2024. DOI: 10.48550/arXiv.2406.10181.

[11] Research Nester. Embedded AI Market [Internet]. Available: https://www.researchnester.com/kr/reports/embedded-ai-market/5715.

[12] C. J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia et al., “Machine learning at facebook: Understanding inference at the edge,” in *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Washington: USA, pp. 331-344, 2019. DOI: 10.1109/HPCA.2019.00048.

[13] A. Biglari and W. Tang, “A Review of Embedded Machine Learning Based on Hardware, Application, and Sensing Scheme,” *Sensors*, vol. 23, no. 4, pp. 2131, Feb. 2023. DOI: 10.3390/s23042131.

[14] NVIDIA. CUDA C++ Programming Guide [Internet]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[15] PyTorch. PyTorch Documentation(Get Started) [Internet]. Available: https://pytorch.org/.

[16] D. Pereira, S. Ghosh, and S. Dey, “Multi-Stream Scheduling of Inference Pipelines on Edge Devices - a DRL Approach,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 6, pp. 1-36, Sep. 2024. DOI: 10.1145/3677378.

[17] T. Chu, J. Wang, L. Codeca, and Z. Li, “Multi-Agent Deep Reinforcement Learning for Large-scale Traffic Signal Control,” *arXiv preprint arXiv:1903.04527*, Mar. 2019. DOI: 10.48550/arXiv.1903.04527.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” arXiv preprint arXiv:1606.01540, Jun. 2016. DOI: 10.48550/arXiv.1606.01540.

[19] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, pp. 279-292, 1992. DOI: 10.1007/BF00992698.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *arXiv preprint arXiv:1312.5602*, Dec. 2013. DOI: 10.48550/arXiv.1312.5602.

[21] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” *arXiv preprint arXiv:2010.11929*, Oct. 2020. DOI: 10.48550/arXiv.2010.11929.

박현빈(Hyeon-been Park)

2020~현재 한성대학교 IT융합공학부

※ 관심분야 : 딥러닝 최적화, Linux kernel, 병렬 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 IT융합공학부 부교수

※ 관심분야 : NPU(인공지능 프로세서), Linux kernel, HW/SW Co-design, 객체인식, 딥러닝 가속 및 병렬 컴퓨팅