강화학습 최종 프로젝트

20201444 김민서
120250327 송근수
120250336 이승우
git : https://github.com/seungwooLee00/rl_proj

프로젝트 주제 및 목표

프로젝트 주제

MCU 환경에서 동작 가능한 효율적 이미지 분류 모델을 찾기 위한 강화학습 기반 Neural Architecture Search(NAS) 설계 및 평가

프로젝트 목표

- Flash·SRAM·Latency 제약을 동시에 만족하는 서브넷을 탐색
- 기존 유전자 알고리즘(GA) 기반 탐색 대비 더 나은 정확도 - 지연시간 - 자원 사용량 trade-off 확보
- PPO를 활용한 정책 기반 NAS가 제약 기반 모델 탐색에서 실질적 성능 향상을 달성할 수 있는지 검증
- 보상 설계, 탐색절차, 예측기 기반 환경 구성 등 강화학습 탐색이 성공적으로 작동하기 위한 조건을 분석

기여도

김민서 : 33%

송근수 : 33%

이승우 : 34%

git : https://github.com/seungwooLee00/rl_proj

신경망 자동탐색

NAS(Neural Architecture Search)가 등장한 배경

딥러닝 모델 구조 설계는 많은 전문 지식과 시행착오를 요구한다.
모델 규모가 커지거나 극단적으로 작아지는 환경에서 사람이 손으로 구조를 최적화하는 비용이 매우 커졌고, 그래서 신경망 자동 탐색 기술이 제안되었다.

NAS의 기본 개념

NAS는 주어진 검색 공간(search space) 안에서 신경망 구조를 탐색해 특정 목표(정확도, 연산량, 지연시간 등)를 최적화하는 절차이다.

1. **검색 공간**은 가능한 연산 종류, 연결 방식, 블록 조합 등이 정의된 영역이다.
2. **탐색 알고리즘**은 이 공간을 어떻게 돌아다니며 후보 구조를 선택할지를 결정한다.
3. **평가 함수**는 선택된 구조가 실제로 얼마나 성능이 좋은지를 반환한다.

*One-shot NAS

- NAS는 강화학습 기반 NASNet, 진화 전략 기반 AmoebaNet, 미분가능 NAS인 DARTS 등 다양한 탐색 방식으로 발전해왔다.
  이들 방식과 달리, One-shot NAS 계열은 **하나의 슈퍼넷(supernet)을 학습한 뒤 서브넷을 재학습 없이 평가**할 수 있어 탐색 효율이 매우 높다.
- 이번 과제에서 사용하는 탐색 방식은 **OFA(Once-for-All Network) 계열의 weight-sharing 기반 접근**에 속하며,
  슈퍼넷을 학습한 후 그 안에서 메인 구조와 보조분류기 구성을 동시에 탐색하는 형태로 확장되어 있다.

*Cai, Han, et al. "Once-for-all: Train one network and specialize it for efficient deployment." *arXiv preprint arXiv:1908.09791* (2019).

MCUNet

MCU 환경과 제약

임베디드 디바이스(MCU)는 수십 KB 수준의 SRAM, 수백 KB~수 MB 수준의 플래시, 매우 낮은 연산 처리량 등 극단적으로 제한된 자원을 가진다.
이 환경에서는 모델 구조가 MCU의 제약조건(Flash, SRAM 요구량)을 만족하면서 모델의 정확도가 높은 것이 필수적이며, 이러한 상충하는 요구 사이에서 최적의 구조를 찾는 것이 본 연구의 핵심 목표가 된다.

*MCUNet의 핵심 접근 방식

MCU 친화적인 검색공간 설계

- MCU의 SRAM 크기, 플래시 용량, 연산 패턴, 버퍼 재사용 구조 등을 반영해서 MCU에서 실제로 실행 가능한 연산 조합과 블록 구조만으로 검색 공간을 제한한다.
- 일반 NAS처럼 폭넓은 연산 조합을 허용하는 것이 아니라, 메모리 사용량을 예측 가능하게 유지하면서도 정확도를 낼 수 있는 구조로 구성한다.

탐색 알고리즘

- MCUNet은 weight-sharing 기반 supernet을 학습한 뒤 유전자 알고리즘(Genetic Algorithm)으로 다양한 서브넷을 샘플링·평가하여 최적 구조를 선택한다.
- 정확도 예측기와 지연시간 예측기를 함께 사용해 정확도 - 지연시간 tradeoff를 만족하면서 메모리 제약을 동시에 만족하는 구조를 찾는다.

TinyEngine

MCUNet의 큰 특징은 단순히 모델 구조만 설계하는 것이 아니라, 모델 구조와 inference engine을 함께 최적화하는 전체 스택 설계라는 점이다.

- TinyEngine은 MCU 메모리 구조에 맞춘 초경량 추론 엔진으로, 네트워크의 연산 순서, 버퍼 크기, 메모리 재사용 등을 모델 구조와 함께 조정한다.
- 이 덕분에 탐색 결과 모델은 실제로 MCU에서 메모리 오버헤드 없이 동작하고, 예측된 지연시간과 실측 성능의 차이가 매우 작다.

*Lin, Ji, et al. "Mcunet: Tiny deep learning on iot devices." *Advances in neural information processing systems* 33 (2020): 11711-11722.

탐색공간 정의

이번 과제에서는 MCUNet의 검색공간을 기반으로 하되, **네트워크 중간에 보조분류기(auxiliary classifier)를 배치할 수 있는 선택지까지 포함**한 확장된 탐색공간을 사용한다.

메인 네트워크의 탐색 변수는 다음과 같다.

- kernel size: 20개 블록 각각이 {3, 5, 7} 중 하나를 선택
- expand ratio: 20개 블록 각각이 {3, 4, 6} 중 하나를 선택
- depth 변화량: 6개의 레이어마다 {2, 3, 4} 중 하나를 선택

보조분류기는 *T-RECX 기반의 Pointwise Conv → Depthwise Conv → Fully-Connected 구조로 이루어지며, 각 레이어마다 다음을 선택한다.

- 활성 여부 {False, True}
- 임계치 8가지 {0.5, 1.0, 1.5, …, 4.0}
- kernel size {3, 5, 7}
- expand ratio {1.0, 1.5, 2.0}

결과적으로 탐색기는 메인 구조 × 보조분류기 구조 전체 조합을 평가하는 형태가 되며, 이는 조기 종료(Early Exit) 가능성까지 포함된 대규모 NAS 문제로 확장된다.

탐색공간 크기

메인 네트워크: 3^20 × 3^20 × 3^6 = 3^46 ≈ 8.9 × 10^21

보조분류기: 144^6 ≈ 8.9 × 10^12

전체 탐색공간: 3^46 × 144^6 ≈ 7.9 × 10^34

따라서 효율적인 탐색 알고리즘 없이는 탐색이 불가능하며, 탐색 전략의 성능이 최종 모델 품질에 직접적인 영향을 미친다.

*Ghanathe, Nikhil P., and Steve Wilton. "T-recx: Tiny-resource efficient convolutional neural networks with early-exit." *Proceedings of the 20th ACM International Conference on Computing Frontiers*. 2023.

탐색공간 정의

< 표1. 탐색공간 정의>

구성 요소	후보 값	설명
ks_list	{3, 5, 7}	메인 블록의 커널 크기
e_list	{3, 4, 6}	메인 블록의 expand ratio
d_list	{2, 3, 4}	레이어 안의 블록 개수
aux_a_list	{False, True}	보조분류기 활성 여부
aux_t_list	{0.5, 1.0, 1.5, …, 4.0}	임계치 후보
aux_k_list	{3, 5, 7}	보조분류기 depthwise 커널 크기
aux_e_list	{1.0, 1.5, 2.0}	보조분류기 pointwise expand ratio

탐색 목표

본 과제의 태스크는 **CIFAR-100 이미지 분류**이며, MCUNet에서 확장된 탐색공간(메인 네트워크 + 보조분류기) 내에서 주어진 하드웨어 제약을 만족하면서 가장 높은 정확도를 갖는 서브넷을 찾는 것을 목표로 한다.

타겟 하드웨어는 STM32H7 MCU(Flash 2MB, SRAM 1MB)이며, 탐색 과정에서 모든 후보 구조는 Flash, SRAM, 지연시간(latency)에 대한 제약을 동시에 만족해야 한다.

이를 위해 미리 훈련된 예측기가 제공된다고 가정한다.

- 자원 예측기: Flash, SRAM 사용량 추정
- 지연시간 예측기: 타겟 MCU에서의 latency 추정
- 정확도 예측기: 서브넷의 classification 성능 추정

탐색 알고리즘은 실제 학습 없이 예측기가 산출한 성능 지표만으로 평가되며, 본 과제에서는 동일한 예측기를 사용하여 **유전자 알고리즘(GA)**과 **강화학습(RL)** 두 방법의 성능을 비교한다.

유전자 알고리즘의 한계

1. 제약 활용 측면

유전자 알고리즘이 지연시간 제약을 끝까지 활용하지 못하고 비교적 안전한 여유 구간에서만 머무르는 경향이 있다.
예를 들어, 지연시간 제약조건이 2초인 설정에서 탐색된 서브넷의 지연시간은 대체로 1.6초이다.
즉, 정확도를 더 끌어올리기 위해 제약 상한을 공격적으로 활용하기보다는, 제약보다 다소 낮은 지연시간에서 안정적인 해를 찾는 방향으로 편향되는 것을 관찰할 수 있다.
이 관점에서 보면, 유전자 알고리즘은 주어진 자원 예산을 충분히 활용하지 못하는 서브넷을 탐색한다.

*표2: 요구 지연시간을 다르게 설정하여 10회씩 탐색을 수행했으며, 표는 각 설정에서 얻어진 서브넷의 **정확도와 지연시간의 평균과 표준편차**를 정리하였다.

< 표2. 요구 지연시간에 따른 탐색된 서브넷의 정확도 및 지연시간 >

지연시간 제약 (s)	평균 정확도	정확도 표준편차	평균 지연시간 (s)	지연시간 표준편차
2.0	0.7468	0.00159	1.6840	0.1538
1.8	0.7464	0.00131	1.5716	0.1401
1.6	0.7462	0.00205	1.4975	0.0702
1.4	0.7435	0.00131	1.3166	0.0718
1.2	0.7384	0.00124	1.1468	0.0308
1.0	0.7257	0.00757	0.9436	0.0383

유전자 알고리즘의 한계

2. 지연시간 예산 변경에 대한 재탐색 비용

유전자 알고리즘 기반 탐색은 지연시간 요구 조건을 바꿀 때마다 탐색을 다시 수행해야 한다.
실제 응용에서는 사용 환경이나 디바이스에 따라 지연시간 예산이 자주 바뀔 수 있는데, 이때마다 새로 탐색을 돌려야 하므로 검색 비용이 지연시간 설정 개수에 비례해 선형적으로 증가한다.

3. 확률적 탐색 방식이 가지는 구조적 제약

유전자 알고리즘은 확률적 탐색 절차에 기반하기 때문에, 어떤 구조가 선택되고 변이되는지가 명확한 규칙으로 기술되기 어렵고 탐색 과정 전반의 의사결정 흐름을 해석적으로 파악하기가 제한적이다.
이러한 특성은 구조 선택의 일관성을 떨어뜨릴 수 있으며, 탐색공간이 매우 큰 경우에는 동일 설정에서도 결과가 크게 달라지는 등 탐색 안정성과 재현성을 확보하기 어렵게 만드는 요인으로 작용한다.

과제 목표

MCUNet 기반 확장 탐색공간(메인 네트워크 구조 + 보조분류기 설정)에 대해, 기존 유전자 알고리즘(GA)이 갖는 탐색 한계를 보완할 수 있는 강화학습 기반 구조 탐색 방법을 설계하고 평가한다. 탐색 대상 모델은 CIFAR-100 분류를 수행하며, STM32H7 MCU(Flash 2MB, SRAM 1MB)를 타겟 하드웨어로 설정한다. 탐색 과정에서는 정확도·지연시간·자원 제약을 동시에 고려해야 하므로, 미리 학습된 정확도 예측기·지연시간 예측기·자원 예측기를 사용하여 각 서브넷 후보의 성능을 즉시 평가한다.

강화학습 접근법은 서브넷 구조를 순차적으로 결정하는 과정을 하나의 MDP로 정의하고, 각 서브넷의 성능을 기반으로 정책(policy)을 업데이트하여 제약을 만족하면서 정확도가 최대화되는 구조를 선택하는 방향으로 학습된다. 이를 위해 환경은 Gymnasium 스타일로 구성되며, 학습 알고리즘은 Stable-Baselines3의 PPO(Proximal Policy Optimization) 프레임워크를 기반으로 구현된다. 이 구조를 통해 정책이 대규모 탐색공간을 효율적으로 탐색하고, 연속된 의사결정 과정에서 점진적으로 더 나은 구조를 선택하도록 학습할 수 있다.

최종적으로 본 과제는 동일한 환경(예측기 기반 평가, 동일 제약 조건) 하에서 GA와 RL의 탐색 효율성, 안정성, 최종 성능을 정량적으로 비교하고, 보조분류기가 포함된 대규모 탐색공간에서 강화학습이 더 우수한 구조를 발굴할 수 있는지를 평가하는 것을 목표로 한다.

환경 정의

전체 개요

환경은 하나의 서브넷 구조를 순차적으로 설계하는 과정을 하나의 에피소드로 설계하였다. 에피소드가 시작되면 가장 먼저 **가능한 구조 중 가장 큰 서브넷을 초기 상태로 설정**하며, 이후 에이전트가 스텝마다 서브넷 구조를 하나씩 수정한다.
설계 의도는 큰 모델에서 시작해 제약을 만족하는 방향으로 점진적으로 축소·조정하는 탐색하는 것이며, 각 에피소드는 **고정된 길이(총 70 스텝)** 를 가지며, 모든 구성 요소의 선택이 끝나면 종료된다.

상태 (State)

현재까지 설계된 서브넷 구조와 예측된 성능 지표를 함께 포함하는 실값 벡터로 정의된다. 서브넷의 구조는 인코드를 통해 고정 길이의 원핫 벡터로 인코딩된다. 여기에 현재 아키텍처의 예측 정확도와 지연시간·Flash·SRAM이 목표 대비 얼마나 여유가 있는지를 나타내는 slack 값들, 그리고 의사결정 진행률을 추가하여 최종 상태 벡터를 구성한다. 따라서 에이전트는 “현재 구조가 어떤 형태인지”, “제약을 얼마나 여유 있게 만족하고 있는지”, “설계가 얼마나 진행되었는지”를 동시에 관측할 수 있도록 설계하였다.

행동 (Action)

행동은 탐색공간에 정의된 후보값 중 하나를 선택하는 이산값으로 구성된다. 환경 내부에는 미리 정의된 **의사결정 순서(총 70개의 결정 단위)** 가 존재하며, 각 스텝마다 어떤 요소(예: ks의 i번째 블록, aux_t의 j번째 레이어)를 수정할지 결정되어 있다. 에이전트는 해당 위치에 적용할 후보 인덱스를 선택하고, 환경은 이를 반영해 아키텍처를 갱신한다. 이 과정이 반복되면서, 초기의 가장 큰 서브넷이 단계적으로 축소되거나 재구성되며 최종 아키텍처가 완성된다.

환경 정의

보상 (Reward)

환경은 예측된 정확도·지연시간·자원 사용량을 기반으로 하나의 점수(score)를 산출하고, 에이전트는 **현재 점수와 이전 스텝 점수의 차이(현재 점수 – 이전 점수)** 를 보상으로 받는다. 이 방식은 에이전트가 구조를 변경할 때마다 모델 품질이 실제로 개선되었는지를 즉각 반영하게 해 주며, 점수를 증가시키는 행동만이 강화될 수 있도록 한다. 결과적으로 에이전트는 정확도를 유지하거나 높이면서 Flash·SRAM 한도를 초과하지 않고, 지연시간은 목표값에 근접하게 조정하는 방향으로 구조 선택 정책을 학습하게 된다.

점수 (Score)

점수는 세 가지 요소가 곱해진 형태로 구성된다.

먼저 정확도 항은 **accuracy_scale(α)**을 지수로 하는 거듭제곱 형태로 계산되며, 정확도의 작은 개선도 점수 증가에 크게 반영되도록 만든 항이다. 정밀도가 조금이라도 향상되면 그 변화가 명확하게 점수에 드러나도록 설계한 부분이다.

Flash·SRAM은 **하드 제약 (Hard Violation, β)**로 처리되며, 목표 대비 초과 비율을 기반으로 하드 제약 항이 계산된다. 하드 제약이 커질수록 지수 기반 감쇠가 강하게 적용되어 점수가 급격히 감소한다. 이는 메모리 한도를 넘는 구조가 학습 과정에서 자연스럽게 배제되도록 하기 위한 장치이다.

지연시간은 **소프트 제약 요소(Soft Violation, γ)** 로 분리하여 초과 시 감쇠가 적용되지만, Flash·SRAM만큼 급격하게 감소하지는 않는다. 지연시간을 지나치게 줄이면 모델이 과도하게 작아져 정확도가 떨어질 수 있기 때문에, 지연시간을 너무 작게 만드는 방향으로만 학습이 치우치지 않도록 완만한 감쇠로 구현된 부분이다. 이로써 정확도 확보와 지연시간 관리 간의 균형을 자연스럽게 유지할 수 있다.

$$\text{score} = \text{accuracy}^{\alpha} \cdot \exp\big(-\beta\,(\text{flash\_over} + \text{sram\_over})\big) \cdot \exp\big(-\gamma\,\text{lat\_over}\big)$$

강화학습 알고리즘 - PPO

에이전트는 정책 기반 강화학습 알고리즘인 **PPO(Proximal Policy Optimization)** 로 학습된다. PPO는 정책을 크게 바꾸지 않고 안정적으로 개선하도록 설계된 강화학습 방법이며, 비교적 단순한 구조로도 안정적인 수렴 특성을 보여 다양한 연속·이산 의사결정 문제에서 널리 사용된다. 이러한 특성 때문에 복잡한 NAS 탐색공간을 모델링하는 정책을 업데이트할 수 있으며, 환경의 잡음에도 비교적 견고하게 작동할 것으로 기대되어 본 연구에서 채택하였다.

에이전트는 다음과 같은 방식으로 동작한다.

1. 상태 벡터를 입력받아 현재 스텝에서 선택할 후보 인덱스를 확률적으로 출력한다.
2. 환경이 그 선택을 반영해 아키텍처를 갱신하고, 예측된 성능 변화에 따라 보상이 부여된다.
3. 여러 에피소드에서 수집된 상대적 성능 차이를 바탕으로 정책을 갱신한다.
4. 학습이 반복될수록, 자원 제약을 만족하면서 정확도가 높은 구조를 선택하는 행동을 더 자주 시도하게 된다.

PPO는 정책은 단순히 매 스텝을 독립적으로 최적화하는 것이 아니라, 처음 상태부터 최종 상태까지의 연관성을 학습한다는 점에서 해결하고자 하는 문제에 적합한 방법이라고 판단하였다.

PPO 하이퍼파라미터

- **γ (discount factor)**
  장기 보상을 얼마나 중시하는지 결정한다.
  구조 선택이 연속적으로 영향을 주는 본 문제에서는 γ를 높게 설정해, 후반부 정확도 및 제약 만족을 높이는 행동을 선호하도록 한다.
- **GAE λ (Generalized Advantage Estimation)**
  보상 추정의 분산과 편향을 조절한다.
  λ를 적절히 설정하면 탐색이 지나치게 요동치지 않으면서도 보상 신호를 안정적으로 전달할 수 있다.
- **clip range (policy update 제한 폭)**
  한 번의 업데이트에서 정책이 너무 급격하게 바뀌는 것을 방지한다.
  구조 탐색 문제는 보상 스케일이 민감하기 때문에, 안정성을 유지하는 데 필수적인 요소이다.
- **entropy coefficient**
  행동 다양성을 유지하도록 강제로 exploration을 부여한다.
  탐색공간이 매우 크고 조기 수렴을 피하고 싶은 본 과제에서는 적당한 엔트로피 보너스가 탐색 성능을 크게 좌우한다.
- **rollout length (n_steps)**
  정책을 한 번 업데이트하기 위해 얼마나 많은 스텝을 모을지 결정한다.
  에피소드 길이가 고정되어 있으므로, rollout을 어떻게 설정하느냐가 학습 안정성·샘플 효율에 큰 영향을 준다.
- **batch size, n_epochs**
  정책을 얼마나 정교하게 업데이트할지를 결정하며, 지나치게 크거나 작으면 정책이 불안정해질 수 있다.
- **learning rate**
  최적화의 수렴 속도를 결정하며, 보상 구조가 강하게 제약되어 있기 때문에 과도한 학습률은 폭주를 유발할 수 있다.

실험 환경

1. 소프트웨어 환경
    a. 도커 기반 실행 환경 사용
        - 베이스 이미지 : PyTorch 공식 torch-cuda12.1 이미지
    b. Python 실행 환경
        - Python 3.10
2. 하드웨어 환경
    a. GPU
        - NVIDIA TITAN V
        - 메모리 : 12GB
    b. CPU
        - AMD Ryzen 5 5600

실험 1. PPO 하이퍼파리미터 튜닝

본 환경의 PPO 성능을 좌우하는 핵심 요소가 **정책 업데이트 폭**과 **탐색 분포의 폭**이기 때문이다. NAS 구조 탐색은 행동 공간이 크고 한 번의 결정이 전체 구조에 연쇄적으로 영향을 주기 때문에, PPO가 안정적으로 학습하려면 다음 세 축이 특히 중요하게 작동한다.

- **learning_rate**
  정책이 구조 공간에서 얼마나 빠르게 이동할 수 있는지를 결정하며,
  lr이 너무 작으면 제약을 만족하는 영역까지 도달하지 못하고 지나치게 느리게 수렴하는 문제가 발생한다.
- **clip_range**
  정책 업데이트의 허용 폭을 직접 제어하는 항목이다.
  NAS처럼 구조 변화가 크게 필요한 문제에서는 clip_range가 너무 작으면
  정책이 기존 구조 근처에서만 머물며 유효한 subnet을 찾지 못하게 된다.
- **ent_coef**
  탐색(exploration)의 폭을 결정하는 요소로,
  초기 학습 단계에서 다양한 구조를 시도하도록 하여
  제약을 만족하는 영역을 충분히 탐색할 수 있게 한다.

따라서 이 세 가지를 중심으로 그리드를 구성하면 “정책 업데이트 크기”와 “탐색 다양성” 두 가지 축을 모두 조절할 수 있어, PPO가 NAS 환경에서 실제로 수렴 가능한 조건을 빠르게 확인할 수 있다.

평가지표는 PPO가
1) 제약을 얼마나 잘 지키는지 (any_violation_rate),
2) 제약 내에서 얼마나 좋은 평균·최고 정확도를 내는지 (valid_mean_accuracy, valid_max_accuracy),
3) reward 설계가 정책 학습을 제대로 유도하고 있는지(mean_total_reward) 를 평가한다.

실험 1. PPO 하이퍼파리미터 튜닝

탐색 그리드

- learning_rate: {3e-4, 1e-4}
- ent_coef: {0.01, 0.03}
- clip_range: {0.1, 0.2}

설정별 수행 절차

- 약 50k timesteps의 짧은 학습
- 여러 target_latency [1.0, 1.2, …, 2.0]에 대해 episode 평가

평가 지표

- any_violation_rate
- valid_mean_accuracy
- valid_max_accuracy
- mean_total_reward

실험 1. PPO 하이퍼파리미터 튜닝 결과

< 표3. 실험 1 결과>

lr	ent	clip	mean_reward	any_violate	valid_mean_acc	valid_max_acc
0.0003	0.01	0.2	0.192	0.0	0.719	0.727
0.0003	0.03	0.2	0.192	0.0	0.719	0.727
0.0003	0.03	0.1	0.074	1.0	0.0	0.0
0.0001	0.01	0.1	0.009	1.0	0.0	0.0
0.0001	0.01	0.2	0.009	1.0	0.0	0.0
0.0003	0.01	0.1	0.0	1.0	0.0	0.0
0.0001	0.03	0.1	0.0	1.0	0.0	0.0
0.0001	0.03	0.2	0.0	1.0	0.0	0.0

실험 1. PPO 하이퍼파리미터 튜닝 결론

Learning Rate (lr)

- lr=3e-4 조합만 제약을 만족하는 subnet을 발견하였다.
- lr=1e-4는 동일한 timesteps(50k)에서는 학습 폭이 너무 작아, 정책이 constraint 영역까지 이동하지 못했다.
  → NAS 환경에서는 초기 단계에서도 정책 업데이트가 충분히 크게 일어나야 탐색이 의미 있는 방향으로 진행됨을 시사한다.

Clip Range

- clip=0.2만 성공적으로 제약을 만족.
- clip=0.1은 정책 변화 폭이 지나치게 제한되어, 구조를 크게 바꿔야 하는 NAS 탐색 특성과 부적합했다.
  → NAS처럼 high-dimensional discrete search에서는 clip range를 너무 좁히면 구조 변형 자체가 불가능해질 수 있음을 보여준다.

Entropy Coefficient

- ent_coef=0.01과 0.03은 실험 1에서 거의 동일한 성능을 보였다.
  → 이미 exploration이 충분히 확보된 상황에서는 entropy가 결정적 요인이 되지 않았다.

따라서 학습률은 3e-4, clip은 0.2, entropy는 0.01로 고정하여 reward 설계를 평가할 수 있도록 설정을 통일하였다.

실험 2: Reward 함수 실험 설계

실험 2에서는 **Reward 설계가 탐색 행동에 어떤 영향을 주는지**를 분리해 평가하였다. NAS 환경은 정확도와 지연시간·메모리 두 가지 제약이 동시에 존재하므로, Reward 함수의 설계 방식이 탐색 경로를 크게 바꾸는 요소가 된다. Reward는 세 가지 축(정확도 강화, 하드 제약, 소프트 제약)으로 구성되며, 이 세 항의 상대적 비중을 조절하는 것이 실험 2의 핵심 실험 목표이다.

Hard_violation, β (Flash/SRAM 제약 페널티)

Flash·SRAM은 초과 시 모델이 실제 MCU에서 구동될 수 없기 때문에, 이 항은 실제로 배포 가능한 아키텍처를 찾도록 한다. 값이 클수록 메모리 초과가 강하게 억제되고, 값이 낮으면 정확도 향상에 밀려 메모리 제약이 무시되는 경향이 생긴다.
따라서 hard_violation 조정은 “제약을 얼마나 엄격하게 적용할지”를 결정하는 실험이다.

Soft_violation, γ (지연시간 제약 페널티)

지연시간은 정확도와 trade-off 관계에 있어, 너무 빠르게 만들면 정확도가 떨어지고 너무 느리면 실용성이 떨어진다. soft_violation은 이를 완만한 제약으로 두어, 모델이 지연시간 예산을 적절히 사용하면서 정확도도 확보하도록 균형을 잡는 파라미터이다.
soft_violation 조정은 “지연시간을 얼마나 민감하게 다룰지”를 평가하는 실험이다.

Accuracy_scale, α (정확도 항 강조 정도)

정확도 항의 지수승 α는 reward에서 정확도의 비중을 결정하는 가장 직관적 파라미터다. α가 커질수록 정확도 향상에 대한 보상이 상대적으로 과대해져, 제약을 일부 희생하더라도 정확도를 더 높이려는 경향이 나타날 수 있다.
accuracy_scale 조정은 “정확도와 제약 간의 우선순위 조정”을 시험하는 실험이다.

실험 2: Reward 함수 실험 설계

탐색 그리드

- hard_violation: {3.0, 5.0}
- soft_violation: {0.5, 1.0}
- accuracy_scale: {4.0, 6.0}

평가 지표

- 실험 1과 동일

실험 2: Reward 함수 실험 결과

< 표4. 실험 2 결과>

hard violation	soft violation	accuracy scale(alpha)	mean_reward	any_violate	valid_mean_acc	valid_max_acc
3.0	1.0	4.0	0.265	0.0	0.718	0.725
3.0	0.5	6.0	0.137	0.0	0.718	0.725
5.0	1.0	4.0	0.265	0.0	0.718	0.725
5.0	0.5	4.0	0.265	0.0	0.718	0.725
3.0	0.5	4.0	0.265	0.0	0.718	0.725
5.0	1.0	6.0	0.137	0.0	0.718	0.725
5.0	0.5	6.0	0.137	0.0	0.718	0.725
3.0	1.0	6.0	0.0	1.0	0.0	0.0

실험 2: Reward 함수 실험 결론

Accuracy Scale (α)

- α = 4.0에서는 모든 β/γ 조합이 제약을 안정적으로 만족하였다.
  → 정확도 항이 과도하게 우세해지지 않아, 제약·정확도 간 균형적 탐색이 유지됨.

- α = 6.0에서는 hard_violation이 충분히 크지 않은 경우 Flash 초과가 빈번하게 발생하였다.
  → accuracy 항의 비중이 일정 수준을 넘어서면, 정책이 '정확도 최대화'로 수렴하여 제약 기반 NAS 설정과 충돌함.

Hard Violation (β)

- β = 5.0은 α = 6.0에서도 Flash 위반을 효과적으로 억제하였다.
  → 높은 β는 Flash/SRAM 제약의 절대적 우선순위를 보장하며, 정확도 강조 상황에서도 제약 유지가 가능해짐.

- β = 3.0은 α = 6.0 조합 중 일부에서 Flash 위반률 100%를 기록하였다.
  → hard_violation의 크기는 NAS에서 실제 제약을 강제하는 핵심 파라미터임을 확인.

Soft Violation (γ)

- 지연시간 제약은 전체 탐색에서 거의 모든 episode가 안정적으로 만족되었으며, γ 값 변화는 성능에 유의한 영향을 주지 않았다.
  → 현재 search space와 target_latency 조건에서는 latency 제약이 상대적으로 느슨하고, 실제 탐색의 병목은 Flash/SRAM임을 명확히 확인.

α=4.0 / β=3.0 또는 5.0 / γ는 안정적 범위 내에서 일관적이고 제약을 준수하는 탐색을 제공한다.

강화학습과 유전자 알고리즘 비교 결과

< 표5. 유전자 알고리즘과 강화학습 결과 비교>

Target latency	GA mean acc	RL mean acc	GA mean lat	RL mean lat
2.0	0.7468	0.7213	1.6840	0.6908
1.8	0.7464	0.7184	1.5716	0.7023
1.6	0.7462	0.7215	1.4975	0.7034
1.4	0.7435	0.7155	1.3166	0.7353
1.2	0.7384	0.7194	1.1468	0.7178
1.0	0.7257	0.7153	0.9436	0.7097

강화학습과 유전자 알고리즘 비교

이번 실험에서 동일한 예측기를 사용하여 GA와 PPO를 비교한 결과, PPO 기반 NAS는 주어진 탐색 환경에서 구조를 제대로 학습하지 못하고, GA 대비 **모든 지표에서 명확하게 열등한 성능**을 보였다.

1. 정확도: 모든 조건에서 GA가 우세

6개 target latency 전 구간에서 GA mean accuracy > RL mean accuracy. PPO는 단 한 번도 GA를 이긴 지점이 없었으며, 정확도 차이는 최대 2.8%p 수준으로 반복적으로 나타났다. → PPO는 정확도 신호를 정책 업데이트에 반영하지 못하고 collapse된 상태로 수렴한다.

2. 지연시간: PPO는 목표를 맞추지 못하고 과도하게 축소

GA는 target latency가 감소할수록 latency도 자연스럽게 감소해 목표 제약선을 따라가는 정상적인 NAS 행동을 보이지만, 반면 PPO는 target이 달라져도 지연시간이 0.69~0.73s 범위로 거의 고정된다.

- target latency 2.0에서도 0.69s
- target latency 1.0에서도 0.70s

PPO는 지연시간 제약 조절 능력을 완전히 상실했고, 모델을 무조건 작게 만드는 정책으로 학습되었다..

3. trade-off 탐색 능력 자체가 동작하지 않음

정확도가 낮고 지연시간이 낮은 결과는 정확도–지연시간 균형점을 찾으려는 NAS의 본질적 목적과 정반대이다. PPO는 reward 구조를 활용하지 못하고 단순한 subnet 축소기처럼 동작한다.

최종 결론

1. Reward 구조가 gradient 기반 업데이트에 매우 불리함

환경의 보상은 현재 스코어와 이전 스코어의 차로 설계되어 있다. 이 방식은 score가 조금이라도 나아지면 강화"라는 의도였지만, 실제로는 다음과 같은 문제가 발생했다.

- latency/Flash/SRAM 위반이 초반 구조 수정에서 쉽게 발생하여 중간 reward가 거의 항상 0 혹은 음수로 수렴
- accuracy는 episode 후반에만 반영되기 때문에 대부분의 step에서 reward가 정보성을 잃음
- accuracy는 episode 후반부에서만 값이 변하므로, 전체 70 step 중 상당 부분은 **정책 개선에 기여하지 않는 reward**만 발생하여 정책이 방향성을 학습하지 못한 채 분산된 상태로 유지

GA는 이러한 reward sparsity에 영향을 거의 받지 않기 때문에, 동일한 예측기 환경에서도 PPO가 더 불리했다.

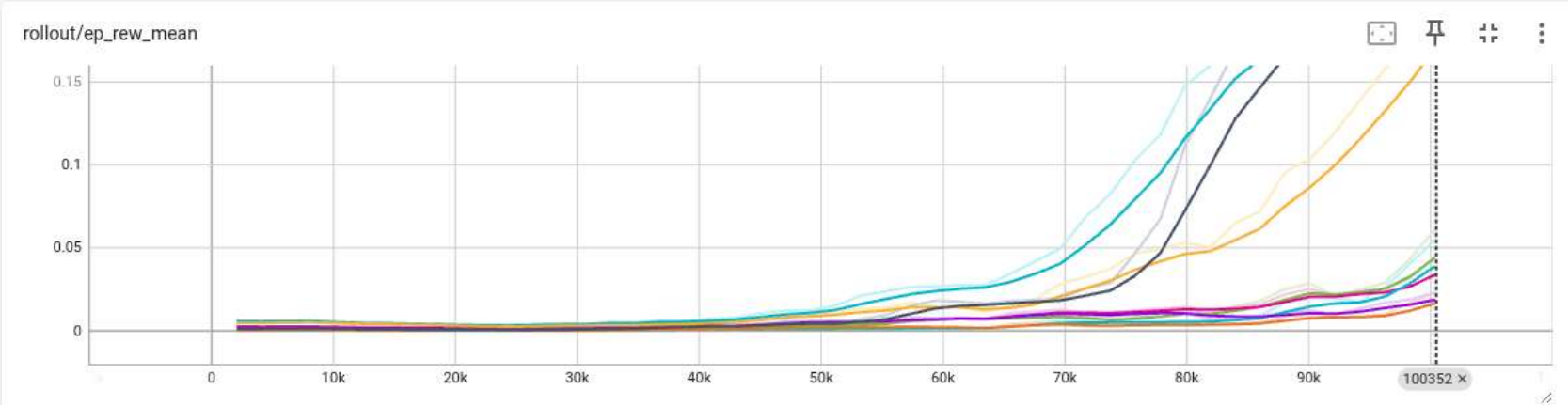
2. PPO 알고리즘이 NAS에서 갖는 구조적 한계

PPO는 안정적인 정책경사 계열 알고리즘이지만, NAS처럼 조합 공간이 거대하고 의사결정 길이가 긴 문제에서는 구조적 제약이 뚜렷하게 드러난다.

- PPO는 clipped objective 때문에 정책 업데이트가 기존 정책 주변(local neighborhood)에서만 일어나며, 대규모 구조 변화를 요구하는 NAS 탐색과 잘 맞지 않는다.
- 정책이 특정 방향으로 일찍 수렴하면 clip_range로 인해 정책을 크게 되돌리기 어려워 탐색 다양성이 급격히 떨어진다.
- accuracy·latency·자원 predictors는 noise를 포함하며, 이 노이즈는 gradient 기반 PPO 업데이트를 더욱 불안정하게 만든다.

따라서, 반면 GA는 여러 개체를 동시에 평가하고 noise에 강하기 때문에 이번 실험과 같은 NAS 설정에서는 구조적으로 우위를 가진다.

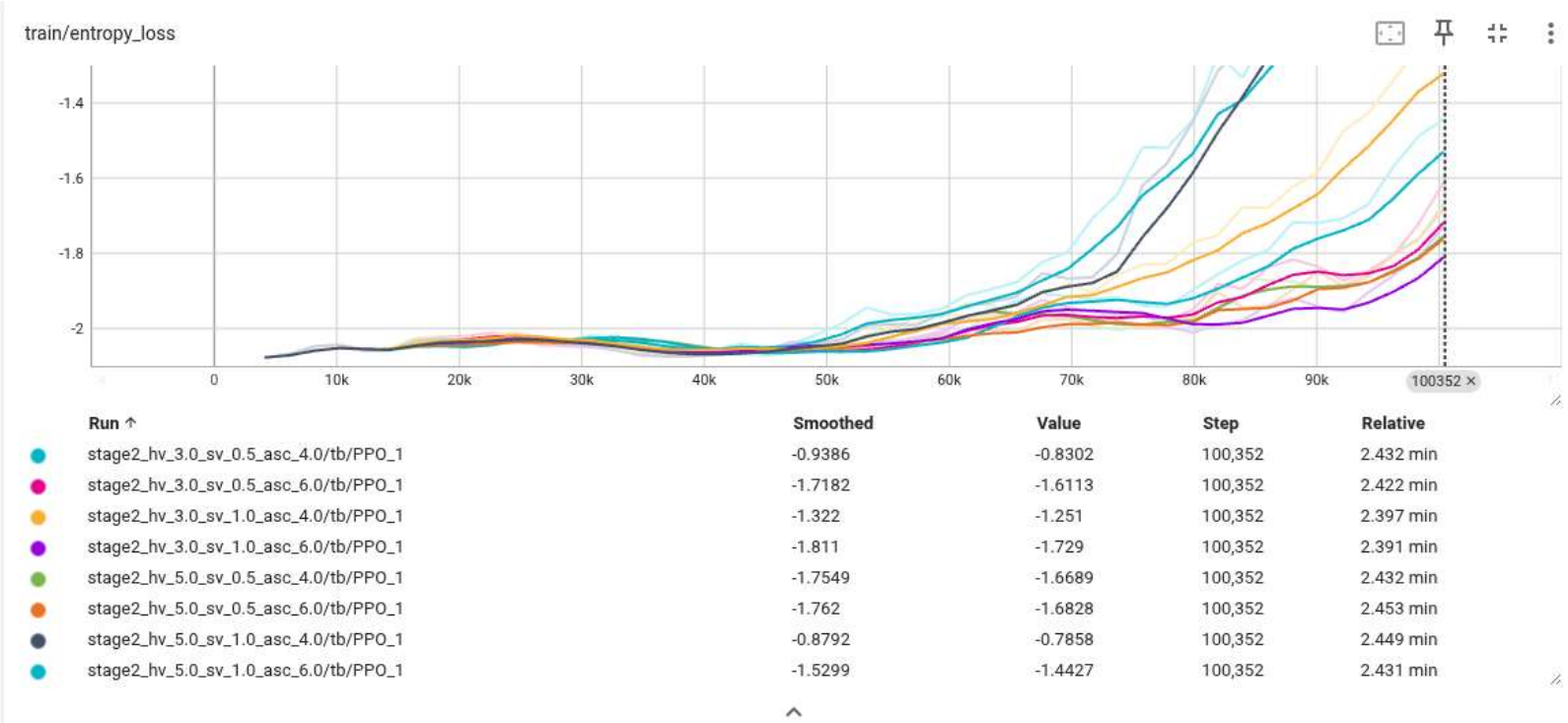
실험 2 학습 로그 분석



Run	Smoothed	Value	Step	Relative
stage2_hv_3.0_sv_0.5_asc_4.0/tb/PPO_1	0.229	0.2392	100,352	2.484 min
stage2_hv_3.0_sv_0.5_asc_6.0/tb/PPO_1	0.0338	0.0444	100,352	2.473 min
stage2_hv_3.0_sv_1.0_asc_4.0/tb/PPO_1	0.1701	0.2002	100,352	2.446 min
stage2_hv_3.0_sv_1.0_asc_6.0/tb/PPO_1	0.0187	0.023	100,352	2.442 min
stage2_hv_5.0_sv_0.5_asc_4.0/tb/PPO_1	0.044	0.0612	100,352	2.484 min
stage2_hv_5.0_sv_0.5_asc_6.0/tb/PPO_1	0.0164	0.0229	100,352	2.504 min
stage2_hv_5.0_sv_1.0_asc_4.0/tb/PPO_1	0.228	0.239	100,352	2.501 min
stage2_hv_5.0_sv_1.0_asc_6.0/tb/PPO_1	0.039	0.0546	100,352	2.483 min

< 그래프 1. 실험 2 엔트로피 로스 >

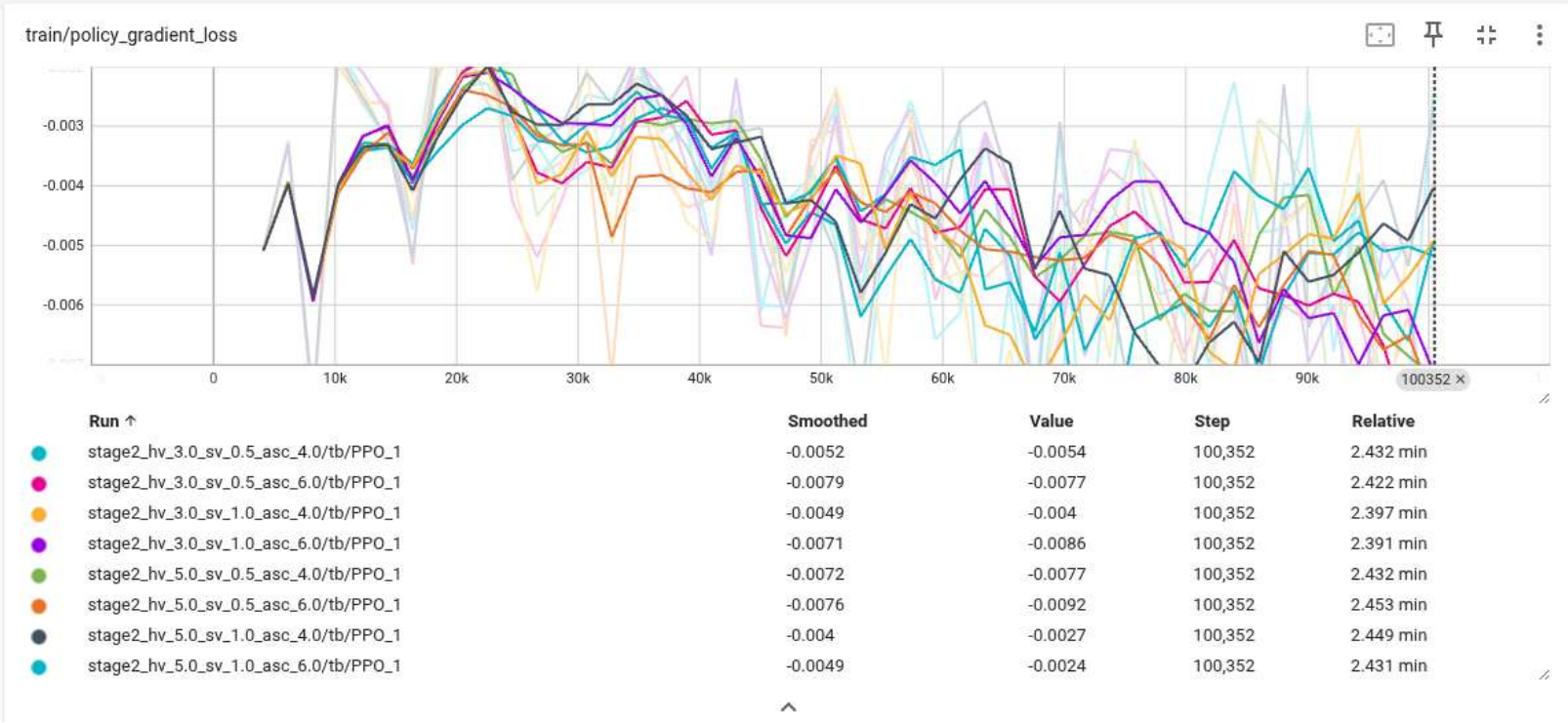
실험 2 학습 로그 분석



Run	Smoothed	Value	Step	Relative
stage2_hv_3.0_sv_0.5_asc_4.0/tb/PPO_1	-0.9386	-0.8302	100,352	2.432 min
stage2_hv_3.0_sv_0.5_asc_6.0/tb/PPO_1	-1.7182	-1.6113	100,352	2.422 min
stage2_hv_3.0_sv_1.0_asc_4.0/tb/PPO_1	-1.322	-1.251	100,352	2.397 min
stage2_hv_3.0_sv_1.0_asc_6.0/tb/PPO_1	-1.811	-1.729	100,352	2.391 min
stage2_hv_5.0_sv_0.5_asc_4.0/tb/PPO_1	-1.7549	-1.6689	100,352	2.432 min
stage2_hv_5.0_sv_0.5_asc_6.0/tb/PPO_1	-1.762	-1.6828	100,352	2.453 min
stage2_hv_5.0_sv_1.0_asc_4.0/tb/PPO_1	-0.8792	-0.7858	100,352	2.449 min
stage2_hv_5.0_sv_1.0_asc_6.0/tb/PPO_1	-1.5299	-1.4427	100,352	2.431 min

< 그래프 2. 실험 2 엔트로피 로스 >

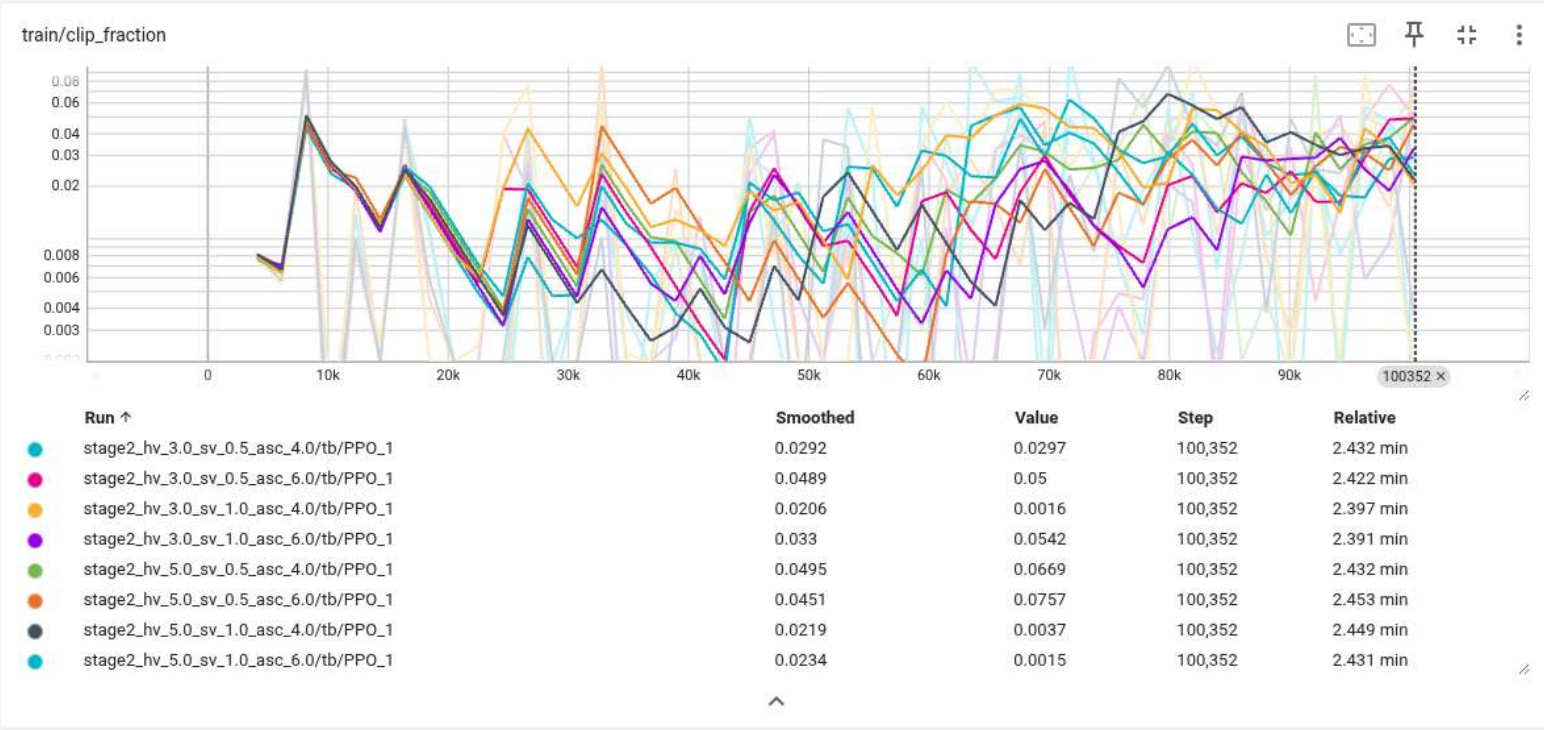
실험 2 학습 로그 분석



Run	Smoothed	Value	Step	Relative
stage2_hv_3.0_sv_0.5_asc_4.0/tb/PPO_1	-0.0052	-0.0054	100,352	2.432 min
stage2_hv_3.0_sv_0.5_asc_6.0/tb/PPO_1	-0.0079	-0.0077	100,352	2.422 min
stage2_hv_3.0_sv_1.0_asc_4.0/tb/PPO_1	-0.0049	-0.004	100,352	2.397 min
stage2_hv_3.0_sv_1.0_asc_6.0/tb/PPO_1	-0.0071	-0.0086	100,352	2.391 min
stage2_hv_5.0_sv_0.5_asc_4.0/tb/PPO_1	-0.0072	-0.0077	100,352	2.432 min
stage2_hv_5.0_sv_0.5_asc_6.0/tb/PPO_1	-0.0076	-0.0092	100,352	2.453 min
stage2_hv_5.0_sv_1.0_asc_4.0/tb/PPO_1	-0.004	-0.0027	100,352	2.449 min
stage2_hv_5.0_sv_1.0_asc_6.0/tb/PPO_1	-0.0049	-0.0024	100,352	2.431 min

< 그래프 3. 실험 2 policy gradient >

실험 2 학습 로그 분석



Run	Smoothed	Value	Step	Relative
stage2_hv_3.0_sv_0.5_asc_4.0/tb/PPO_1	0.0292	0.0297	100,352	2.432 min
stage2_hv_3.0_sv_0.5_asc_6.0/tb/PPO_1	0.0489	0.05	100,352	2.422 min
stage2_hv_3.0_sv_1.0_asc_4.0/tb/PPO_1	0.0206	0.0016	100,352	2.397 min
stage2_hv_3.0_sv_1.0_asc_6.0/tb/PPO_1	0.033	0.0542	100,352	2.391 min
stage2_hv_5.0_sv_0.5_asc_4.0/tb/PPO_1	0.0495	0.0669	100,352	2.432 min
stage2_hv_5.0_sv_0.5_asc_6.0/tb/PPO_1	0.0451	0.0757	100,352	2.453 min
stage2_hv_5.0_sv_1.0_asc_4.0/tb/PPO_1	0.0219	0.0037	100,352	2.449 min
stage2_hv_5.0_sv_1.0_asc_6.0/tb/PPO_1	0.0234	0.0015	100,352	2.431 min

< 그래프 4. 실험 2 clip fraction >

참고: 실험 2 학습 로그 분석

1. Reward 증가

- ep_rew_mean은 증가했지만 accuracy는 거의 개선되지 않음
- 정책은 정확도를 개선하지 않고, 구조를 과도하게 축소하는 방향으로만 reward를 확보함
- **reward 함수가 의도한 학습 신호로 작동하지 않음**

2. Entropy 증가

- entropy_loss가 단계 후반으로 갈수록 꾸준히 증가
- PPO가 일반적으로 보이는 entropy 감소 패턴과 정반대
- 정책이 한 방향으로 수렴하지 않고 점점 무작위화되는 현상

3. Policy Gradient 불안정

- policy_gradient_loss가 전 구간에서 큰 변동성 유지
- gradient 방향성이 안정적으로 형성되지 못함

4. PPO 클리핑 작동 거의 없음

- clip_fraction이 지속적으로 매우 낮고 패턴이 일관되지 않음
- 정책이 의미 있는 업데이트를 거의 하지 못함
- PPO의 안정화 메커니즘(clip)의 역할이 작동할 정도의 학습이 일어나지 않음

참고 : 예측기

예측기 구성

- Encoder 출력 벡터를 입력으로 받는 3-layer MLP, hidden size 512.
- Accuracy / Latency / Flash / SRAM 각각 독립적으로 학습된 회귀 모델.
- RL·GA 모두 동일한 예측기를 사용하여 공정한 비교 조건 유지.

모든 예측기가 높은 R2(>0.91), 높은 Pearson·Spearman(>0.92) 을 기록해 서브넷 성능 추정용으로 신뢰 가능한 수준이다.

RL과 GA는 동일한 예측기를 사용하기 때문에 비교 시 predictor bias가 두 알고리즘을 차별하지 않으며,
Flash 예측이 상대적으로 어렵지만 constraint 판단에는 문제 없는 수준.

< 표6. 예측기 오차>

구분	Loss	MAE	RMSE	R2	Pearson	Spearman
정확도	0.0057	0.0057	0.0091	0.9921	0.9961	0.9894
지연시간	0.0264	0.0264	0.0341	0.9917	0.9959	0.9944
Flash	0.0894	0.0894	0.1120	0.9128	0.9571	0.9275
SRAM	0.0014	0.0014	0.0021	0.9999	1.0000	0.8820