# 도로의 신사,
# 신속하고, 정확한 일처리

## F1:1024 모델

: Plasticity 조절, 도로 쉽게 벗어나지 않는 Contrained RL, Vision Transformer를 곁들인.

팀: F1:1024
발표자: 강민구, 전형준

서울시립대학교
UNIVERSITY OF SEOUL

# Contents

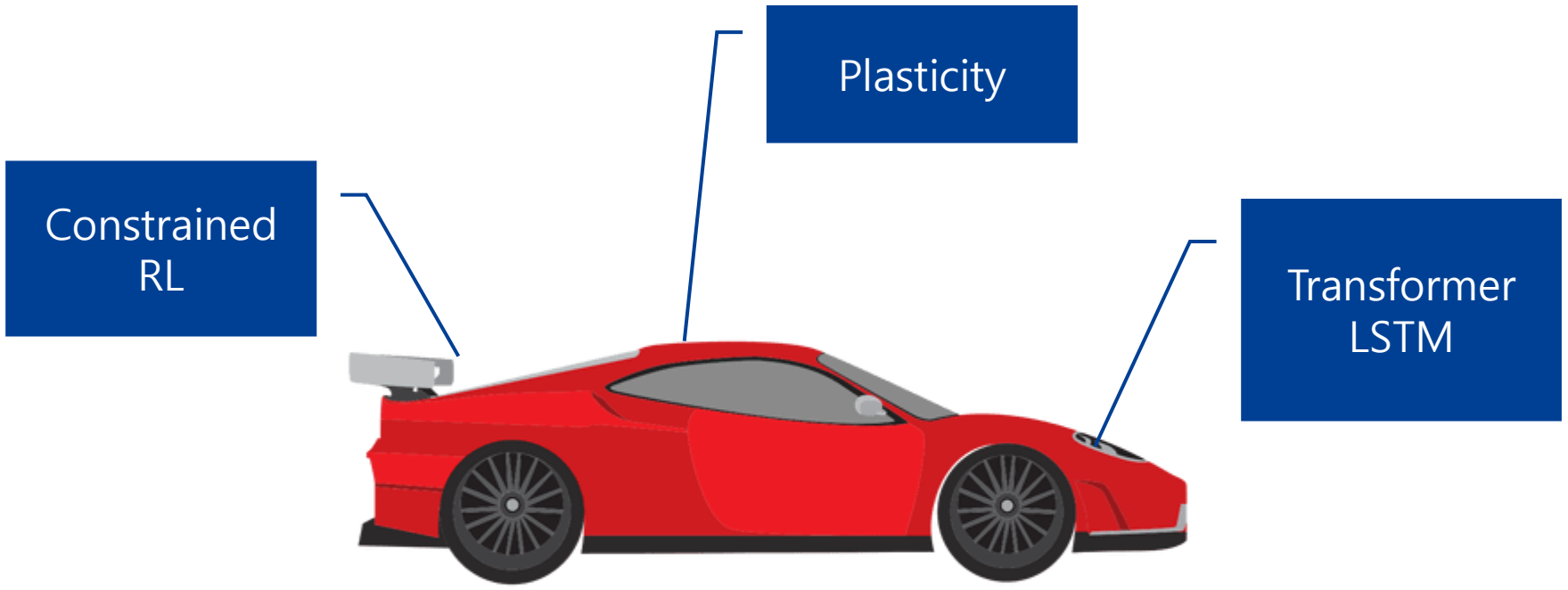


서울시립대학교
UNIVERSITY OF SEOUL

# 1) 개요

서울시립대학교 UNIVERSITY OF SEOUL

본 프로젝트는
안전하며, 다양한 상황에 대처하는 레이싱카를 만드는 과정에 의의를 뒀습니다.



# 2) 적응 능력 키우기: Plasticity 조절 및 성능

서울시립대학교 UNIVERSITY OF SEOUL

가소성(Plasticity)이란? 에이전트가 환경과 상호작용하며 정책/행동을 변화시키고 적응하는 능력
모델은 학습하면서 특정 환경에 특화되기 때문에, 적응 능력인 가소성을 잃게 됨.

참고논문: 'Sample-Efficient Multiagent Reinforcement Learning with Reset Replay' (ICML 2024)

$$P(\theta_t) = b - \mathbb{E}_{l\sim L}[l(\theta_t^*)], \quad \text{where} \quad \theta_t^* = \mathrm{OPT}(\theta_t, l)$$

특정 파라미터 상태에서의 네트워크 Plasticity를 P(theta).
Baseline b와 최적화된 파라미터의 기대 최종 Loss의 차이로 Plasticity를 계산.

$$P(\theta_{t=K}) - P(\theta_{t=0})$$

학습 초기의 가소성에 K번째 학습 후의 가소성 차이를 계산하면, 발생한 가소성 Loss를 계산할 수 있음.
가소성이 일정하게 유지된다면, 모델은 새로운 환경에 대한 적응 능력을 잃지 않았음을 의미.
논문은 Sample Efficient를 위해 Replay Ratio를 올리면서도 Plasticity Loss를 방지하는 방안을 제시함.

# 2) 적응 능력 키우기: Plasticity 조절 및 성능

서울시립대학교 UNIVERSITY OF SEOUL

## 도입한 Method

(1) Single Agent가 병렬 환경에서 데이터를 다양하게 수집. (Replay buffer)
(2) Replay Ratio, Tu(훈련 주기), Tc(타겟 네트워크 업데이트 주기), Tr(Shrink & Perturb 주기) 파라미터 설정
(3) 훈련을 할 때 Replay ratio를 높여서 진행. Tr 주기마다 Shrink & Perturb 진행.

참고논문: 'Sample-Efficient Multiagent Reinforcement Learning with Reset Replay' (ICML 2024)

**(1) Agent's policy parameters**

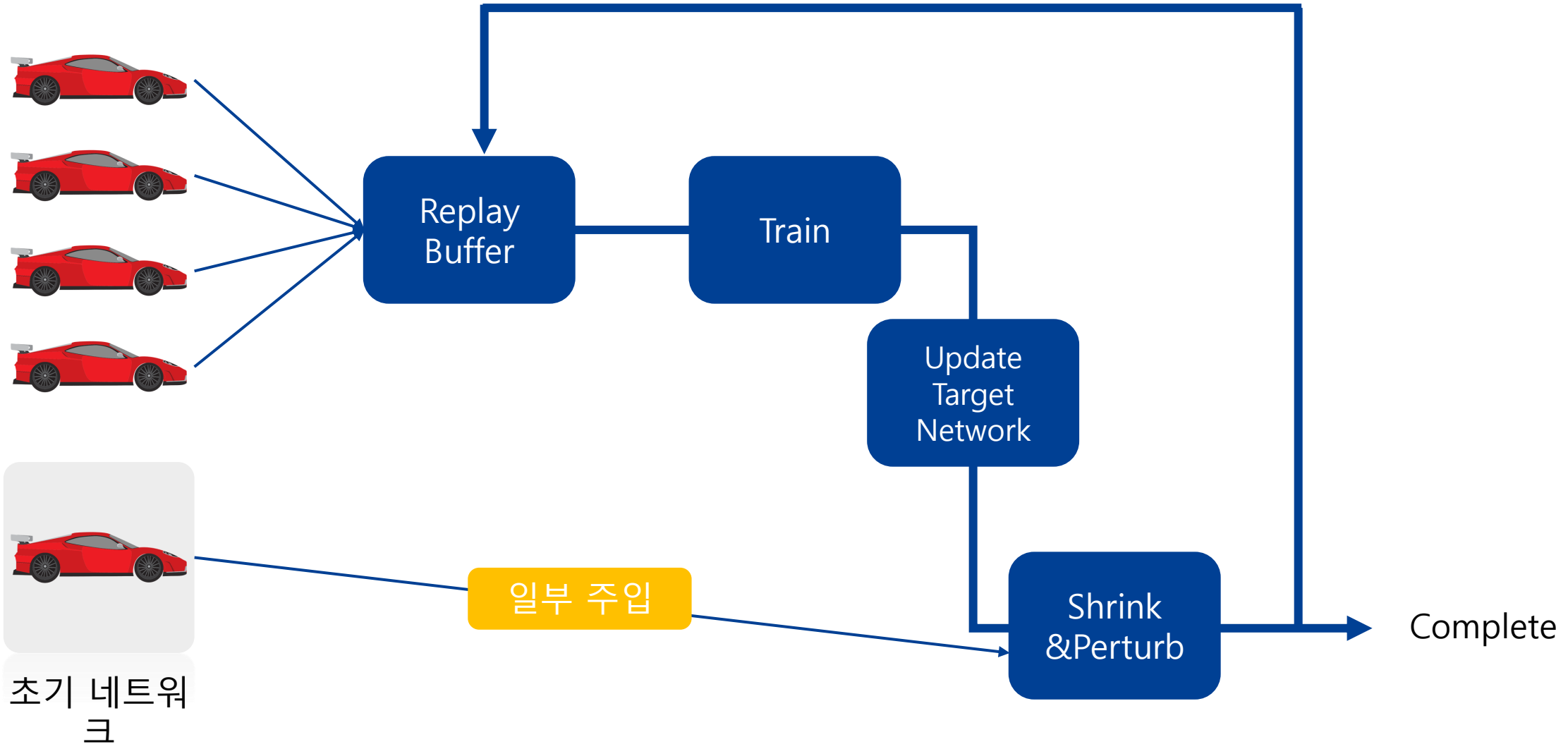
$$\theta_i^t \leftarrow \alpha\theta_i^t + (1-\alpha)\theta_i^0, \quad \text{for } i = 1, 2, \ldots, N$$

**(2) Centralized critic network parameters**

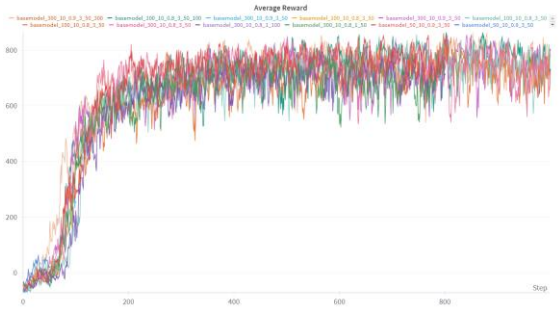
$$\phi^t \leftarrow \alpha\phi^t + (1-\alpha)\phi^0$$

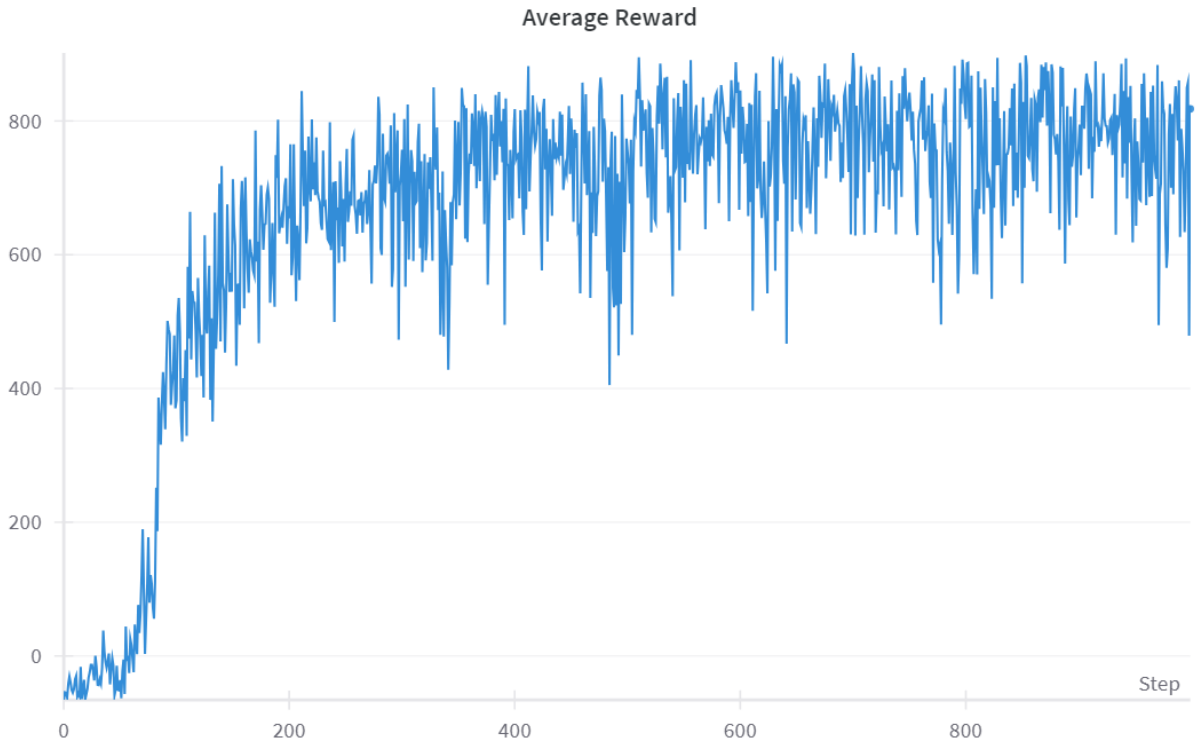
초기 네트워크의 일부를 주입하는 방식

# 2) 적응 능력 키우기: Plasticity 조절 및 성능



# 2) 적응 능력 키우기: Plasticity 조절 및 성능




**Config parameters:** {} 9 keys

alpha: 0.8

env_count: 4

episodes: 1,000

map_reset_interval: 100

max_timestep: 900

replay_ratio: 10

TC: 5

TR: 50

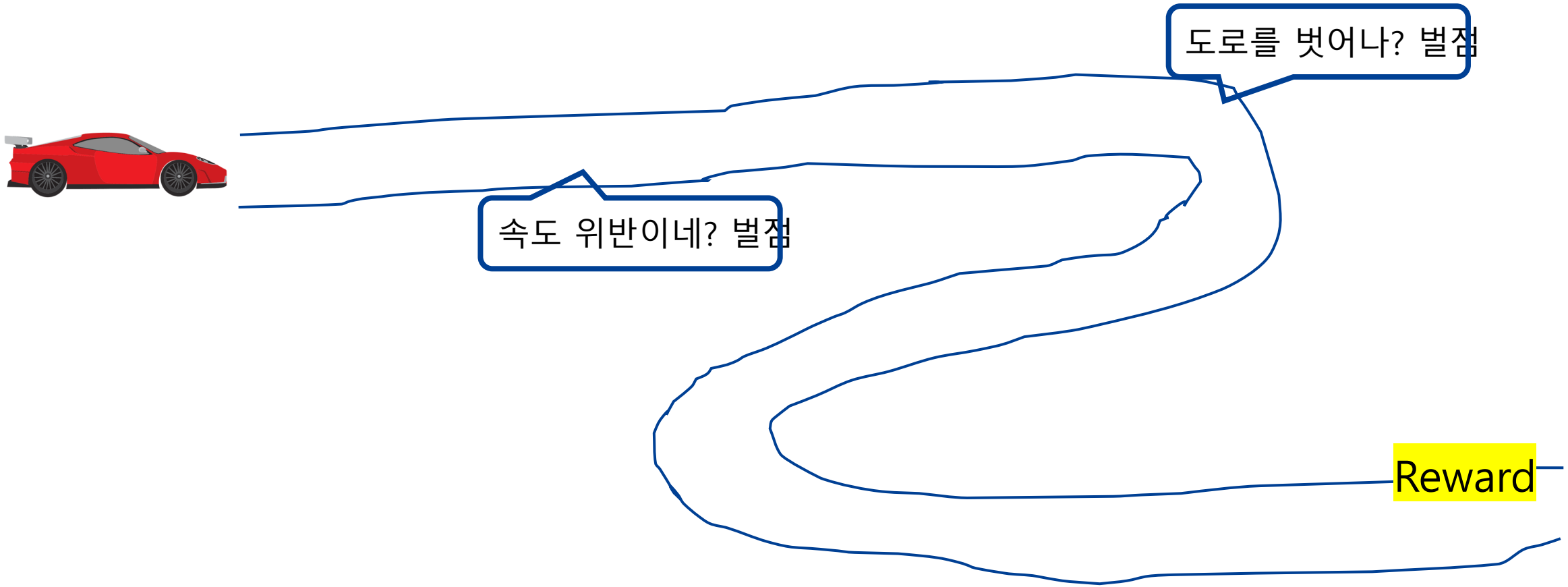
TU: 3

# 3) 준법 정신: Contrained RL 적용

서울시립대학교 UNIVERSITY OF SEOUL

다양한 선택지를 주지만, 최소한의 제약 사항을 지키도록 학습할 수 있지 않을까?
최고 속도를 제한하고, 도로를 벗어나지 않는 준법 정신있는 Constrained Reinforcement Learning

참고논문: 'Constrained Reinforcement Learning Has Zero Duality Gap' (NeurIPS 2019)



# 3) 준법 정신: Contrained RL 적용

서울시립대학교 UNIVERSITY OF SEOUL

라그랑지안 방법(Lagrangian Relaxation)을 활용한 Constrained Reinforcement Learning

알고리즘 설명
1) 초기화: 초기 정책과 라그랑지 승수 설정
2) 반복: Policy optimization > 제약조건 평가 > 라그랑지안 승수 업데이트

$$\max_{\pi} \mathbb{E}_{\pi}[R(s,a)] \quad \text{subject to} \quad \mathbb{E}_{\pi}[g_i(s,a)] \leq c_i,\ \forall i$$

최종 목표

제약조건 함수 – 제약조건 한계값

$$\mathcal{L}(\pi,\lambda) = \mathbb{E}_{\pi}[R(s,a)] - \sum_{i=1}^{m} \lambda_i \left(\mathbb{E}_{\pi}[g_i(s,a)] - c_i\right)$$

$$\pi^* = \arg\max_{\pi} \mathcal{L}(\pi,\lambda), \quad \lambda_i \geq 0,\ \forall i$$

최적 정책은 라그랑지안 함수 최대화 만족

$$\lambda_i \leftarrow \lambda_i + \eta \cdot \max(0, \mathbb{E}_{\pi}[g_i(s,a)] - c_i)$$

# 3) 준법 정신: Contrained RL 적용

서울시립대학교 UNIVERSITY OF SEOUL

라그랑지안 방법(Lagrangian Relaxation)을 활용한 Constrained Reinforcement Learning

알고리즘 설명
1) 초기화: 초기 정책과 라그랑지 승수 설정
2) 반복: Policy optimization > 제약조건 평가 > 라그랑지안 승수 업데이트

```python
def compute_lagrangian_reward(self, reward, info):
    """
    제약 조건 반영 보상 계산 (Constrained RL 활성화 시).
    """
    if not self.use_constrained_rl:
        return reward

    speed = info.get("speed", 0)
    off_track = info.get("off_track", False)

    # 제약 조건 위반 계산
    speed_violation = max(0, speed - self.max_speed)  # 속도 초과량
    off_track_penalty = 1 if off_track else 0

    # 라그랑지 조정 보상
    return reward - self.lambda_speed * speed_violation - self.lambda_off_track * off_track_penalty
```

# 3) 준법 정신: Contrained RL 적용

서울시립대학교 UNIVERSITY OF SEOUL

라그랑지안 방법(Lagrangian Relaxation)을 활용한 Constrained Reinforcement Learning

알고리즘 설명
1) 초기화: 초기 정책과 라그랑지 승수 설정
2) 반복: Policy optimization > 제약조건 평가 > 라그랑지안 승수 업데이트

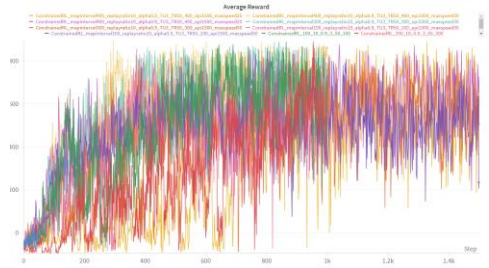
```python
def update_lagrange_multipliers(self, info):
    """
    라그랑지 승수 업데이트 (Constrained RL 활성화 시).
    """
    if not self.use_constrained_rl:
        return

    speed = info.get("speed", 0)
    off_track = info.get("off_track", False)

    # 라그랑지 승수 업데이트
    speed_violation = max(0, speed - 50)
    off_track_penalty = 1 if off_track else 0

    self.lambda_speed = max(0, self.lambda_speed + self.lambda_lr * speed_violation)
    self.lambda_off_track = max(0, self.lambda_off_track + self.lambda_lr * off_track_penalty)
```

# 3) 준법 정신: Contrained RL 적용




**Config parameters:** {} 11 keys

alpha: 0.9

constrained_use: 1

env_count: 4

episodes: 1,500

map_reset_interval: 400

max_speed: 25

max_timestep: 900

replay_ratio: 10

TC: 5

TR: 50

TU: 3

# 4) 확장된 시각 능력: Transformer

서울시립대학교 UNIVERSITY OF SEOUL

**CarEnvironment 환경**

매 step의 image값을 전처리하여 Stack_frame(4)만큼 쌓아서 observation 값으로 사용함.

```python
class CarEnvironment(gym.Wrapper):
    def __init__(self, env, skip_frames=2, stack_frames=4, no_operation=5, **kwargs):
        super().__init__(env, **kwargs)
        self._no_operation = no_operation
        self._skip_frames = skip_frames
        self._stack_frames = stack_frames

    def reset(self):
        observation, info = self.env.reset()

        for i in range(self._no_operation):
            observation, reward, terminated, truncated, info = self.env.step(0)

        observation = image_preprocessing(observation)
        self.stack_state = np.tile(observation, (self._stack_frames, 1, 1))
        return self.stack_state, info

    def step(self, action):
        total_reward = 0
        for i in range(self._skip_frames):
            observation, reward, terminated, truncated, info = self.env.step(action)
            total_reward += reward
            if terminated or truncated:
                break

        observation = image_preprocessing(observation)
        self.stack_state = np.concatenate((self.stack_state[1:], observation[np.newaxis]), axis=0)
        return self.stack_state, total_reward, terminated, truncated, info
```

# 4) 확장된 시각 능력: Transformer

서울시립대학교 UNIVERSITY OF SEOUL

**기본 CNN 모듈**

4개의 연속된 observation 데이터를 공간적 특성으로 간주하며 시간적인 정보를 고려하지 못함.

```python
class CNN(nn.Module):
    def __init__(self, in_channels, out_channels, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._n_features = 32 * 9 * 9

        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, 16, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(16, 32, kernel_size=4, stride=2),
            nn.ReLU(),
        )

        self.fc = nn.Sequential(
            nn.Linear(self._n_features, 256),
            nn.ReLU(),
            nn.Linear(256, out_channels),
        )


    def forward(self, x):
        x = self.conv(x)
        x = x.view((-1, self._n_features))
        x = self.fc(x)
        return x
```

# 4) 확장된 시각 능력: Transformer

```python
def forward(self, x):
    # Input shape: (batch_size, channels, frames, 1, height, width)
    batch_size, channels, frames, _, height, width = x.size()

    assert channels == 1, f"Expected channels to be 1, got {channels}"

    # Process each frame through CNN
    cnn_features = []
    for i in range(stop/frames):
        frame = x[:, :, i, 0]  # Shape: (batch_size, 1, height, width)
        frame_features = self.cnn(frame)  # Shape: (batch_size, 32, 9, 9)
        cnn_features.append(object/frame_features.view(batch_size, -1))  # Flatten: (batch_size, 32 * 9 * 9)

    # Stack CNN features for all frames
    cnn_features = torch.stack(cnn_features, dim=1)  # Shape: (batch_size, frames, 32 * 9 * 9)

    # Embed features for Transformer
    embeddings = self.embedding_layer(cnn_features)  # Shape: (batch_size, frames, hidden_dim)

    # Transformer expects input as (seq_len, batch_size, hidden_dim)
    transformer_input = embeddings.permute(1, 0, 2)  # Shape: (frames, batch_size, hidden_dim)
    transformer_output = self.transformer(transformer_input)  # Shape: (frames, batch_size, hidden_dim)

    # Use the last frame's output
    last_frame_output = transformer_output[-1]  # Shape: (batch_size, hidden_dim)

    # Fully connected layers
    output = self.fc(last_frame_output)  # Shape: (batch_size, out_channels)

    return output
```

yers=1):

Obser
사용할
Tr

단일 o
Feat
Tran

https://

# 4) 확장된 시각 능력: Transformer

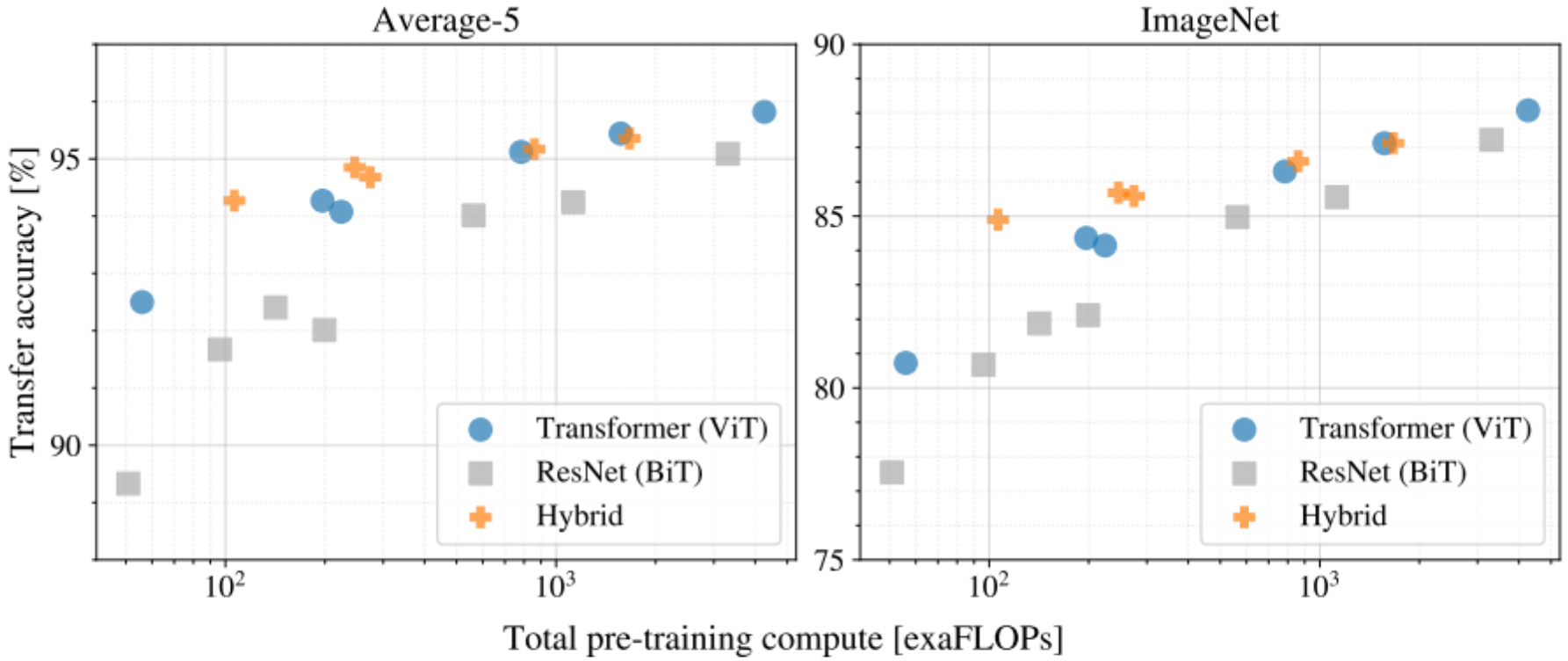



Figure 5: Performance versus pre-training compute for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

https://arxiv.org/pdf/2010.11929

# 4) 확장된 시각 능력: Transformer

서울시립대학교
UNIVERSITY OF SEOUL

**CNN+Transformer**

All Scores: [882.3181818181655, 858.0971731448608, 896.3328621907975, 836.8957597172993, 889.8992932862029, 833.3621908127058, 894.0328621907976, 798.0265017667695, 794.4929328621761, 851.0300353356735, …] Average Score: 798.6006097173105

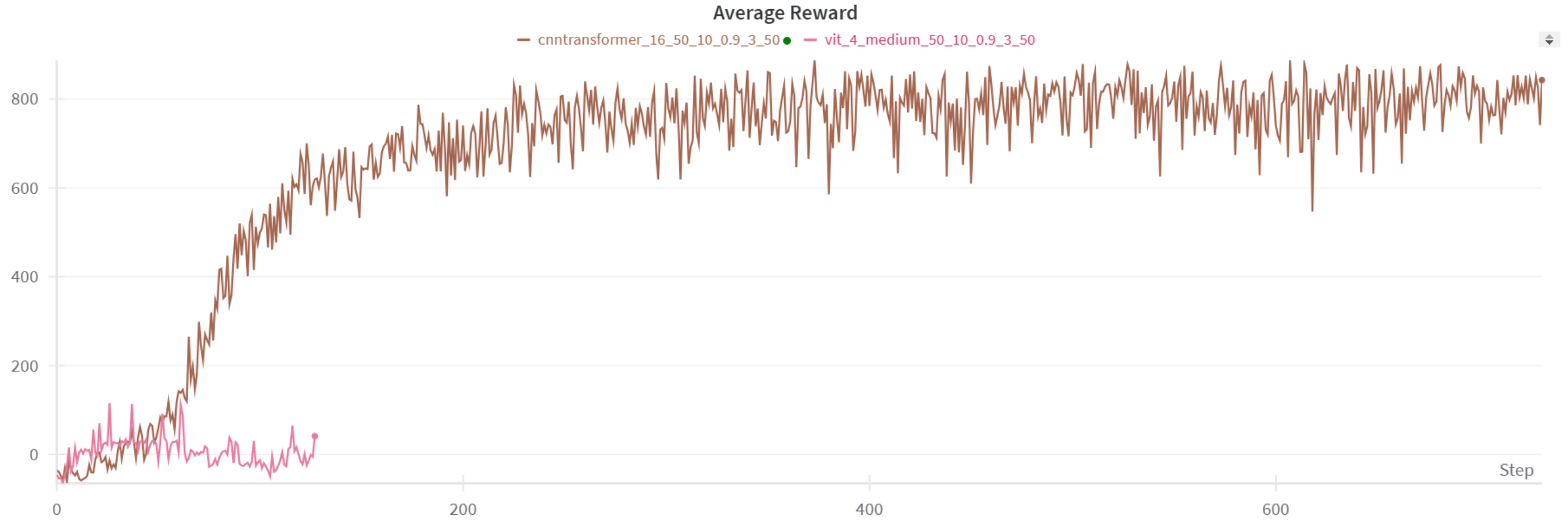
**CNN Only**

All Scores: [664.1986301369715, 723.8215547703037, 723.8215547703037, 723.8215547703037, 723.8215547703037, 723.8215547703037, 723.8215547703037, 723.8215547703037, 723.8215547703037, 723.8215547703037, …] Average Score: 631.5752055091165

100개의 random environment에서 기존 CNN을 사용한
모델보다 꾸준히 더 좋은 성능을 보임

# 4) 확장된 시각 능력: Transformer

추가적인 구현 세부사항으로,
CNN의 Feature map을 Transformer를 통해 Encoding하는 과정에서
Positional Encoding을 사용하지 않는 경우 제대로 학습이 불가능함

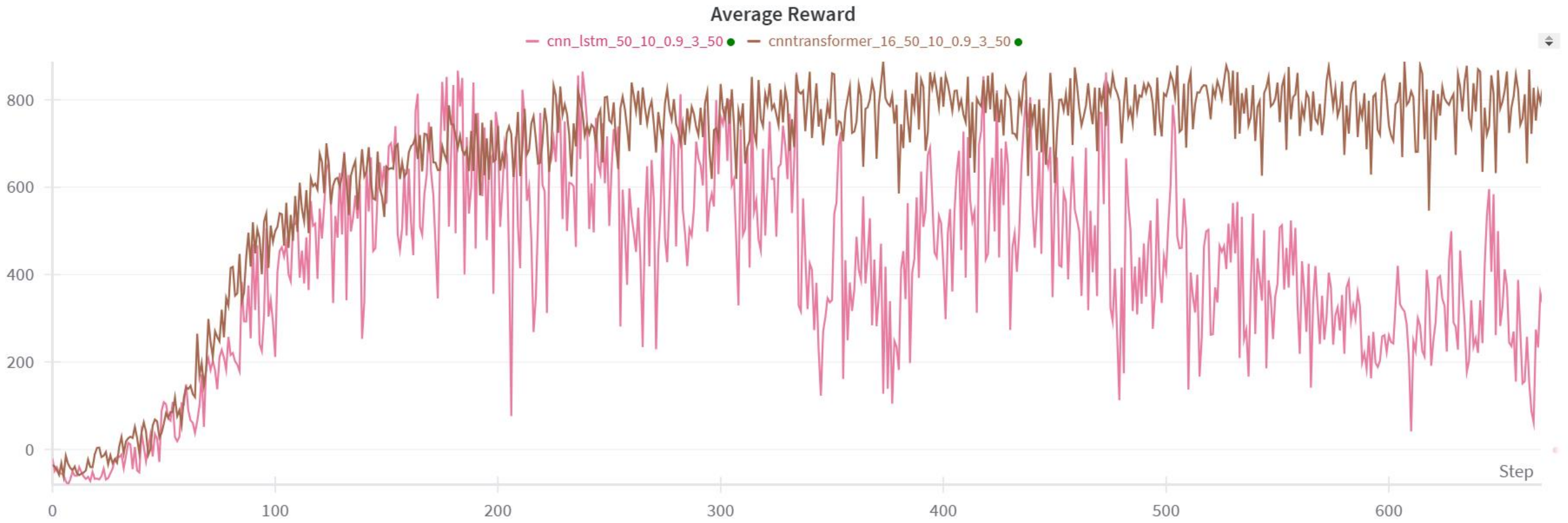


# 5) 모델 경량화 시도 CNN + LSTM

서울시립대학교 UNIVERSITY OF SEOUL

**CNN+LSTM**

CNN을 통해 공간적 정보를, Transformer를 통해 시간적 정보를 처리했을 때 성능이 향상됨.
위 방식을 차용해 CNN과 LSTM을 결합하는 방식을 시도했으나,
기본 CNN보다 낮은 성능을 보임



# 5) 최종 모델의 압도적인 주행 시범

서울시립대학교
UNIVERSITY OF SEOUL

JupyterNotebook을 통해 시현

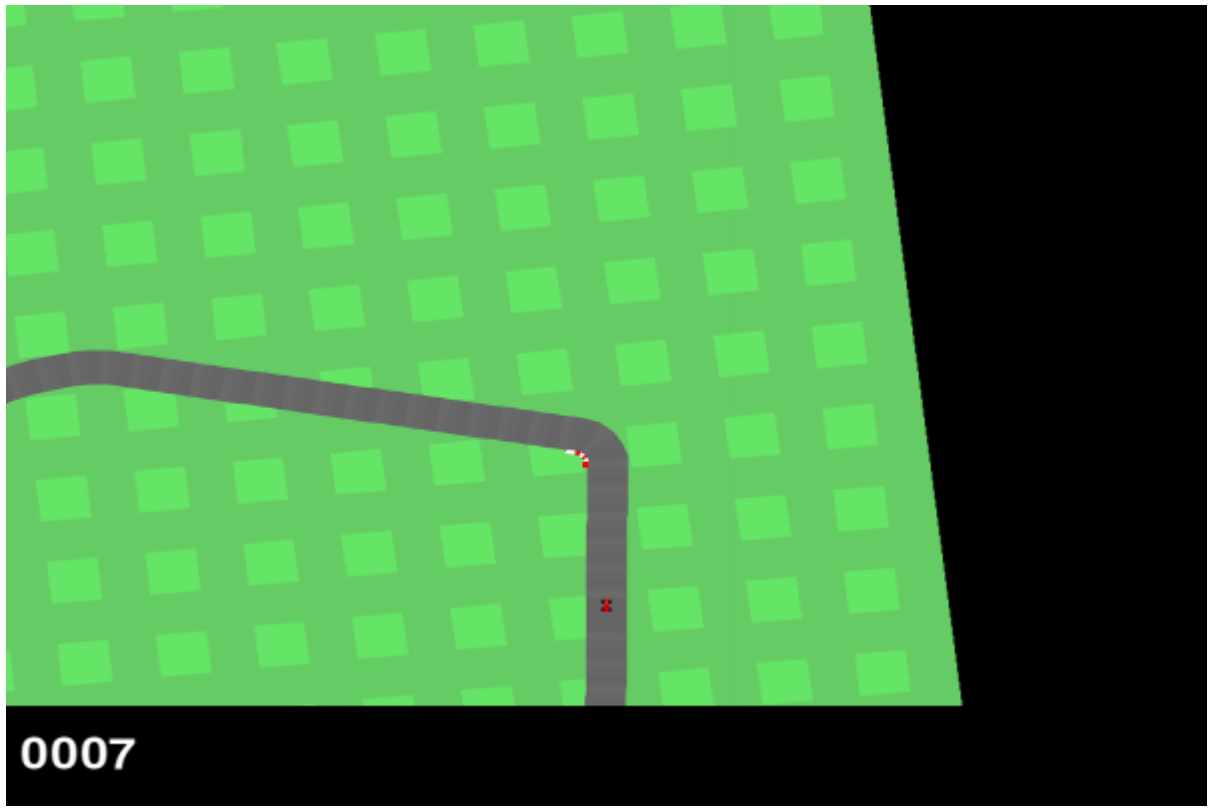
1) Plasticity 적용 모델
2) Plasticity + Constrained 적용 모델

# 5) 최종 모델의 압도적인 주행 시범

서울시립대학교
UNIVERSITY OF SEOUL

JupyterNotebook을 통해 시현
1) Plasticity 적용 모델

900점



# 5) 최종 모델의 압도적인 주행 시범

서울시립대학교
UNIVERSITY OF SEOUL

JupyterNotebook을 통해 시현
2) Plasticity + Constrained 적용 모델

876점