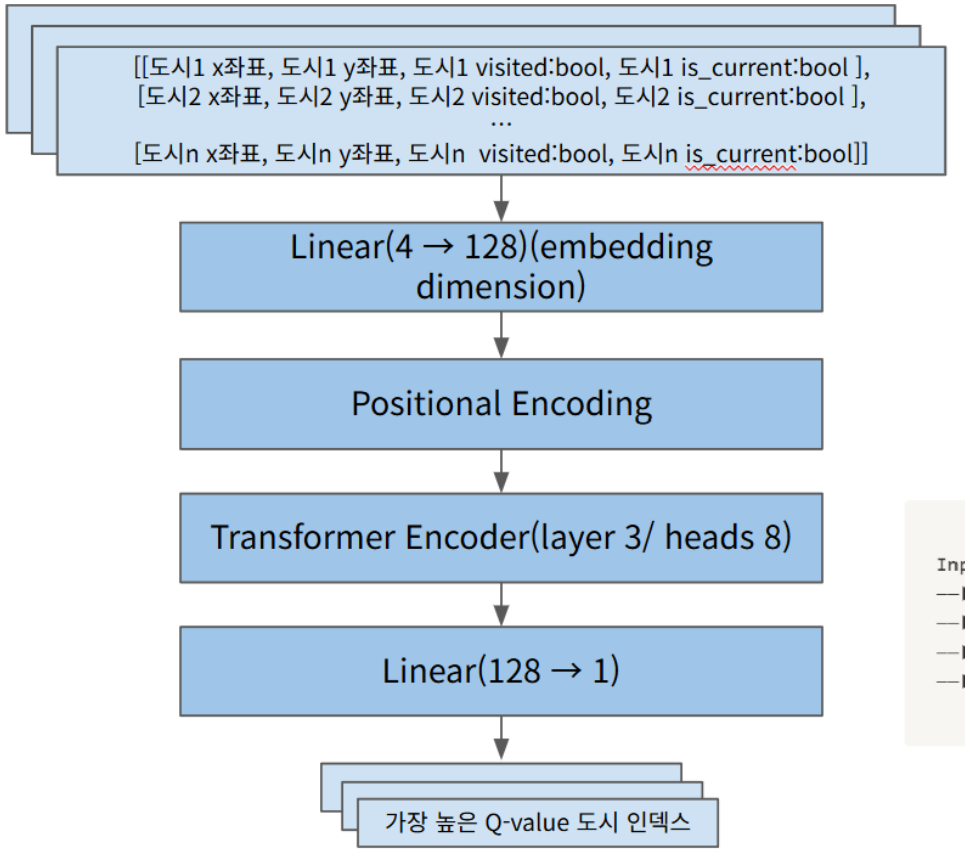
TSP Policy based Transformer

임베디드시스템공학과 김정현
컴퓨터공학과 정환길
산업경영공학과 한근형

Review

- **Value-based (Only Transformer Encoder)**

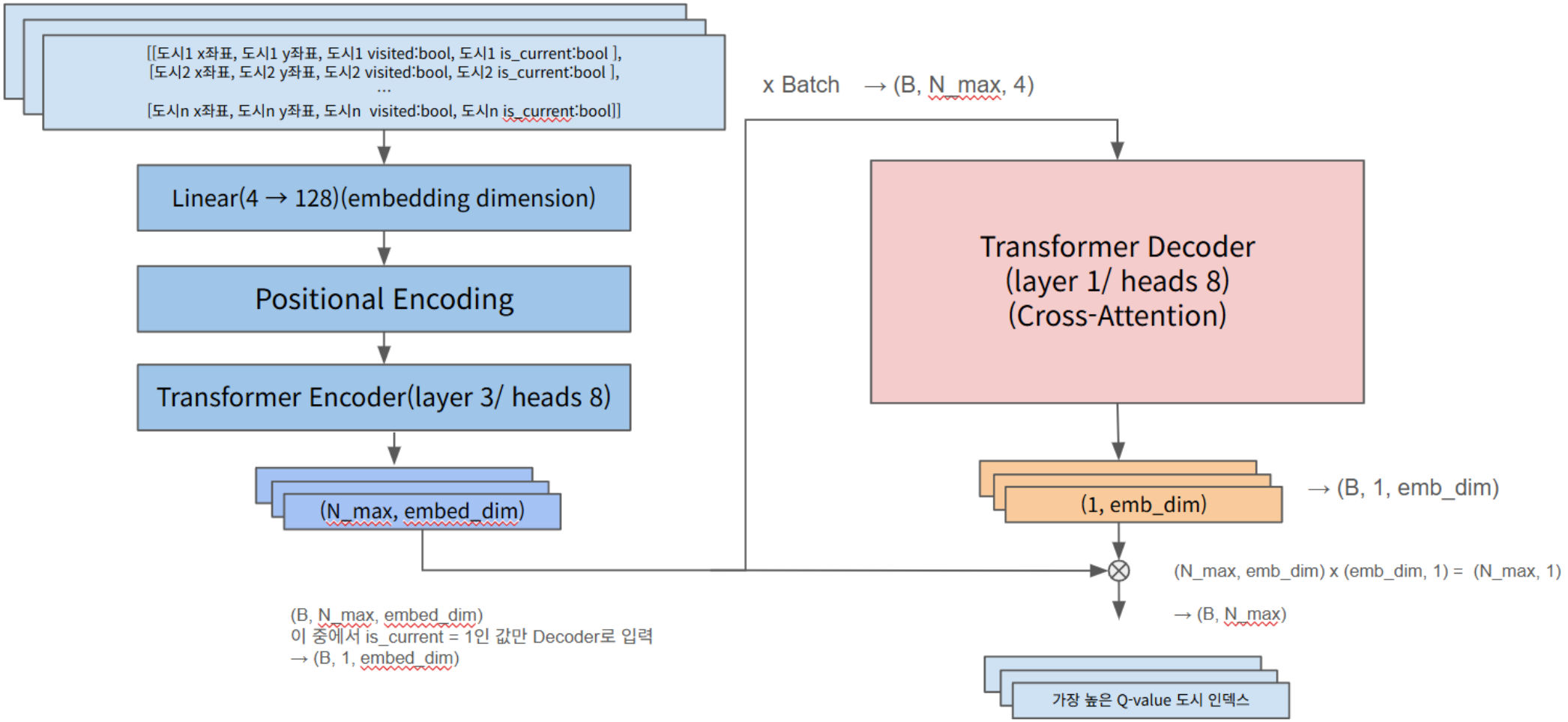


x Batch → (B, N_max, 4)

```
Input: (B, N_max, 4)
—▶ Linear(4 → emb_dim)
—▶ PositionalEncoding
—▶ TransformerEncoder (n_layers × Multi-Head Attention)
—▶ Linear(emb_dim → 1) → Q-values (B, N_max)
```

Review

- **Value-based (Full Transformer)**



Contents

Objective

- **과제 목적**
  - Transformer의 Encoder 구조와 policy-based(REINFORCE) 방식을 통해서 Traveling Salesman Problem 해결
- **제약 사항**
  - **State**
    - (x, y) coordinates of each city & the status of each city
  - **Action**
    - Acity to visit next time ⇒ Choose the next city by $\pi\theta(a|s)$
    - You should mask the visited cities
  - **Reward**
    - -1 × distance between the cities

Method

■ Summary

• 학습 데이터 (커리큘럼 학습)

- ► 노드 수가 적은 문제부터 노드 수가 많은 복잡한 문제 순으로 학습
- ► 단계가 바뀔 때마다 경험 캐시를 비워 새로운 분포로 학습

단계	도시 개수	에피소드 수
1	10	1,400
2	10, 20	2,400
3	20, 30, 40	3,200

• 하이퍼 파라미터 및 실험 환경

항목	값
임베딩 차원 (EMB_DIM)	128
헤드 수 / 레이어 수	8 / 4
드롭아웃 비율	0.1
옵티마이저 / 학습률	Adam / 2e-4
배치 크기	24 에피소드
감쇠율 γ	1.0 (에피소드 단위)
그래디언트 클리핑	1.0
AMP	활성화

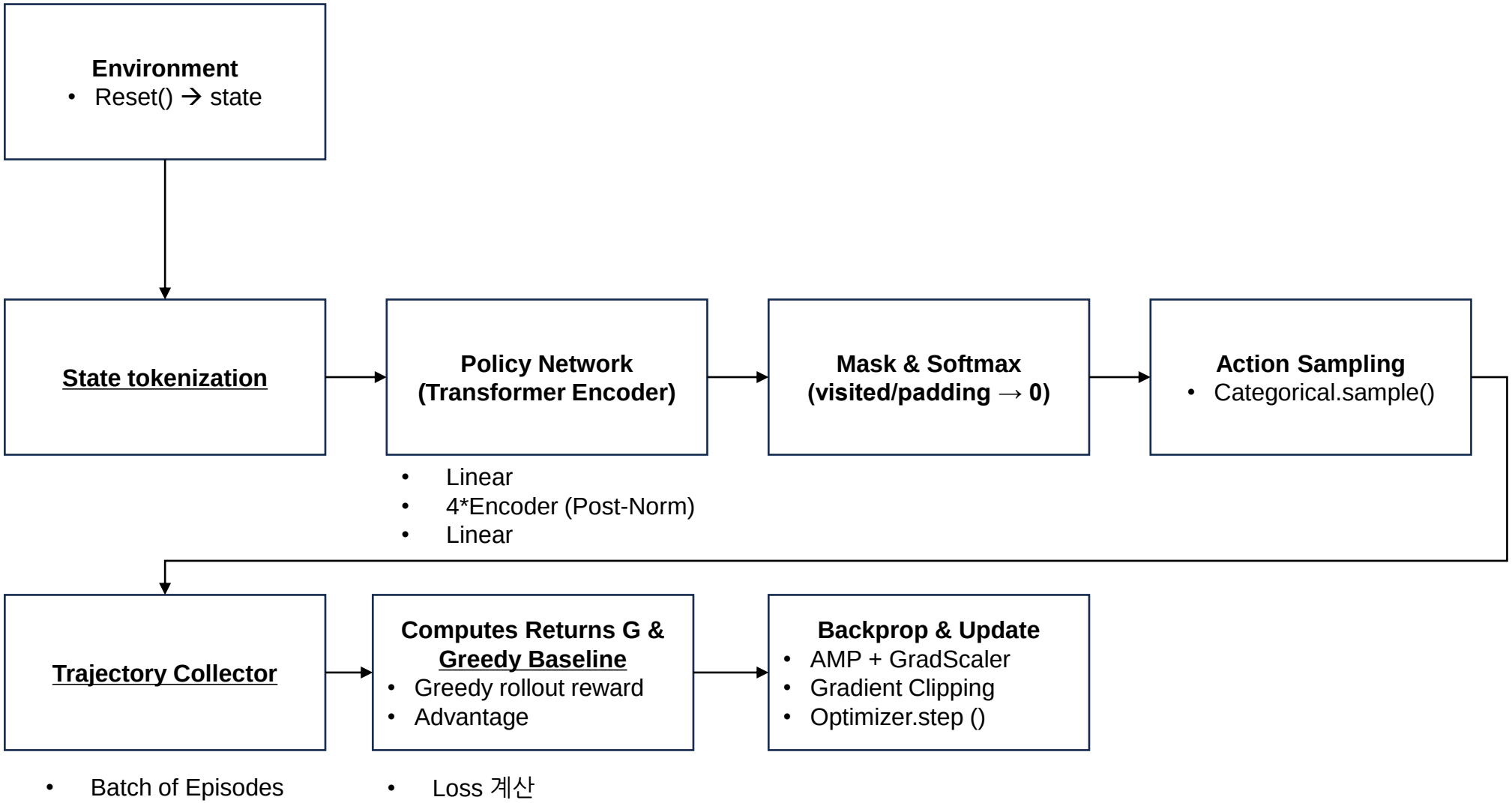
실험 환경

✓ OS: Ubuntu 22.04.3 LTS x86_64 CPU: Intel Xeon W-2255 (20) @ 4.500GHz GPU: NVIDIA Quadro RTX 5000 16GBMemory: 128491MiB

AMP (Automatic Mixed Precision)

✓ 연산 일부를 FP16(half)로 처리 → GPU 메모리 절약, 연산 속도↑
✓ torch.cuda.amp.GradScaler 로 작은 값의 기울기도 안전하게 스케일링

Method


Environment
• Reset() → state
State tokenization
Policy Network
(Transformer Encoder)
• Linear
• 4*Encoder (Post-Norm)
• Linear
Mask & Softmax
(visited/padding → 0)
Action Sampling
• Categorical.sample()
Trajectory Collector
• Batch of Episodes
Computes Returns G &
Greedy Baseline
• Greedy rollout reward
• Advantage
• Loss 계산
Backprop & Update
• AMP + GradScaler
• Gradient Clipping
• Optimizer.step ()

Method

- **State (좌표 포함 7가지 차원)**
  - **1. 절대 좌표 (x, y)**
    - [0,1] 구간에서 무작위로 생성된 도시의 위치
  - **2. 상대 좌표 (dx, dy)**
    - $dx = x_i - x\_cur$, $dy = y_i - y\_cur$
    - 현재 머물러 있는 도시(current city)와의 상대적 방향
  - **3. 거리 정보 (d_norm)**
    - 상대 좌표의 크기, 즉 “지금 있는 도시에서 이 도시까지의 직선 거리”
  - **4. 방문 여부 (visited)**
    - 이미 경로에 포함된 도시는 1, 미포함 도시는 0으로 표시
    - 방문한 도시는 다시 방문하지 않도록 마스킹할 때 사용
  - **5. 현재 도시 표시 (is_current)**
    - 원-핫(one-hot) 방식으로, 현재 모델이 머물고 있는 도시만 1, 나머지는 0

Method

- **Transformer Encoder**
  - Positional encoding 사용 X
  - Categorical 분포로 다음 도시 선택(샘플링 또는 greedy argmax)
  - **Self-Attention**
    - 각 도시 임베딩이 서로 얼마나 중요한지 가중치(attention)를 계산

Method

- **REINFORCE**
  - Policy → 모델이 어떤 도시를 선택할지에 대한 확률 분포를 출력
  - **샘플링 (rollout)**
    - ► 모델이 매 단계마다 도시를 확률적으로 선택 (sampling)
    - ► 전체 경로 길이를 L_sample 계산
  - **손실 함수 (Loss)**

$$\mathcal{L} = -\sum_{t=1}^{T} \log \pi(a_t \mid s_t) \times R$$

  - $log\pi(a_t \mid s_t)$: 선택된 행동(도시) 확률의 로그
  - R: 에피소드 전체 보상((-)경로 길이)
  - 짧은 경로를 만들면 R가 커지므로((-) 길이 → 큰 값), 손실이 줄어들도록 학습

Method

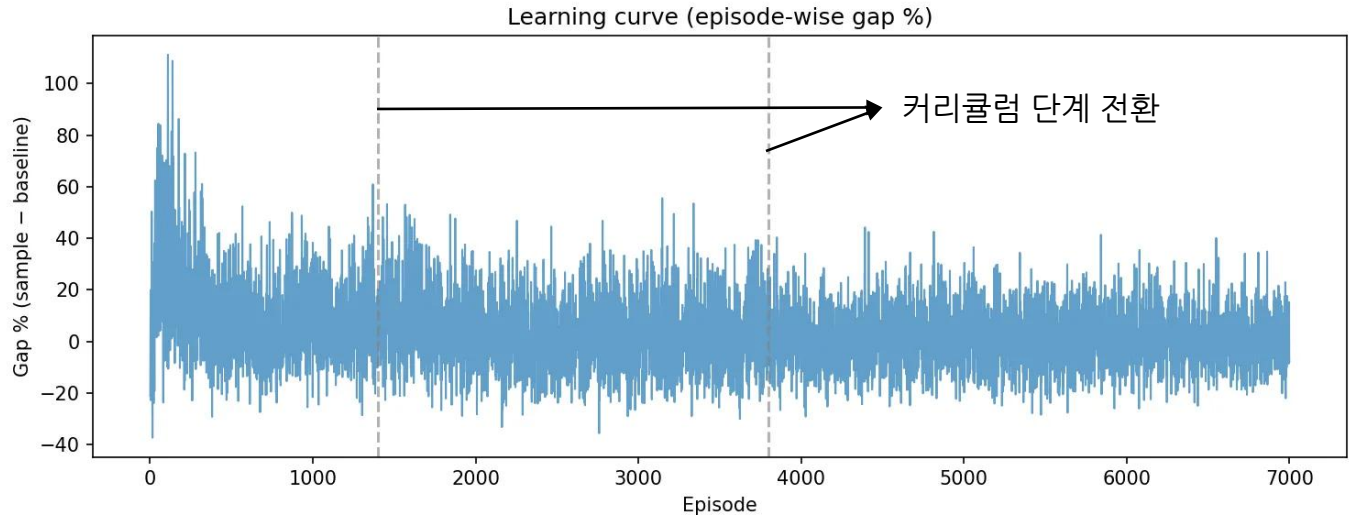
■ Self-Critical Baseline

- REINFORCE는 보상이 **분산(variance)이 커서 학습이 불안정 → ** 기준 경로를 하나 더 굴려서 보상
- 기존 경로를 greedy만으로 한번 더 풀어서 아래와 같이 보상을 계산

1. **샘플 경로** → 보상 R_s
2. **Greedy 경로**(argmax만 사용) → 보상 R_g
3. **Advantage:** $A = R_s - R_g$
4. **변경된 손실:** $\mathcal{L} = -\sum_t \log \pi(a_t) \times A$
   - 샘플 경로가 더 짧으면 R_s가 크고, $A > 0$ → 정책을 강화(확률↑)
   - 샘플 경로가 더 길면 $A < 0$ → 정책을 억제(확률↓)

Result

- **학습 결과**
  - 총 소요 시간: **약 14분**
  - **실험 환경:**
    - OS: Ubuntu 22.04.3 LTS x86_64
    - CPU: Intel Xeon W-2255 (20) @ 4.500GHz
    - GPU: NVIDIA Quadro RTX 5000 16GBMemory: 128491MiB



✓ 학습 곡선의 수렴이 300 에피소드 이후부터 진행
✓ 차이가 0에 가까움 → 안정적으로 수렴
✓ 음수 구간 증가 → 샘플링 경로가 baseline보다 짧은 에피소드가 자주 등장

Fig. X축: 에피소드, Y축: 샘플링한 투어길이와 Greedy 차이

Result

- 학습 결과
  - 각 도시 수(N=10, 20, 30)에 따라 각각 5문제 테스트
    - TSP10_avg_gap = 6.94%
    - TSP20_avg_gap = 10.63%
    - TSP30_avg_gap = 14.04%

→ Overall avg gap 10.54% over 15 instances

전체 실험 결과 비교

Problem	Optimal	Q-learning Best	DQN Method 1	DQN Method 2 (Instance)	DQN Method 3 (Generalization)	Value-based: Transformer Method (Encoder only)	Value-based: Transformer Method (Full)	Policy-based: Transformer Method (Encoder only)
Problem 1	3.3478	3.5113 (+4.88%)	3.3478 (+0.00%)	3.6692 (+9.60%)	8.6396 (+158.07%)	3.6896 (+10.21%)	3.4717 (+3.70%)	3.707 (+10.75%)
Problem 2	4.1313	4.5012 (+8.95%)	4.1313 (+0.00%)	5.4612 (+32.19%)	12.1222 (+193.42%)	4.4312 (+7.26%)	4.1857 (+1.32%)	4.6261 (+11.976%)
Problem 3	4.4684	5.3281 (+19.24%)	4.4684 (+0.00%)	5.1575 (+15.42%)	11.8222 (+164.57%)	5.0166 (+12.27%)	4.8078 (+7.60%)	4.8842 (+9.399%)
Problem 4	3.8637	4.3309 (+12.09%)	3.8637 (+0.00%)	5.0688 (+31.19%)	9.0085 (+133.16%)	4.8316 (+25.05%)	4.5003 (+16.48%)	4.06881 (+5.307%)
Problem 5	4.1212	4.6724 (+13.37%)	4.1216 (+0.01%)	4.6414 (+12.62%)	10.8255 (+162.68%)	4.3914 (+6.56%)	4.4337 (+7.58%)	4.7863 (+15.709%)

Conclusion

- **Summary**
  - **풍부한 입력 표현**과 Pre-Norm 구조로 Transformer가 “어디서 어떻게 이동할지”를 더 명확히 학습
  - Self-Critical Baseline과 미니배치 업데이트로 REINFORCE 특유의 높은 분산을 안정적으로 낮춤
  - AMP, 클리핑 조합으로 학습 효율과 탐색성을 동시에 확보

- **Limitations**
  - **Policy-based(REINFORCE):** 높은 분산, roll-out으로 돌아오는 정보량이 제한적, 장기 의존성 처리 어려움