제약 조건 추가 및 실험

제약 조건 정리

weight

하중 제약은 바닥의 각 위치가 허용 하중 값 $W(x, y)$를 가진다고 보고, 설비는 하중 요구 값 w를 가진다고 보는 규칙입니다.
배치 가능 조건은 설비 footprint가 덮는 모든 위치에 대해 $w \le W(x, y)$가 성립하는 것입니다. 따라서 footprint 안에 단 한 칸이라도 $W(x, y) < w$인 셀이 포함되면 즉시 배치 불가가 됩니다.
이 때문에 무거운 설비는 허용 하중이 높은 구역만 밟을 수 있고, 경계에 조금이라도 걸치면 실패합니다. 부분 겹침은 허용하지 않습니다.

height

높이 제약은 바닥의 각 위치가 천장 높이 값 $H(x, y)$를 가진다고 보고, 설비는 높이 h를 가진다고 보는 규칙입니다.
배치 가능 조건은 설비 footprint가 덮는 모든 위치에 대해 $h \le H(x, y)$가 성립하는 것입니다. 즉 footprint 안에 단 한 칸이라도 $H(x, y) < h$인 셀이 포함되면 배치 불가가 됩니다.
결과적으로 높은 설비는 낮은 천장 구역을 한 칸도 밟을 수 없고, 낮은 천장 존은 높은 설비에게 사실상 금지영역처럼 동작합니다.

dry

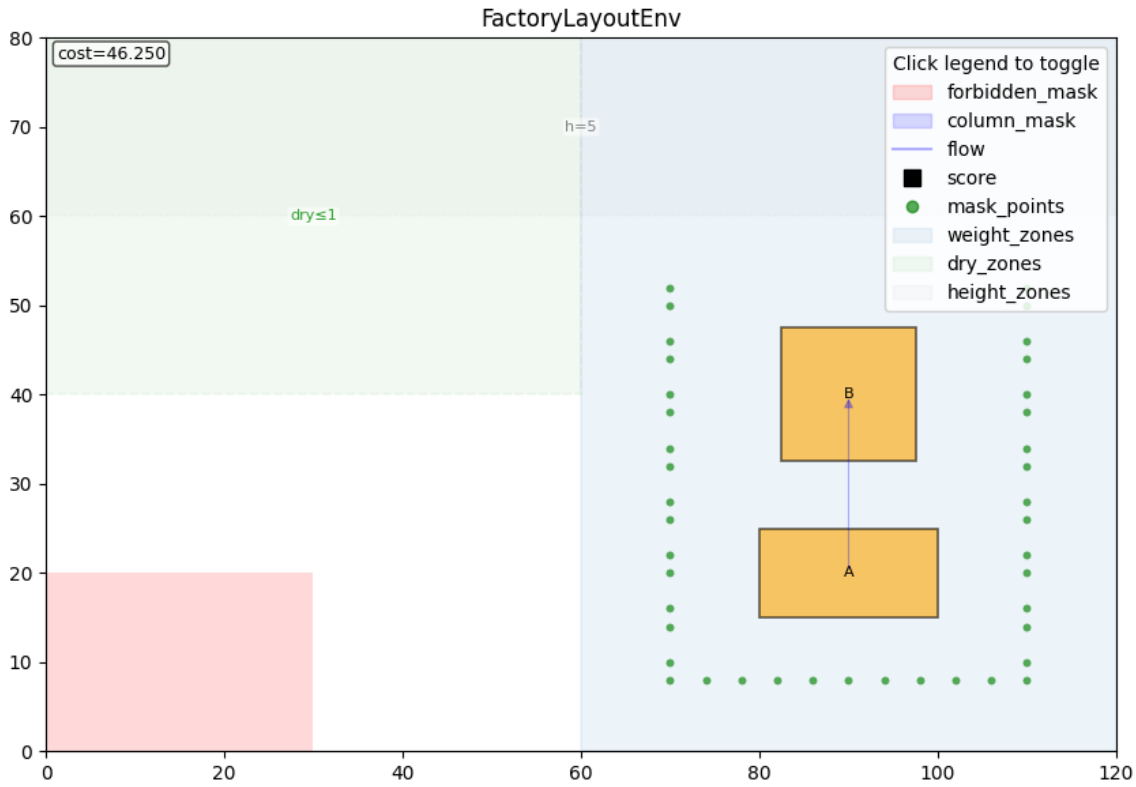
dry 제약은 바닥의 각 위치가 요구 dry 수준 $R(x, y)$를 가진다고 보고, 설비는 만족 가능한 dry 수준 d를 가진다고 보는 규칙입니다.
배치 가능 조건은 설비 footprint가 덮는 모든 위치에 대해 $d \ge R(x, y)$가 성립하는 것입니다. 따라서 footprint 안에 단 한 칸이라도 $R(x, y) > d$인 셀이 포함되면 즉시 배치 불가가 됩니다.

clearance

clearance는 설비 간 안전거리로 해석하며, body (실제 설비 면적)와 pad (body + clearance)를 구분해서 적용합니다. 배치 가능 조건은 아래와 같이 정리됩니다.

- 새 설비의 body가 기존 금지영역(정적 금지, 이미 점유, 존 제약)과 겹치면 실패
- 새 설비의 body가 이미 배치된 다른 설비들의 clearance 영역과 겹치면 실패
- 새 설비의 pad가 기존 금지영역과 겹치면 실패

단, 설비들 사이에서 clearance 영역끼리 서로 겹치는 것은 허용하여 안전거리 규칙이 배치를 과도하게 막는 상황을 줄입니다.



이러한 제약을 도입하면 설비마다 실제로 배치 가능한 영역의 크기가 크게 달라집니다. 따라서 초기 배치 순서는 단순히 면적이 큰 설비를 우선 배치한다는 기준이 아니라, (정적 금지 영역 + 해당 설비가 위반하는 존 제약)으로 인해 남는 유효 공간 대비 설비 footprint 비율이 클수록 먼저 배치하는 난이도 기반 기준을 사용하여 실패 가능성을 낮추었습니다. 또한 추후에는 footprint 크기별로 실제로 배치 가능한 위치 수를 빠르게 추정하여, 보다 정확한 난이도 정렬 방식으로 고도화할 수 있습니다.

또한 제약 및 clearance를 추가하면 배치 가능 판정이 더 자주, 더 높은 비용으로 호출되므로, 불필요한 대규모 맵 합성이나 재할당이 누적될 경우 시뮬레이션 비용이 크게 증가할 수 있습니다. 이에 따라 배치 가능 판정 과정에서는 조건을 한 번에 합쳐 큰 맵을 매번 생성하는 방식을 지양하고, 필요한 충돌 조건을 분리하여 단계적으로 확인하였습니다. 아울러 전체 금지 맵 갱신 역시 매번 새로운 텐서를 생성하는 대신 기존 버퍼를 갱신하는 방식으로 처리하여 메모리 및 연산 낭비를 줄였습니다.

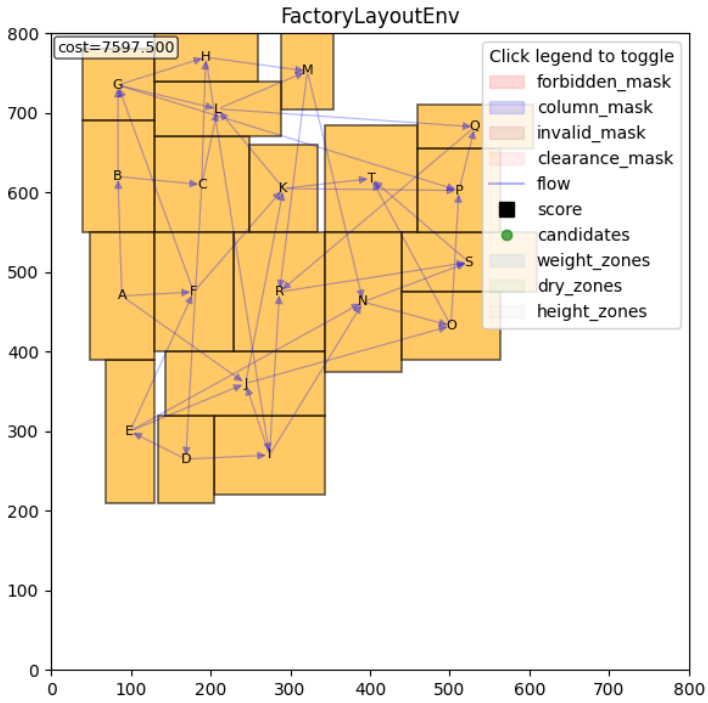
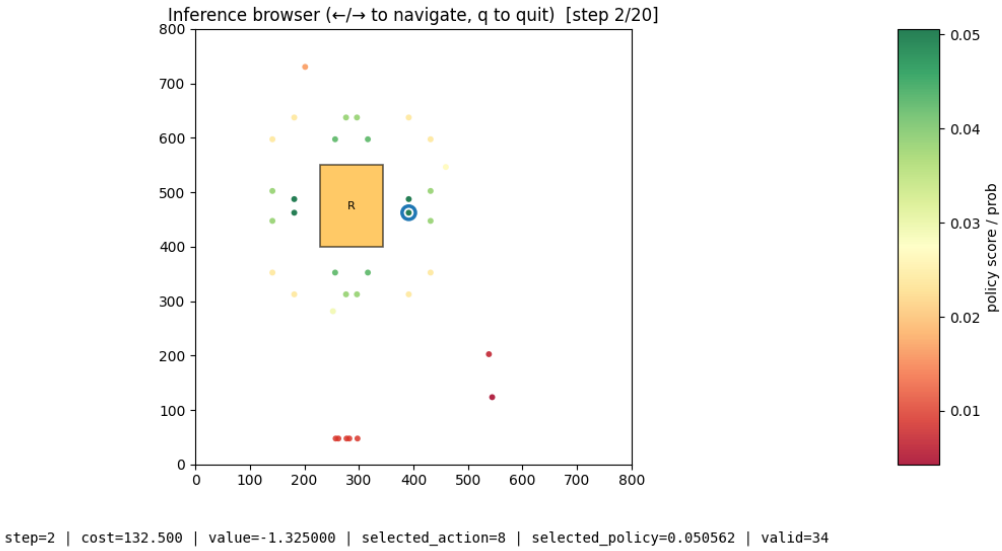
제약조건 실험

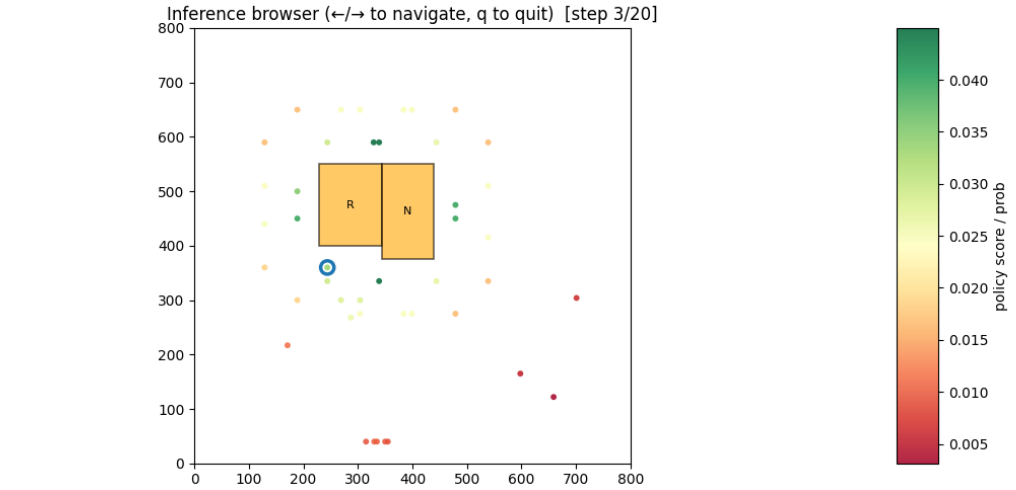
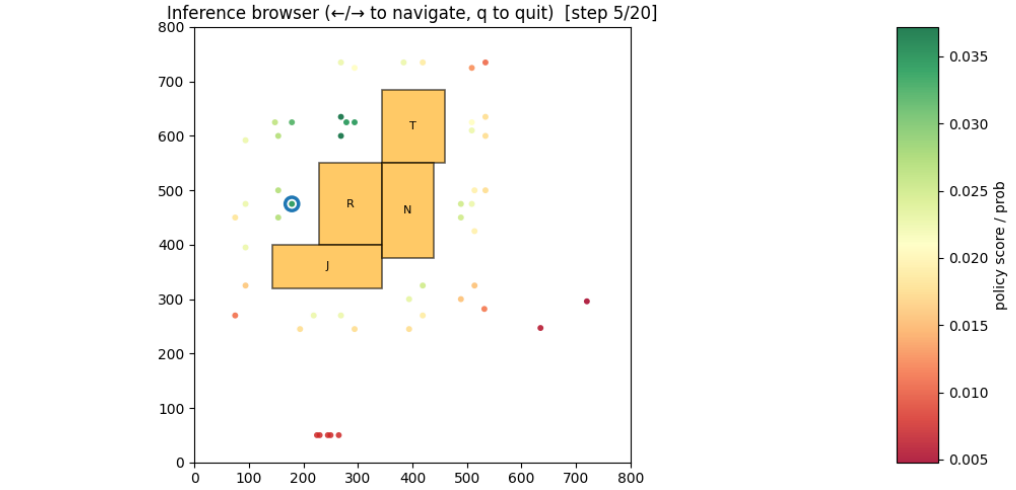
다음은 다양한 조건에서 에이전트가 탐색한 과정을 실험을 진행한 것입니다. 배치할 설비 수가 많은 경우, clearance 제약이 추가된 경우, 그리고 dry/height/weight zone 제약이 추가된 경우를 포함하여 비교하였습니다. 시각화에서는 초록색 좌표가 에이전트가 유망하다고 판단한 배치 중심점 후보를 의미하고, 빨간색 좌표는 유망하지 않다고 판단한 후보를 의미합니다. 실제로 다음 설비가 최종 배치된 위치는 파란색 원으로 표시되어 있습니다.

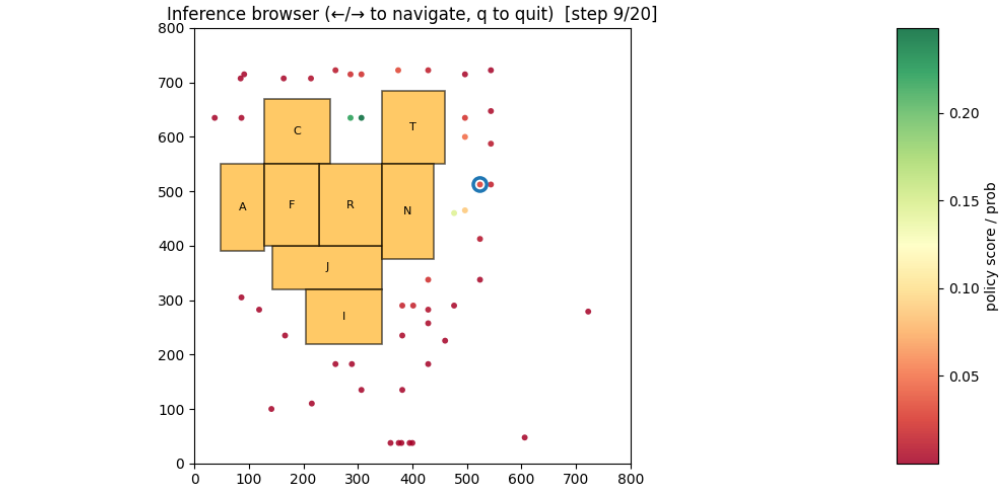
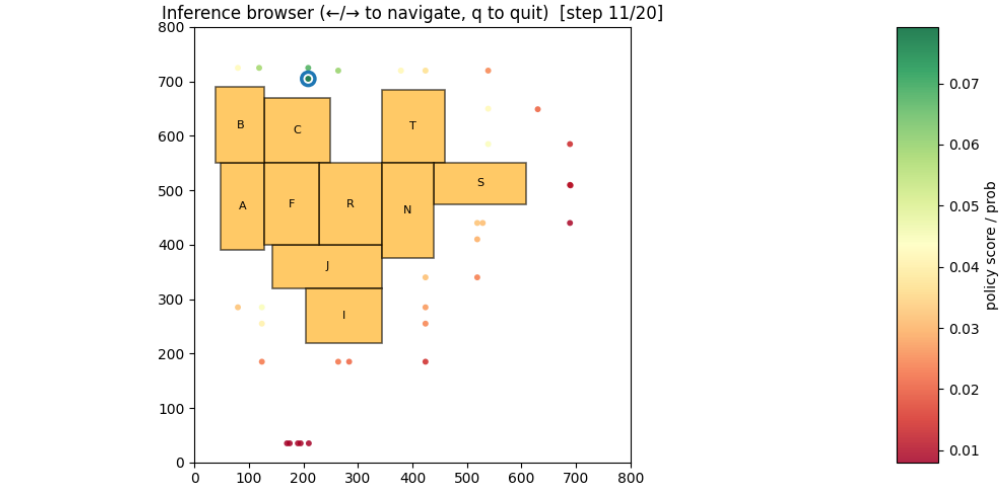
전반적으로 모든 조건에서 배치가 정상적으로 완료되었으며, 특히 clearance 조건에서는 에이전트가 단순히 기존 설비 주변에 밀집 배치하는 방식에만 의존하지 않고, 이후 물류 최적화 동선을 확보하기 위해 의도적으로 멀리 떨어진 위치를 선택하는 사례도 관찰되었습니다. 이는 에이전트가 현재 단계의 즉시 편의성뿐 아니라 향후 단계의 공간 활용과 배치 가능성을 함께 고려하는 탐색을 수행했음을 보여줍니다.

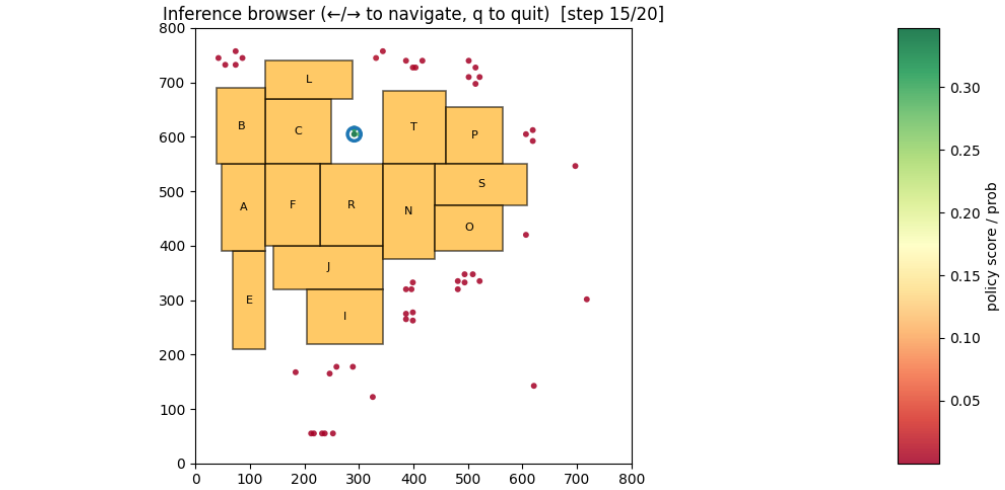
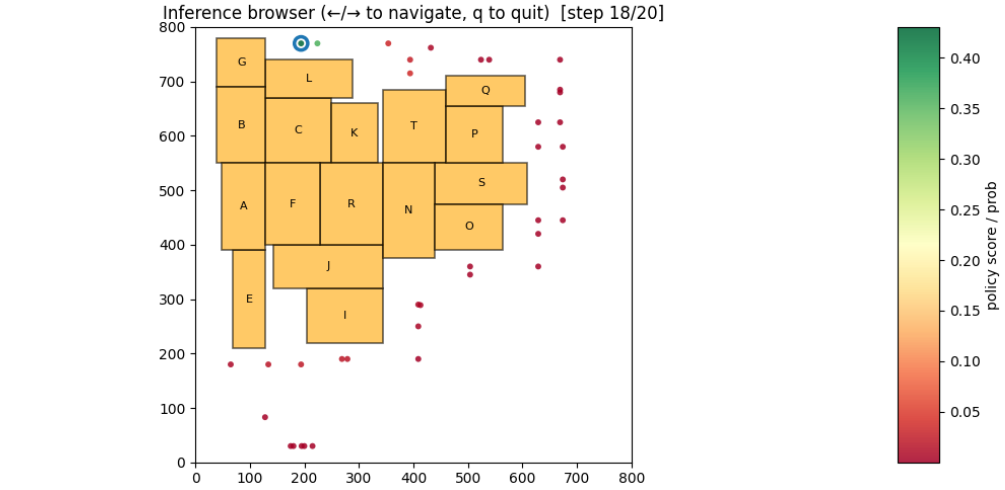
배치할 설비가 많은 경우 (> 20)

구분	이미지

구분	이미지
Final	
Step 1	

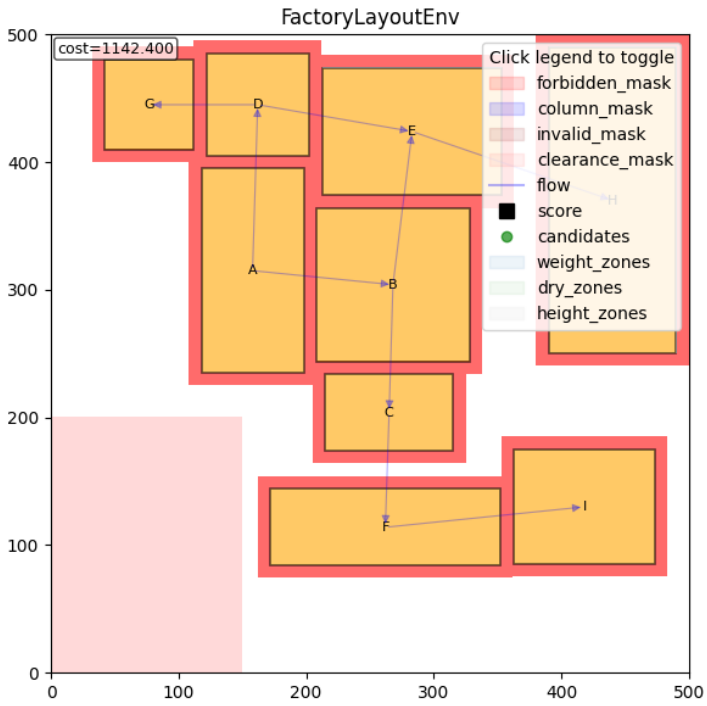
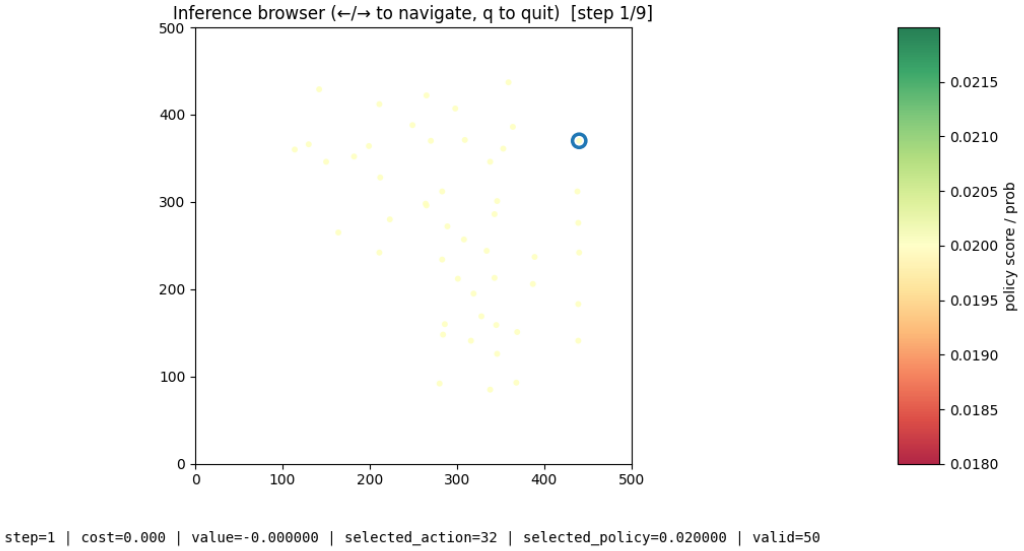
구분	이미지
Step 2	 step=3 | cost=192.500 | value=-1.925000 | selected_action=12 | selected_policy=0.033316 | valid=43
Step 4	 step=5 | cost=340.000 | value=-3.400000 | selected_action=7 | selected_policy=0.034493 | valid=50

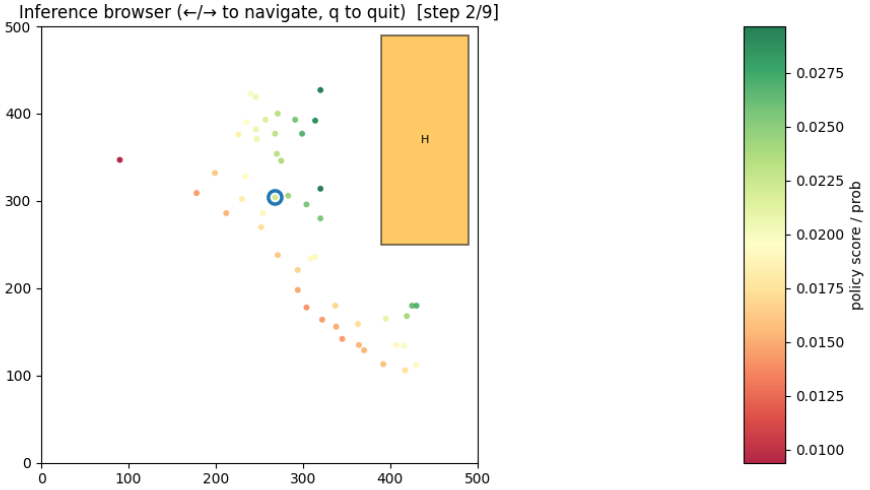
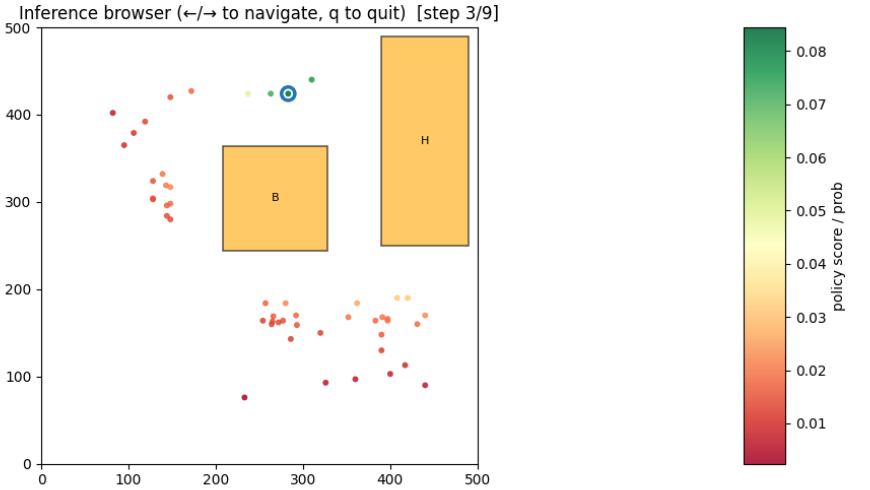
구분	이미지
Step 8	 `step=9 | cost=945.250 | value=-9.452500 | selected_action=11 | selected_policy=0.026825 | valid=50`
Step 10	 `step=11 | cost=1909.750 | value=-19.097500 | selected_action=5 | selected_policy=0.079223 | valid=34`

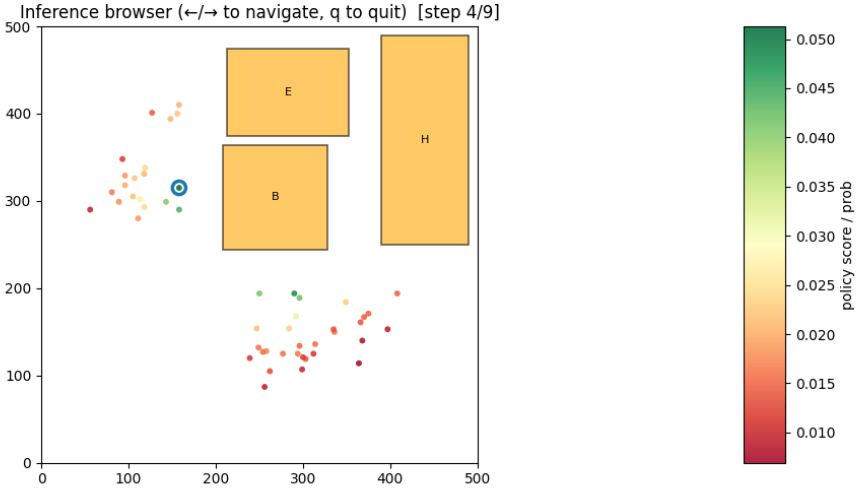
구분	이미지
Step 14	 step=15 | cost=3177.750 | value=-31.777500 | selected_action=5 | selected_policy=0.347024 | valid=49
Step 17	 step=18 | cost=5433.000 | value=-54.330000 | selected_action=5 | selected_policy=0.430264 | valid=38

clearance 가 추가 된 경우

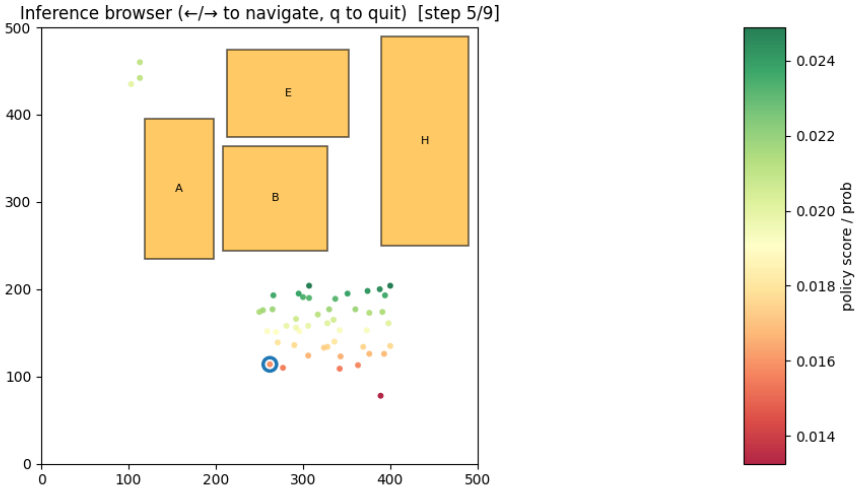
구분	이미지

구분	이미지
Final	
Step 1	

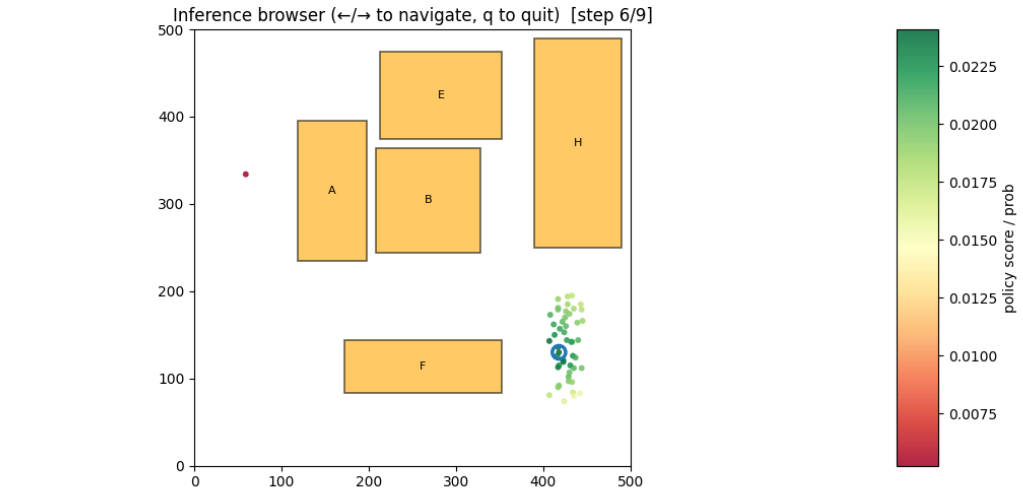
구분	이미지
Step 2	 step=2 | cost=170.000 | value=-1.700000 | selected_action=18 | selected_policy=0.022195 | valid=50
Step 3	 step=3 | cost=264.000 | value=-2.640000 | selected_action=5 | selected_policy=0.084475 | valid=50

구분	이미지
Step 4	

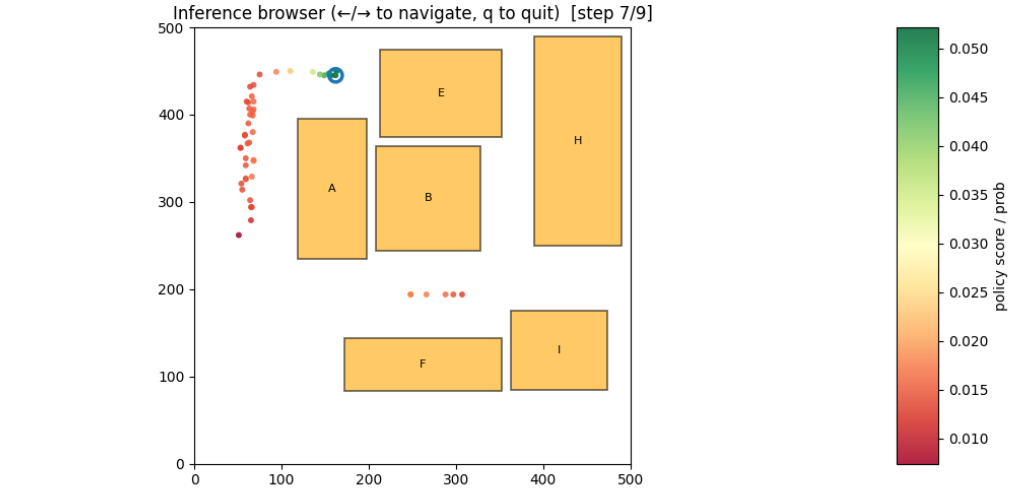
step=4 | cost=444.600 | value=-4.446000 | selected_action=5 | selected_policy=0.051299 | valid=50

구분	이미지
Step 5	

step=5 | cost=615.100 | value=-6.151000 | selected_action=41 | selected_policy=0.015870 | valid=50

구분	이미지
Step 6	

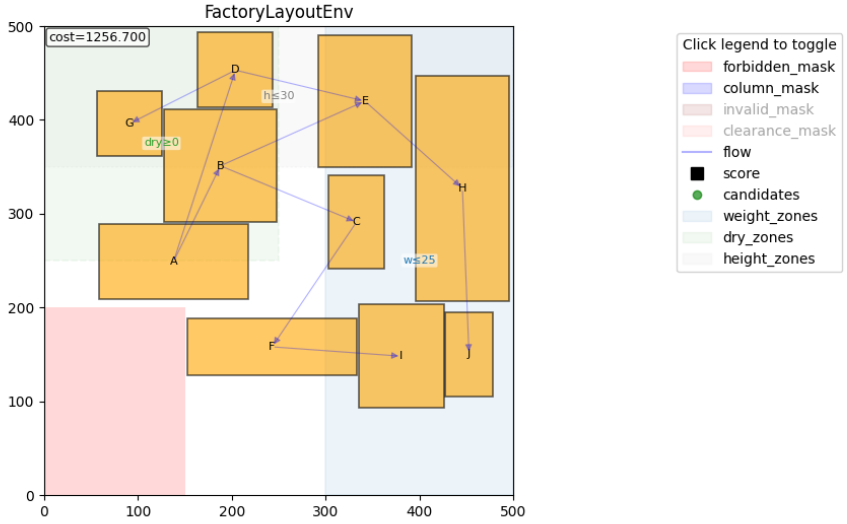
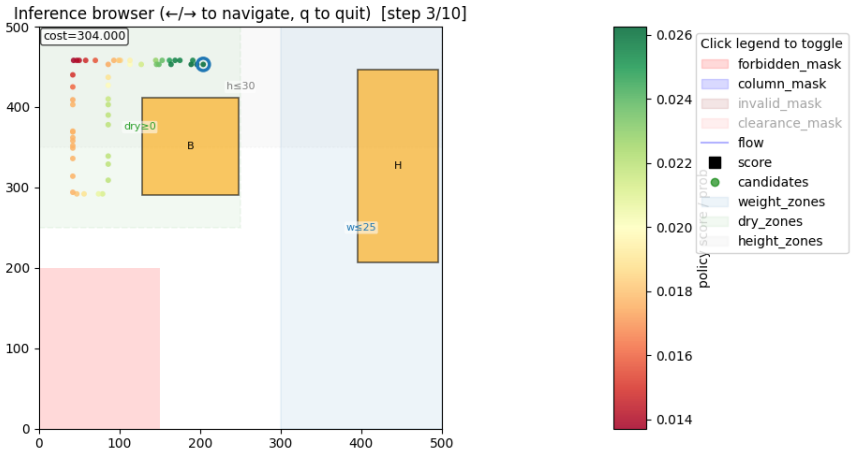
step=6 | cost=690.600 | value=-6.906000 | selected_action=5 | selected_policy=0.024094 | valid=50

구분	이미지
Step 7	

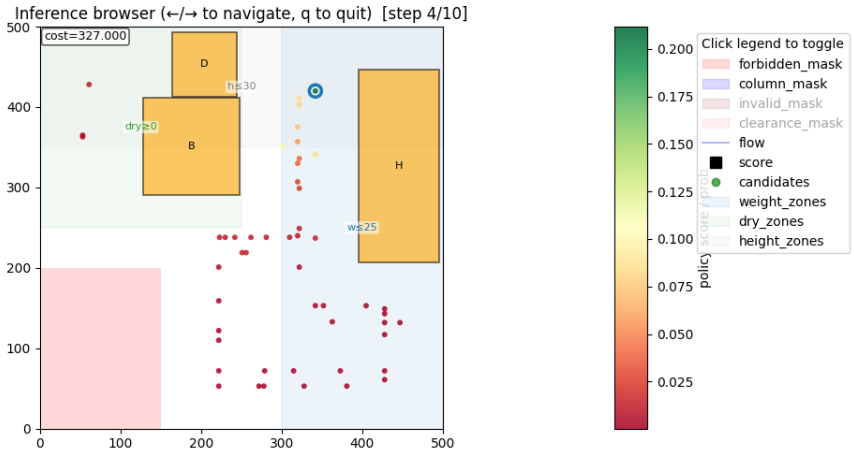
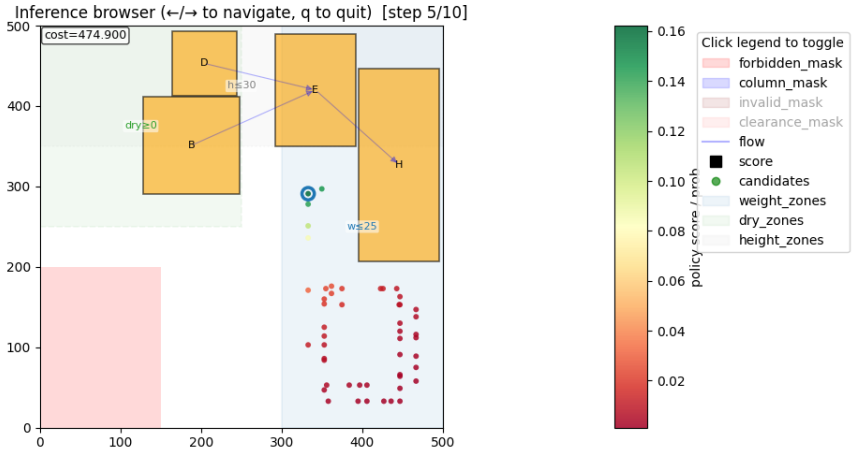
step=7 | cost=759.400 | value=-7.594000 | selected_action=5 | selected_policy=0.052184 | valid=50

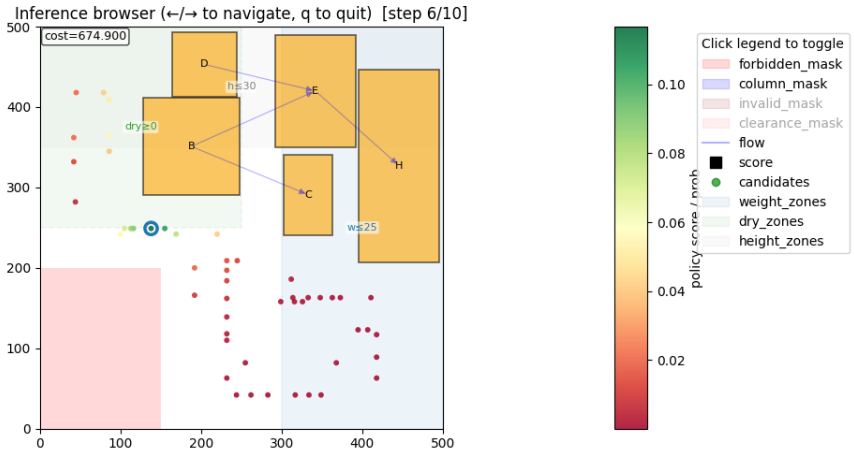
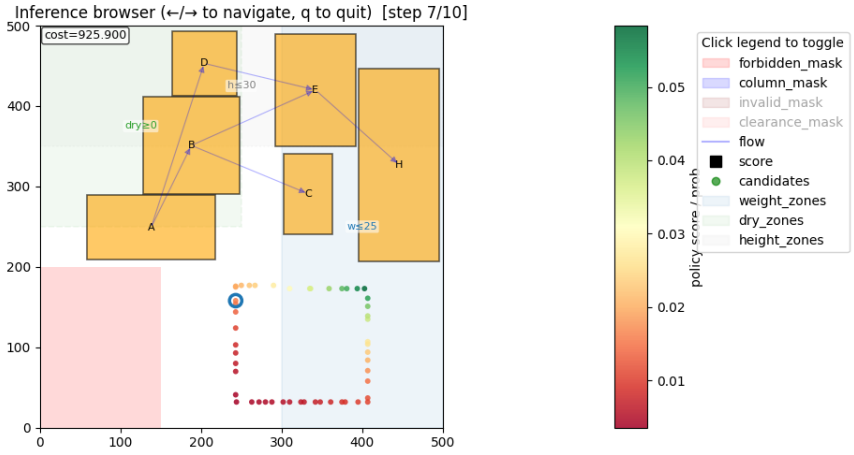
dry, height, weight zone이 추가 된 경우

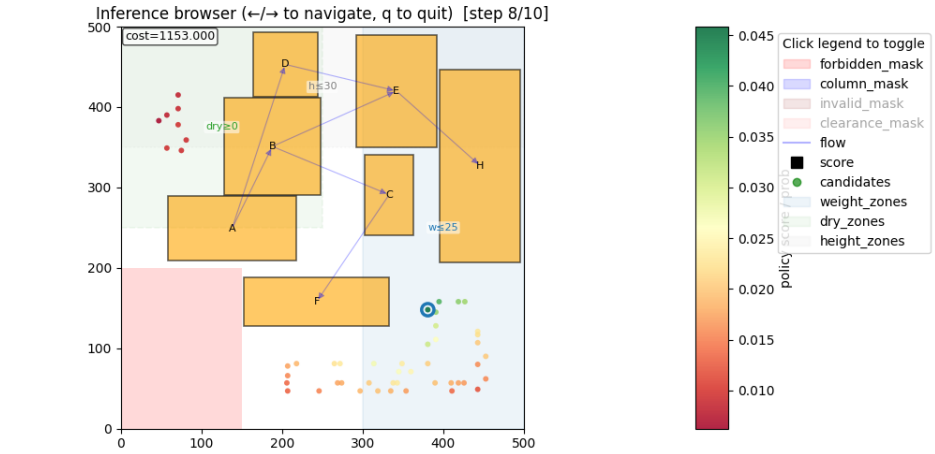
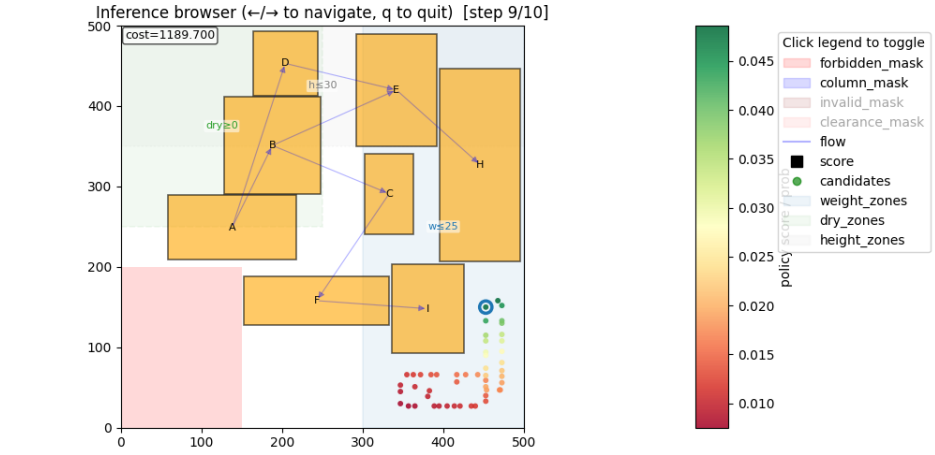
구분	이미지

구분	이미지
Final	
Step 3	


```
step=3 | cost=304.000 | value=-3.040000 | selected_action=42 | selected_policy=0.026260 | valid=50
```

구분	이미지
Step 4	 step=4 | cost=327.000 | value=-3.270000 | selected_action=43 | selected_policy=0.211748 | valid=50
Step 5	 step=5 | cost=474.900 | value=-4.749000 | selected_action=42 | selected_policy=0.162216 | valid=50

구분	이미지
Step 6	 `step=6 | cost=674.900 | value=-6.749000 | selected_action=27 | selected_policy=0.116773 | valid=50`
Step 7	 `step=7 | cost=925.900 | value=-9.259000 | selected_action=2 | selected_policy=0.015906 | valid=50`

구분	이미지
Step 8	 step=8 | cost=1153.000 | value=-11.530000 | selected_action=47 | selected_policy=0.045855 | valid=50
Step 9	 step=9 | cost=1189.700 | value=-11.897000 | selected_action=27 | selected_policy=0.048624 | valid=50

RL 에이전트 도입과 실험

앞을 한 수만 내다보는 greedy agent에 MCTS와 같은 탐색 알고리즘을 결합하면, 롤아웃을 통해 여러 수 이후의 결과까지 고려함으로써 보다 장기적인 의사결정이 가능함을 확인했습니다. 다만 이러한 탐색 기반 접근은 공장 크기가 커지고 배치해야 할 설비 수가 증가할수록, 의미 있는 의사결정을 위해 필요한 시뮬레이션 횟수가 급격히 늘어나 계산 비용이 커진다는 한계가 있습니다. 또한 탐색의 가지수를 줄이기 위해 초기 후보를 greedy가 좋아 보이는 위치 중심으로 제한하면, 탐색 자체가 해당 편향을 그대로 물려받아 여전히 local minima에 빠질 위험이 큽니다.

이러한 문제를 완화하기 위해 greedy 휴리스틱을 강화학습 기반 신경망 정책으로 대체할 수 있습니다. 단 한 수 앞의 즉시 보상만을 기준으로 선택하는 greedy와 달리, 정책 네트워크는 현재 배치 상태와 제약, 그리고 앞으로 배치해야 할 설비들의 정보를 함께 입력으로 받아 장기적으로 유리한 배치를 직접 추정합니다. 즉, 탐색이 모든

경우를 시뮬레이션으로 확인하기 전에, 사람이 경험적으로 유망한 수를 먼저 떠올리는 것처럼 네트워크가 유망한 후보에 prior를 부여해 탐색을 안내하는 역할을 수행한다고 볼 수 있습니다.

도입한 RL 모델은 반도체 설계에 연구되던 모델을 차용하였습니다. 공장 설비 배치 문제는 반도체 플로어플래닝과 구조적으로 유사한 점이 많다고 판단하였기 때문입니다. 칩 설계에서 매크로 간 연결이 배치 품질에 큰 영향을 주는 것처럼, 공장 배치에서도 설비 간 물류 흐름과 다양한 제약 조건이 최종 배치 품질에 직접적으로 작용합니다. 결과적으로 두 문제 모두 점유와 가용이라는 공간 제약 위에서 연결과 흐름 비용을 낮추면서 순차적으로 의사결정을 내려야 하는 조합최적화라는 공통 구조를 갖습니다.

칩 설계에서 대표적인 강화학습 기반 모델로는 구글 계열의 AlphaChip과 NeurIPS 2022의 MaskPlace가 있습니다. 두 모델 모두 배치 과정을 순차적 의사결정 문제로 재구성해 강화학습으로 해를 탐색하는 접근으로 알려져 있습니다. 컴포넌트를 하나씩 배치해 나가는 과정에서, 강화학습 에이전트는 현재까지의 배치 상태를 관측으로 받아 이미 배치된 요소와 남은 요소, 요소 간 연결 관계, 혼잡도와 거리 신호 등을 종합적으로 반영하고 다음에 배치할 요소의 위치를 선택하는 정책을 학습합니다.

두 모델은 같은 순차 의사결정 틀을 공유하지만, 입력 표현과 강점은 다릅니다. AlphaChip은 설비 간 관계를 그래프로 표현하고 이를 GNN으로 요약해 연결 구조를 중심으로 의사결정을 유도하는 성격이 강합니다. 공장 배치로 치환하면 물류 흐름과 거리 최적화에 강점이 있습니다. 반면 MaskPlace는 배치 상태를 고해상도 2D 맵으로 구성하고 이를 CNN으로 처리하며, 유효 위치를 마스킹한 뒤 픽셀 수준에서 정책을 학습하는 접근을 취합니다. 공장 배치로 치환하면 제약 조건이 많아 배치 가능성이 낮은 상황에서 유효 후보를 빠르게 좁히고 안정적으로 의사결정을 내리는 데 유리합니다.

모델	핵심 표현 방식	인코더	의사결정이 집중하는 신호	강점
AlphaChip	설비 간 관계를 그래프로 표현	GNN	무엇이 서로 강하게 연결되어 있는지, 연결 구조 기반의 중요도	물류 흐름, 거리 최적화에 유리. 연결이 강한 설비들을 가깝게 배치하도록 유도
MaskPlace	배치 상태를 고해상도 2D 캔버스로 표현, 픽셀 단위 유효 위치 마스킹	CNN	배치 가능한 위치와 공간의 세밀한 형상, 제약을 만족하는지 여부	제약이 많고 배치가 어려운 문제에서 유리. 유효 후보를 빠르게 좁히고 안정적으로 배치 가능

Alphachip

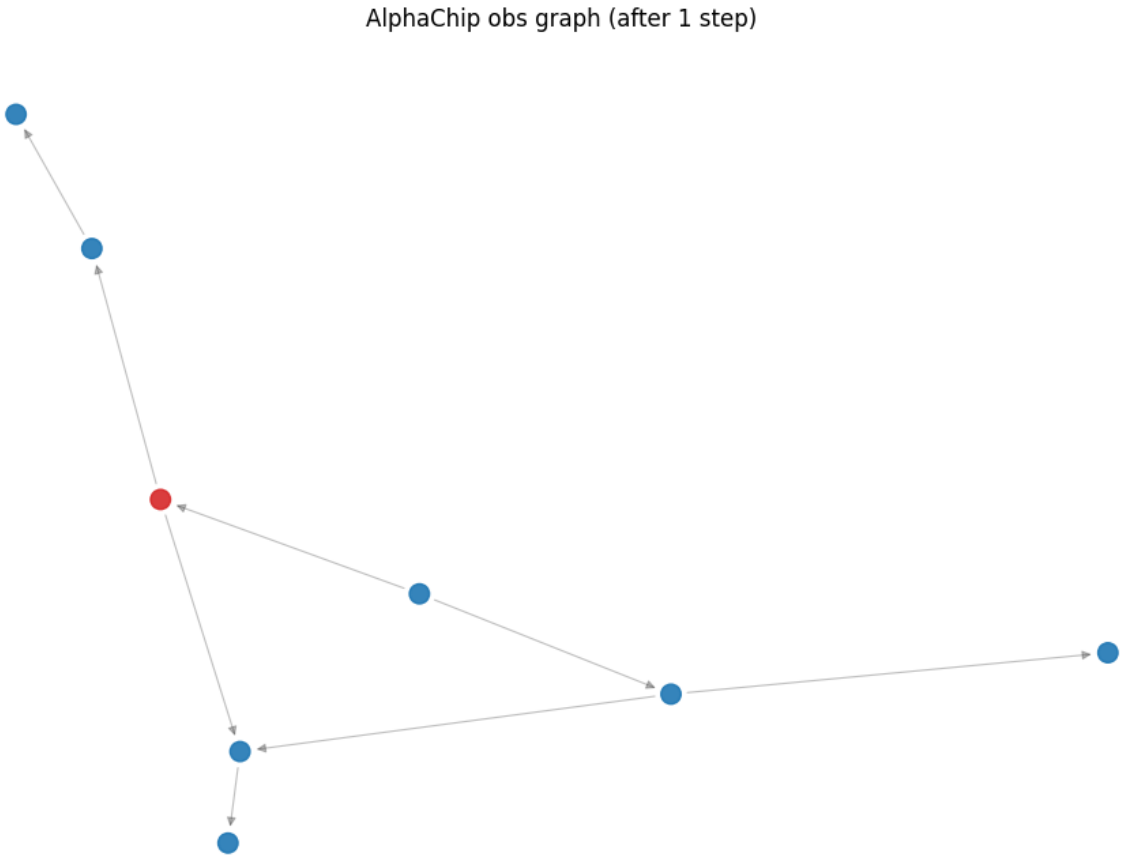
구글 딥마인드에서 공개한 공식 오픈소스 (https://github.com/google-research/circuit_training)는 TensorFlow와 TPU 기반으로 구성되어 있어 범용적인 환경에서 활용하거나 다른 모델 및 파이프라인과 결합하는 데 제약이 있었습니다. 이에 따라 공개 구현을 참고하되, 실행 환경과 확장성을 고려하여 PyTorch와 GPU 기반 코드로 전면 리팩토링을 진행하였습니다.

또한 리팩토링한 구현을 기존에 사용하던 공장 배치 환경과 연동하여, 학습과 평가가 동일한 인터페이스에서 동작하도록 통합하였습니다. 이 과정에서 AlphaChip에서 활용된 범용적인 아이디어 중 하나인 배치 순서를 큰 설비부터 우선 배치하는 전략이 공장 배치뿐 아니라 다른 최적화 모델에도 적용 가능하다고 판단하였고, 이를 환경 코드의 기본 정책으로 반영하였습니다. 추가로 면적이 동일한 설비가 존재하는 경우에는 그래프 연결 구조를 기준으로 우선순위를 결정하도록 하여, 물류 흐름과 연관성이 큰 설비가 먼저 배치되도록 정렬 규칙을 구성하였습니다.

AlphaChip 입력 구성

AlphaChip 계열 접근을 공장 배치 관점에서 정리하면 정책 네트워크 입력은 아래 축으로 구성됩니다.

1. 관계 그래프 요약 입력
    - 노드: 설비 또는 설비 그룹
    - 엣지: 물류 흐름 또는 연결 관계
    - 목적: 어떤 설비끼리 가까워야 하는지 구조적으로 인지하도록 함
2. 현재 배치 상태 입력
    - 이미 배치된 설비의 점유 정보
    - 남은 설비 목록과 속성, 크기, 회전 가능 여부, 제약 요구치
    - 목적: 잔여 공간과 남은 난이도를 함께 반영
3. 공간 신호 입력
    - 거리, 혼잡, 가용성, 제약 존 같은 2D 또는 요약 피처
    - 목적: 여기에 놓았을 때 좋아질지 나빠질지의 힌트를 제공



이 입력을 바탕으로 정책은 다음 설비의 위치 분포를 출력하고, 탐색과 결합하는 경우 이 분포가 prior로 작동하여 트리 확장을 유도합니다.

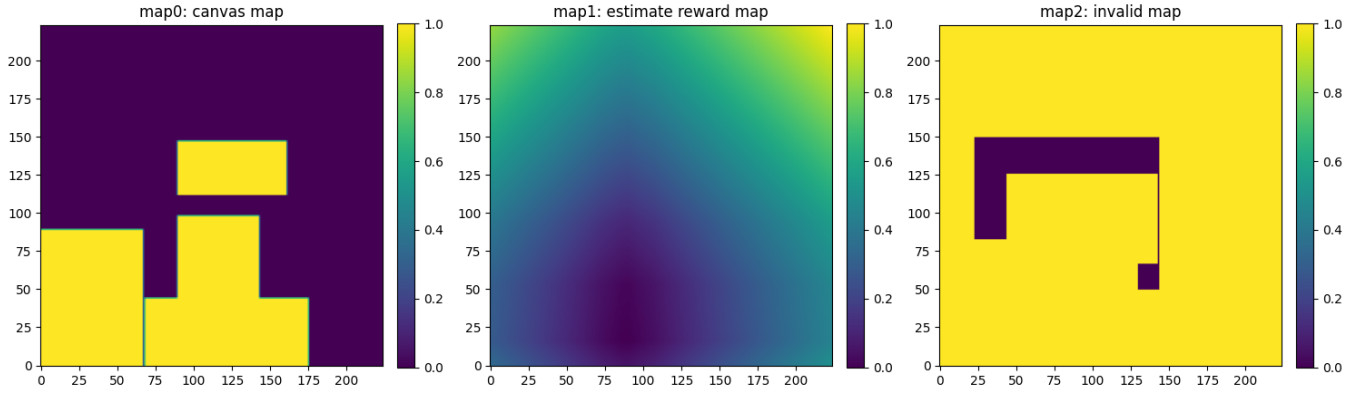
Maskplace

MaskPlace 구현에서는 224×224 해상도의 그리드 캔버스를 기준으로 입력 맵을 구성하고 정책이 유효한 위치 중 어디가 유망한지를 빠르게 판단하도록 설계했습니다. 배치를 픽셀 수준 시각 표현으로 모델링하여 큰 action space에서도 마스킹으로 유효 행동을 다루고 조밀한 학습 신호를 제공하는 점이 핵심입니다.

입력 맵 생성

모델 입력은 3가지 맵으로 구성했습니다.

1. Canvas map
    - 현재까지 배치된 설비의 점유 상태를 나타내는 맵
2. Invalid map
    - 현재 스텝에서 해당 설비를 놓을 수 없는 위치를 표시하는 맵
    - 경계 초과, 충돌, 제약 위반 등을 반영하며 정책 출력에 마스크로 적용
3. Reward map
    - 각 위치에 놓았을 때 보상 변화가 얼마나 발생하는지를 좌표별로 근사한 맵
    - 정책이 단순 가용 공간이 아니라 좋은 자리를 직접 구분하도록 하는 핵심 입력



reward map은 현재 단계에서 설비를 특정 좌표에 배치했을 때 전체 목적함수(물류 비용, 거리, 혼잡 등)가 얼마나 개선되거나 악화되는지를 좌표별로 미리 계산해 둔 지도입니다. 문제는 224×224 모든 후보 좌표에 대해 이 변화를 평가하려면, 각 좌표마다 물류 흐름을 반영한 비용을 다시 계산해야 한다는 점입니다. 만약 이를 flow graph 기반의 최단경로로 정확하게 반영하려고 매 후보마다 Dijkstra를 수행하면, 스텝마다 수만 번의 그래프 탐색이 필요해지고 계산이 순차적 의존성을 가지는 특성상 GPU에서 효율적으로 병렬화하기가 어렵습니다. 결과적으로 reward map 생성이 병목이 되어 학습과 추론 속도를 크게 저하시킵니다.

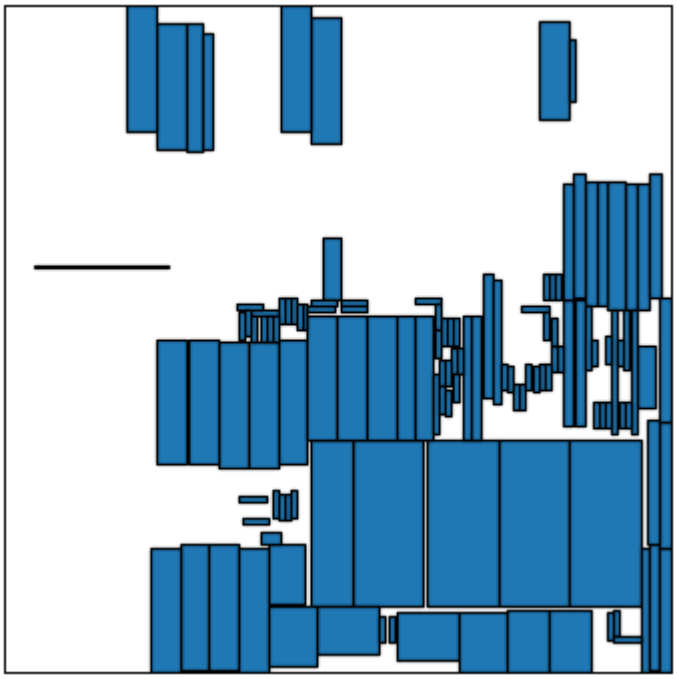
이를 해결하기 위해 reward map 계산을 그래프 최단경로를 정확히 푸는 방식 대신 격자에서 벡터화 가능한 근사 비용으로 재정의했습니다. 구체적으로, 경로 상의 충돌이나 통로 단절 같은 복잡한 요소는 reward map 단계에서는 제외하고, 후보 좌표와 주요 기준점들 사이의 거리를 맨해튼 거리로 근사했습니다. 또한 공장 전체를 그대로 다루지 않고, 핵심 구조만 남긴 compact map 형태로 표현을 단순화하여 거리 및 비용 계산을 텐서 연산으로 병렬 처리할 수 있게 만들었습니다. 이렇게 바꾸면 각 후보 좌표의 비용을 개별 그래프 탐색으로 계산할 필요가 없어지고, 브로드캐스팅과 합산 같은 GPU 친화적 연산으로 한 번에 전체 224×224 reward map을 생성할 수 있습니다.

이 최적화된 파이프라인을 기반으로 reward map 생성부터 후속 입력 맵 구성까지의 전체 과정을 GPU 병렬화하여, 매 스텝에서 100ms 이내에 입력 맵이 생성되도록 구현했습니다. 결과적으로 정책 네트워크는 충분히 촘

촘한 위치별 보상 힌트를 실시간으로 제공받으면서도, 학습 및 탐색 과정의 속도 병목 없이 동작할 수 있었습니다.

사전학습

MaskPlace 공식 구현을 기반으로 RTX 4090 GPU로 약 3일간 사전학습을 수행하였습니다. 공장 레이아웃 배치 문제는 격자 기반의 위치 선택, 유효 행동 마스킹(action masking), 배치 순서에 따른 공간 제약의 누적 등에서 반도체 배치와 구조적으로 유사합니니다. 따라서 반도체 배치 도메인에서 검증된 MaskPlace의 학습 파이프라인을 초기화 단계로 활용하여, 정책이 유효 좌표 선택 및 공간 활용 패턴을 빠르게 학습하도록 했습니니다. Pretrain 결과는 아래와 같으며, 해당 체크포인트를 초기 가중치로 사용하여 공장 배치 환경에서 fine-tuning을 진행하였습니다.



실험

동일한 문제 설정에서 각 방법의 최종 배치 결과를 나란히 비교하기 위한 실험을 진행하였습니다. Greedy는 한 수 앞의 즉시 보상만을 기반으로 의사결정을 내리는 특성상 장기적인 배치 가능성을 확보하기 위해 MCTS 탐색을 결합하여 평가하였으며, 회전 선택도 함께 적용하였습니다.

반면 AlphaChip과 MaskPlace는 정책 네트워크가 유망한 후보에 대한 prior를 출력하므로 향후 MCTS 기반 탐색 및 회전 선택을 결합하는 확장도 가능하지만, 이번 비교에서는 우선 정책 단독으로 유효한 배치를 생성할 수 있는지 확인하는 것을 목표로 하여 탐색과 회전 기능을 제외한 상태로 결과를 산출하였습니다. 따라서 이번 실험은 최고점 기준의 정밀한 성능 비교라기보다는, 각 방법이 현재 설정에서 생성하는 배치의 feasibility와 배치 품질 경향을 비교하는 데 목적을 두었습니다.

Greedy + MCTS	**AlphaChip**	**MaskPlace**

Greedy + MCTS	AlphaChip	MaskPlace