

불확실한 공급망 환경에서 지속 가능한 공급망 최적화 연구

01 연구 배경

글로벌 불확실성으로 인해 공급망 리스크 증가 추세

- 팬데믹, 자연재해, 보호 무역 강화, 기계 고장, 원자재 부족 등 예기치 못한 사건은 공급망 정체에 혼란을 초래
- 이러한 공급망 리스크를 효과적으로 관리하지 않으면 기업의 운영 안정성과 수익성에 부정적인 영향

02 연구 목표

지속 가능한 공급망 구축

- 비용 측면과 공급 리스크 측면을 동시에 고려한 최적의 안전재고 주문량 및 옵션 주문량 결정
- 공급망의 안정성을 유지하면서도 기업의 수익성을 극대화할 수 있는 최적의 조달 및 공급 전략 수립

문제 정의

PROBLEM

안전재고 전략

충분한 안전재고를 확보하면 공급망 리스크를 완화할 수 있지만, 높은 재고 비용 발생

이중 소싱 전략

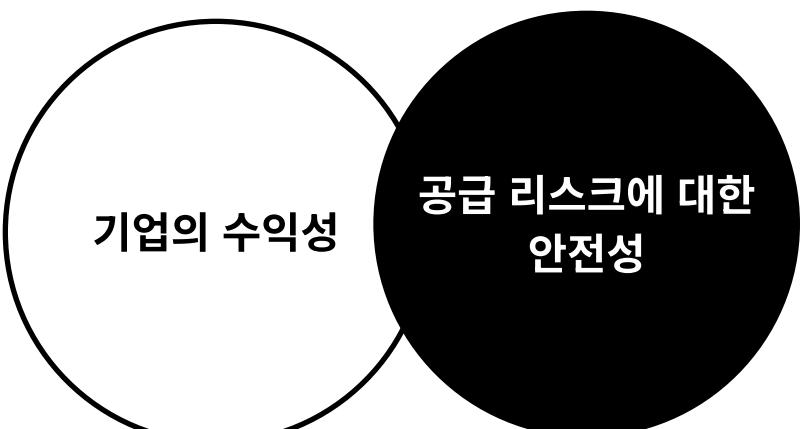
보조 공급업체를 활용하면 공급망 리스크를 줄일 수 있으나, 옵션 계약으로 인해 원가가 상승함

SOLUTION

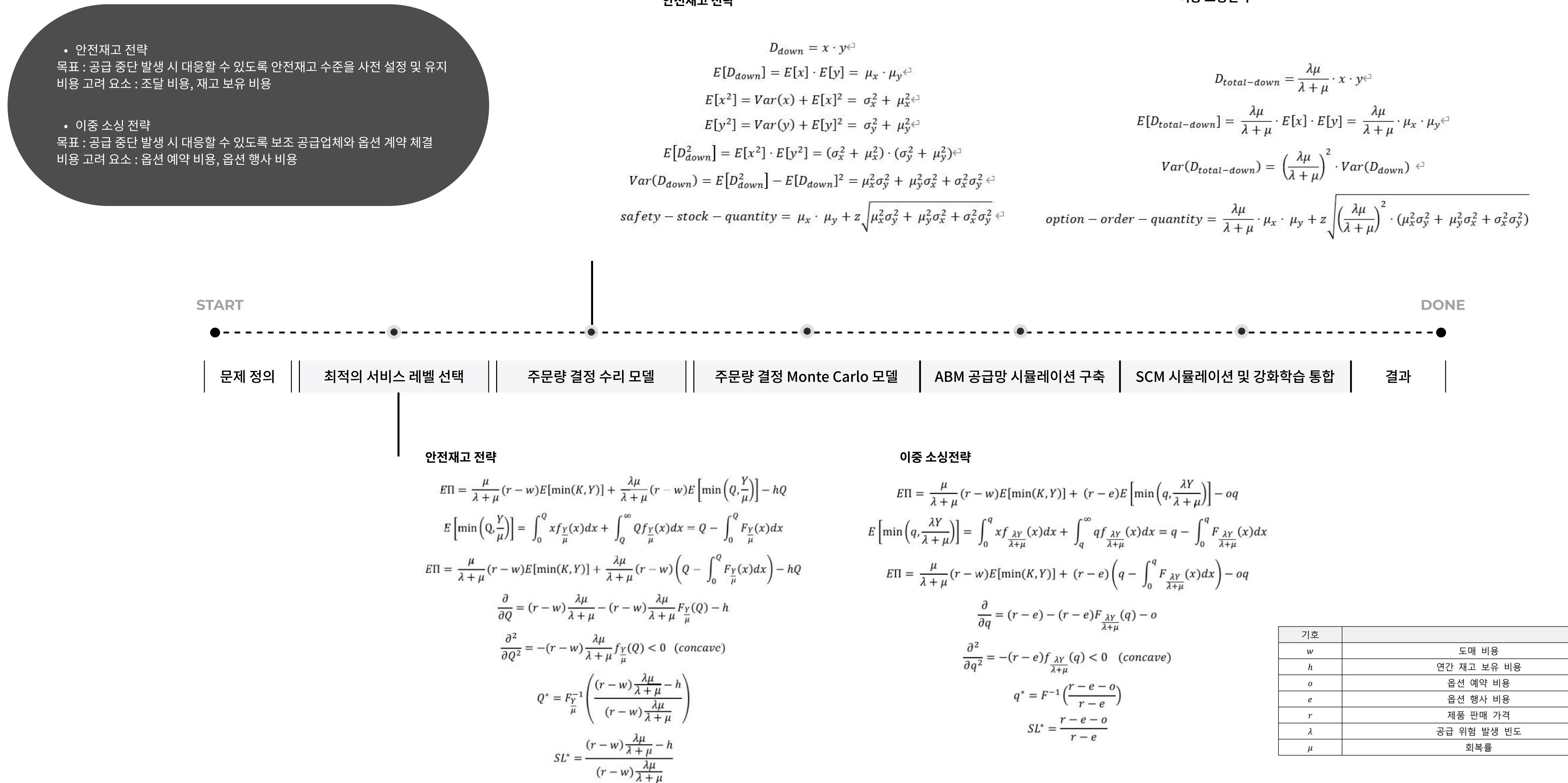
비용 효율성과 공급 리스크를 균형 있게 고려한 적절한 안전재고 주문량 및 옵션 주문량 결정

다양한 공급 리스크 시나리오에 맞는 최적의 조달 전략 탐색

방법론

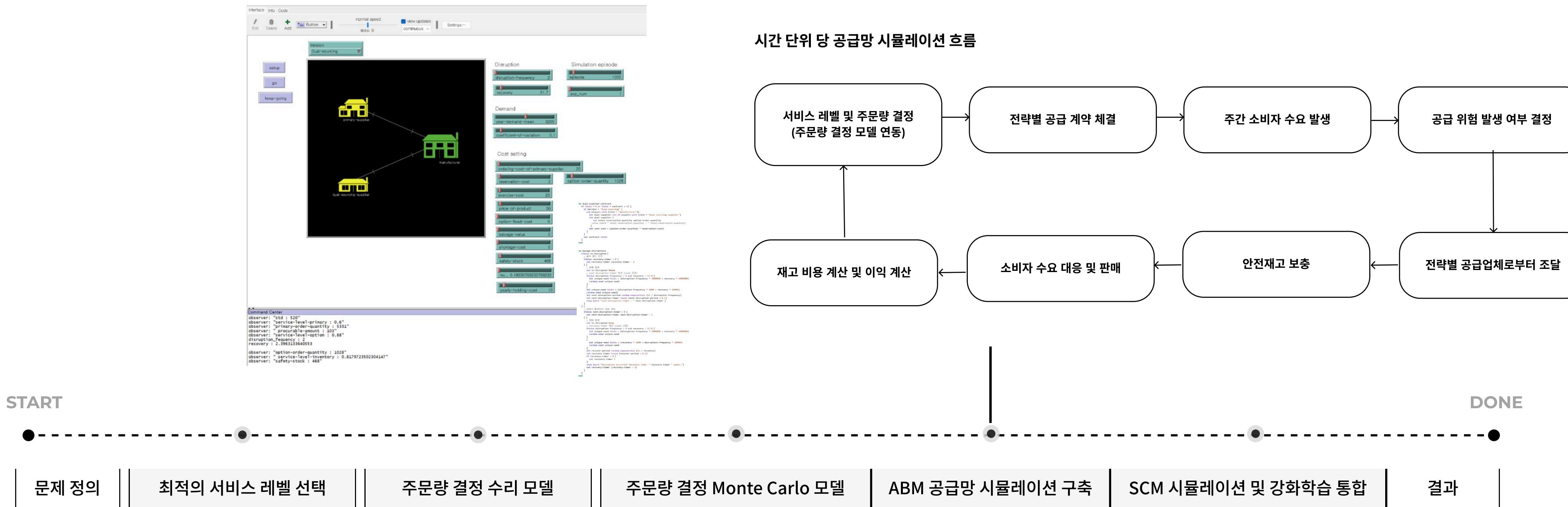


Research Process

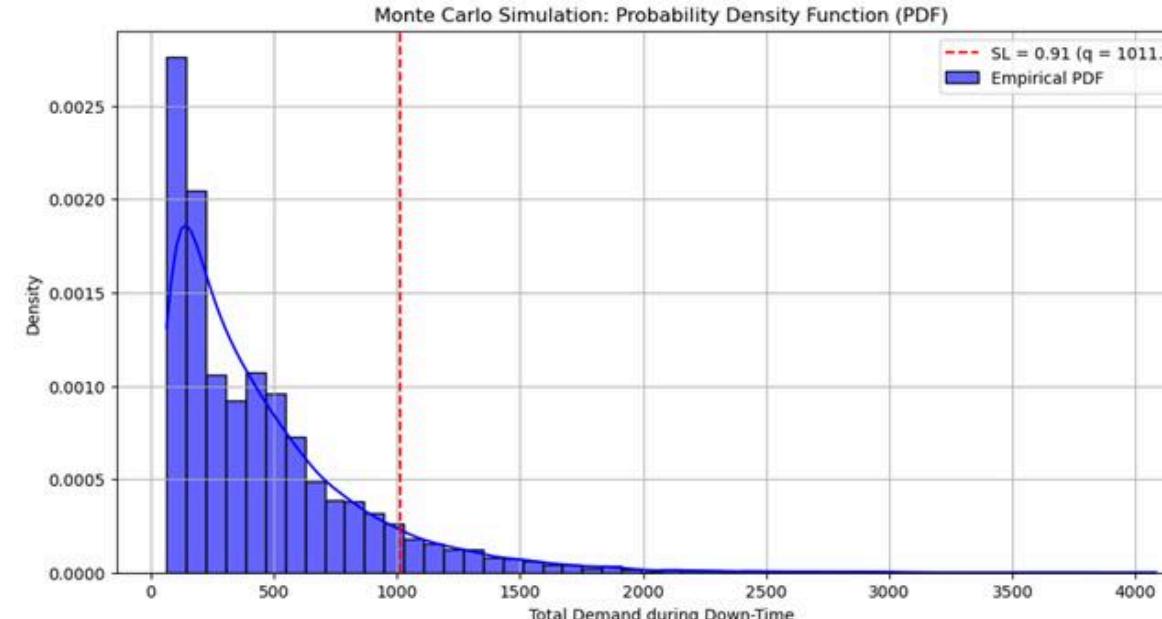


Research Process

공급망 시뮬레이션 환경(NetLogo 6.4.0)



안전재고 전략



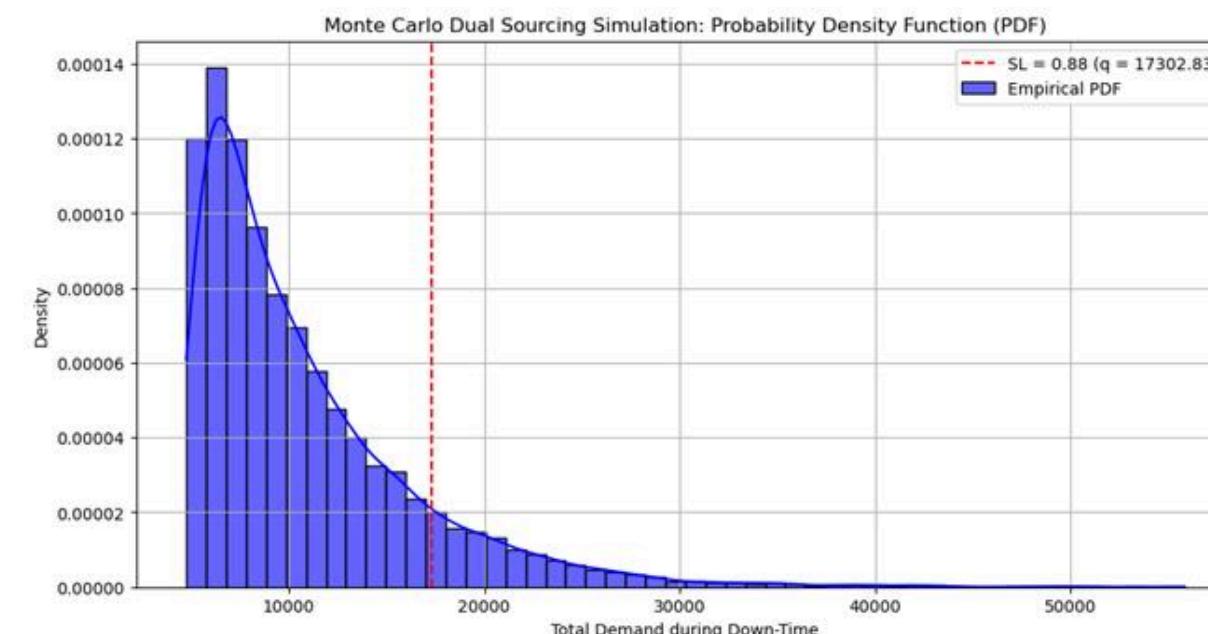
- 수요 : 정규분포
- 공급 중단 길이 : 지수분포

-> 두 분포의 결합을 Monte Carlo Simulation으로 생성

<Monte Carlo Simulation 흐름>

1. 공급 중단 기간 샘플링 : 지수 분포에서 공급망 중단 기간 샘플링
2. 수요 샘플링 : 정규 분포에서 공급 중단 동안의 총수요 샘플링 후 합산
3. 위의 과정 max_step까지 반복
4. 확률 분포 생성 : 반복 샘플링을 통해 분포 도출
5. 최적 안전재고 산출 : 목표 서비스 레벨 기반 백분위수 활용

이중 소싱 전략



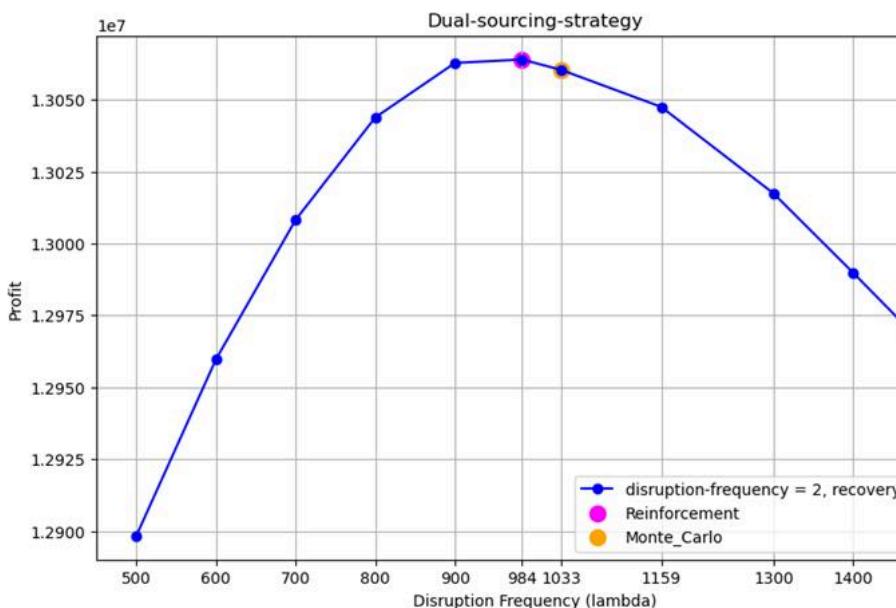
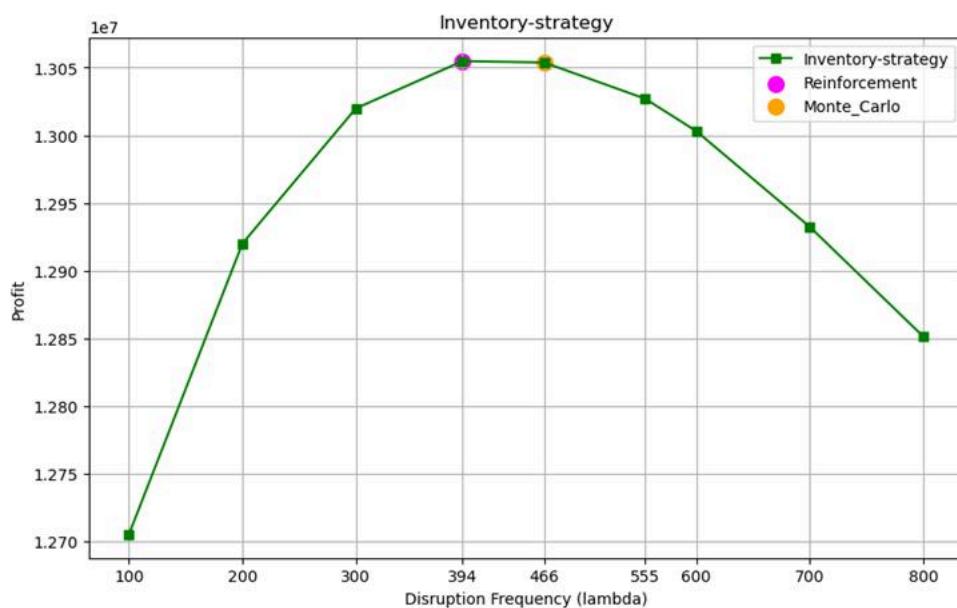
<Monte Carlo Simulation 흐름>

1. 미니 공급망 시뮬레이션 구축 : 단위 기간(52주) 동안 공급 중단 빈도 및 회복률 값이 반영된 공급망 시뮬레이션 실행으로 총 공급 중단 길이 도출
2. 공급 중단 길이 동안의 수요 샘플링 : 정규 분포에서 총공급 중단 동안의 총수요 샘플링 후 합산
3. 위의 과정 max_step까지 반복
4. 확률 분포 생성 : 반복 샘플링 통해 분포 도출
5. 최적 옵션 주문량 산출 : 목표 서비스 레벨 기반 백분위수 활용

Research Process

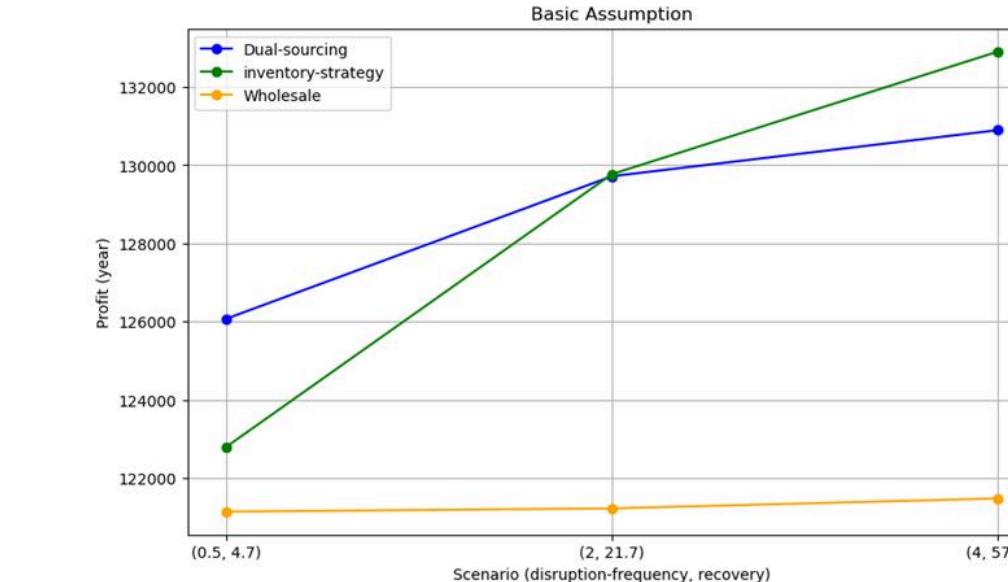
이혜진

1. 시뮬레이션 통합 강화학습을 통해 각 전략별 최적의 주문량 결정



각 조달 및 공급 전략에서 강화학습을 통한 주문량 결정량이 최적의 이익을 주고 있음

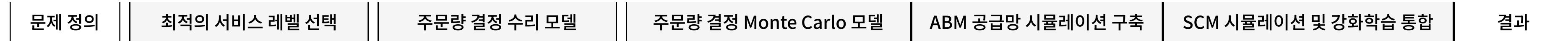
2. 다양한 공급 리스크 시나리오에 따라 최적의 조달 전략 선택 가능



1. 공급 위험이 드물게 발생하지만, 회복 기간이 긴 경우 : 이중 소싱 전략
2. 공급 위험이 자주 발생하지만, 회복 기간이 짧은 경우 : 안전재고 전략

START

DONE



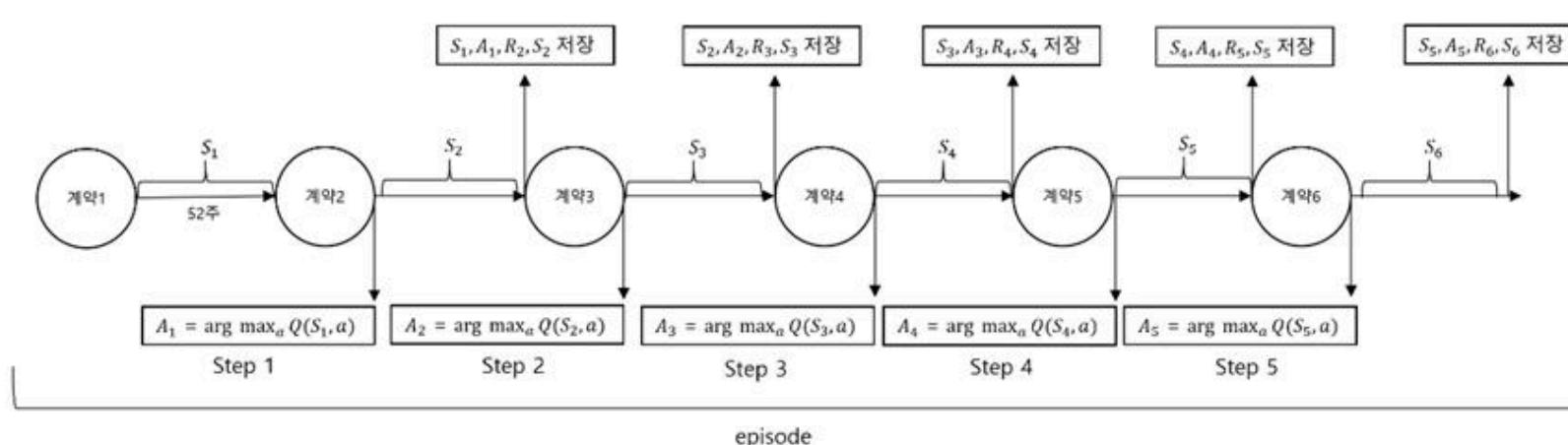
안전재고 전략

- 목표 : 공급 위험에 대비한 최적의 안전재고 수준 결정
- 상태 변수 : 공급 중단 동안 총수요량, 미충족 수요량, 공급 중단 길이, 사용한 안전재고
- 행동 변수 : 안전재고 증가/ 감소/ 유지
- 보상 함수 : 공급 중단 동안의 이익 - 안전재고 보유 비용

이중 소싱 전략

- 목표: 공급 위험에 대비한 최적의 옵션 주문량 결정
- 상태 변수 : 공급 중단 동안 총수요량, 미충족 수요량, 총공급 중단 길이, 사용한 옵션 주문량
- 행동 변수 : 옵션 주문량 증가/ 감소/ 유지
- 보상 함수 : 공급 중단 동안의 이익 - 옵션 예약 비용

DQN 강화학습 통합 시뮬레이션 흐름



Reward 기록



ABM 시뮬레이션 기반 수요 예측 모델 개발 및 공급망 최적화 연구

01 연구 배경

수요 예측 실패로 과잉 생산 및 폐기

02 연구 목표

부족한 데이터 환경에서도 수요 예측 성능을 향상시키는 방법론 개발

- 스마트팜 기업은 신선 제품 유통 과정에서 수요 예측 실패로 인한 과잉 생산 및 폐기 문제를 겪고 있음
- 하지만, 초기 단계의 기업으로 1년 치의 제한된 수요 데이터만 보유하고 있음

- 스마트팜 공급망을 시뮬레이션 상에 구현해 가상의 소비자 행동 데이터 생성으로 부족한 데이터 총족
- 스마트팜 생산, 운송, 재고 측면에서 최적화를 통해 수익성 극대화

문제 정의

기존 수요 예측 모델은 수요 데이터가 부족하여 예측 성능이 저하

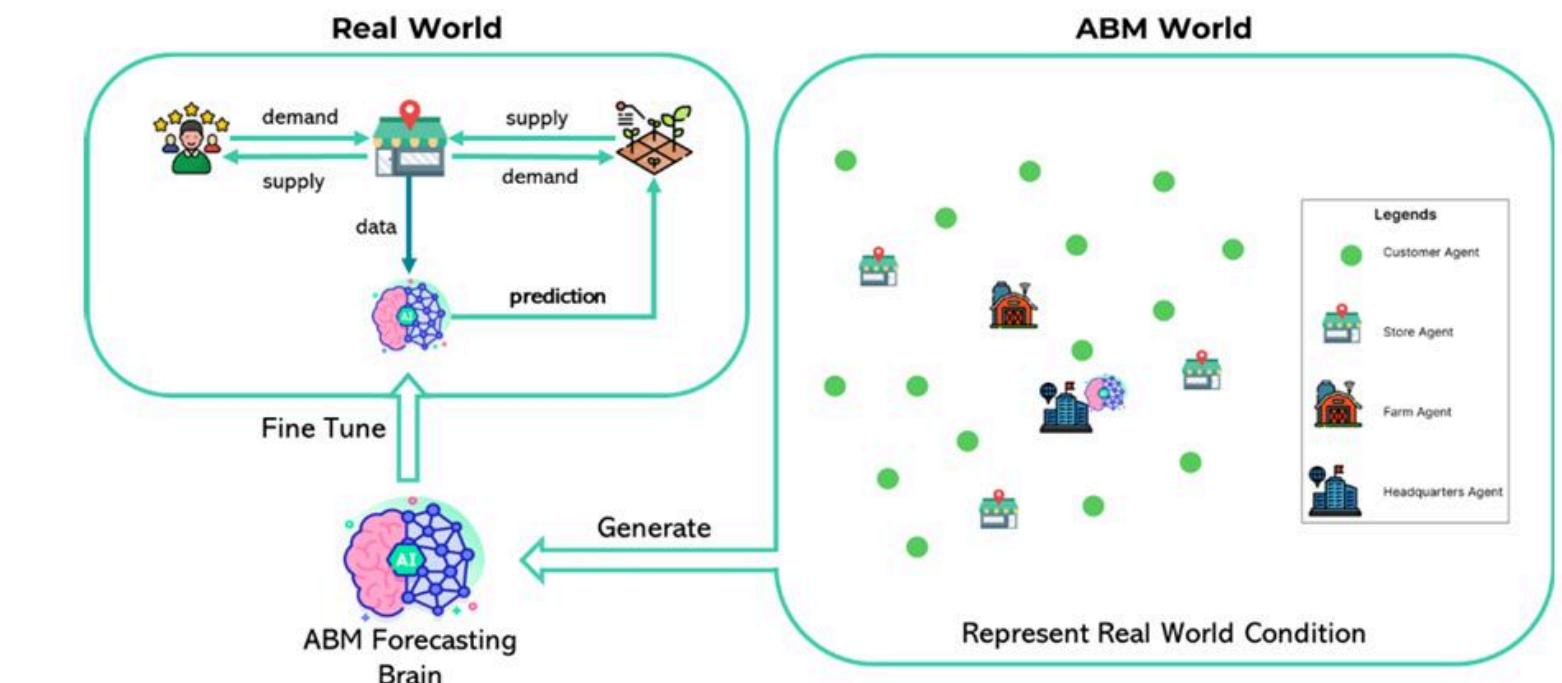
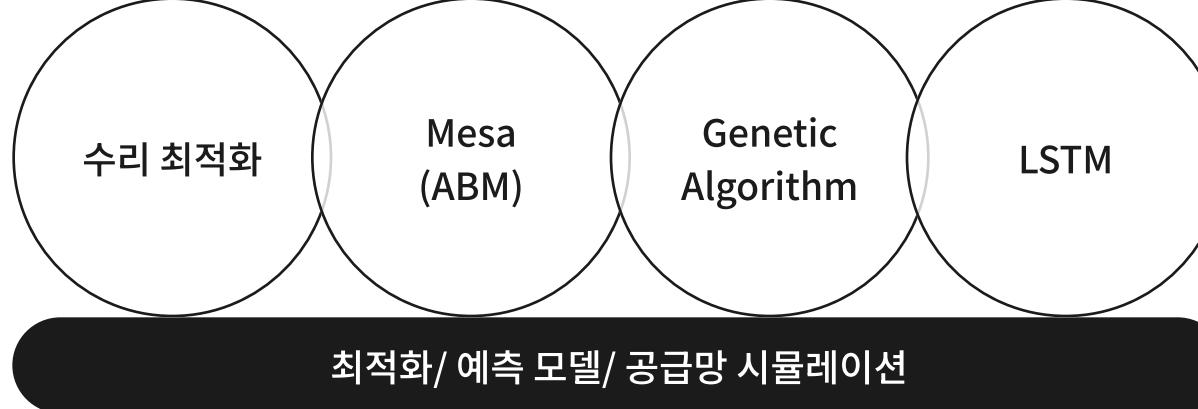


최적의 생산 계획 부재

데이터를 보완하기 위해 ABM 시뮬레이션을 활용하여 현실적인 수요 데이터를 생성하고, 이를 머신 러닝 모델과 결합하여 예측 성능 개선

예측된 수요를 기반으로 생산 능력, 생산 리드타임과 운송 거리를 고려한
최적의 생산 계획 수립

방법론



Research Process

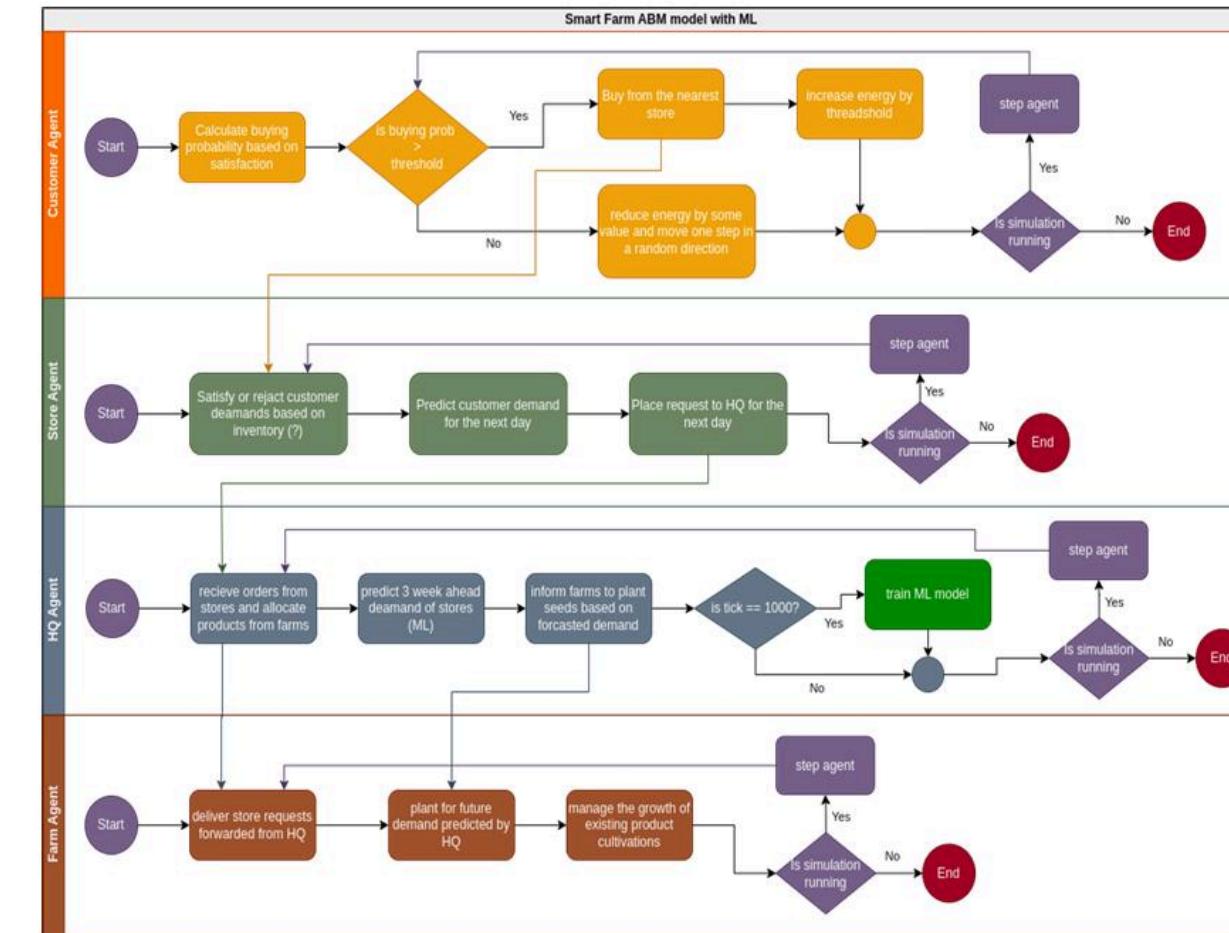
이혜진

실제 Mesa 시뮬레이션 구현



START

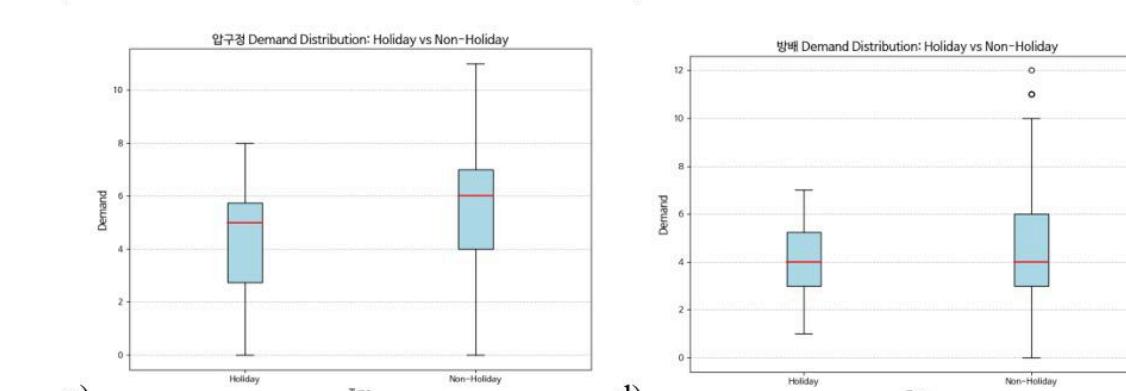
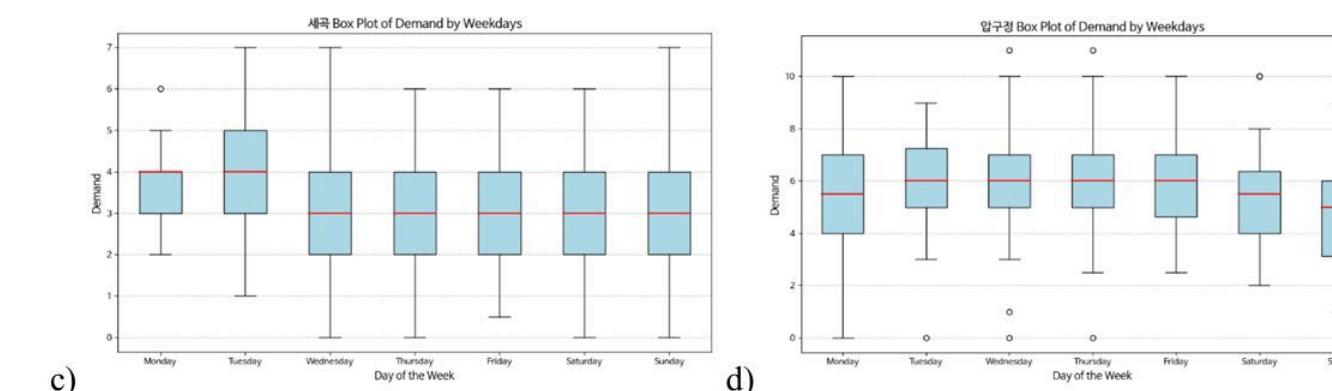
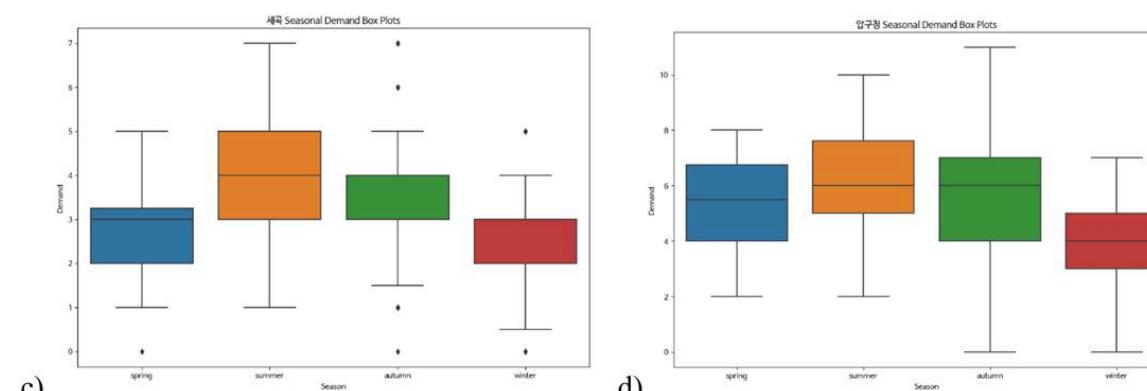
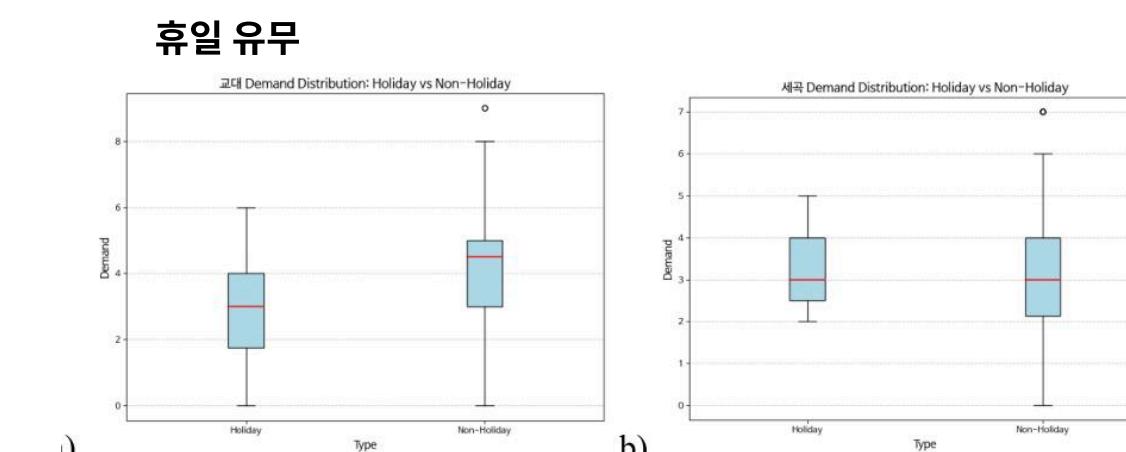
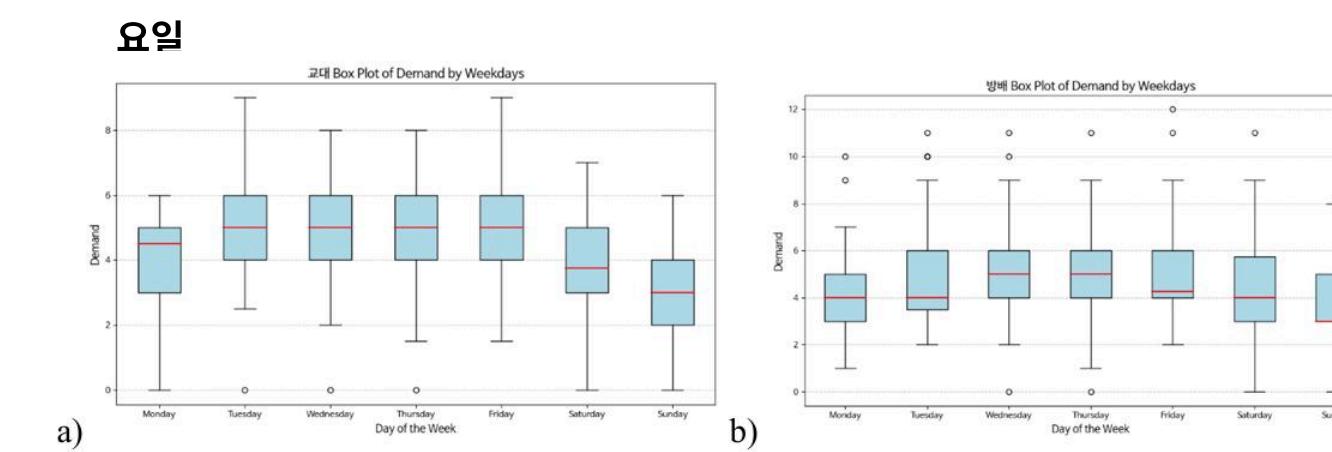
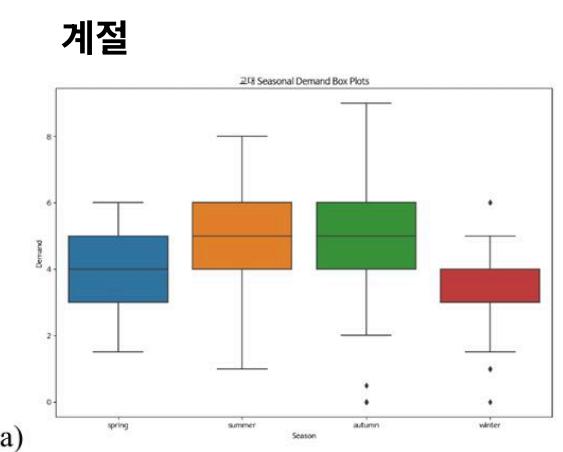
각 에이전트의 규칙



DONE



- 4개의 스토어 별로 각 계절, 요일, 휴일별 수요 패턴 분석
- 이후, ABM 시뮬레이션에 분석 내용 통합



- Python 기반 ABM 시뮬레이션인 Mesa, MesaGeo 이용

1. MesaGeo를 이용해 시뮬레이션 환경 설정 (서울시 지도)
2. 소비자, 스토어, 농장, 본사 에이전트 정의 및 위치

<소비자 구매 파라미터>

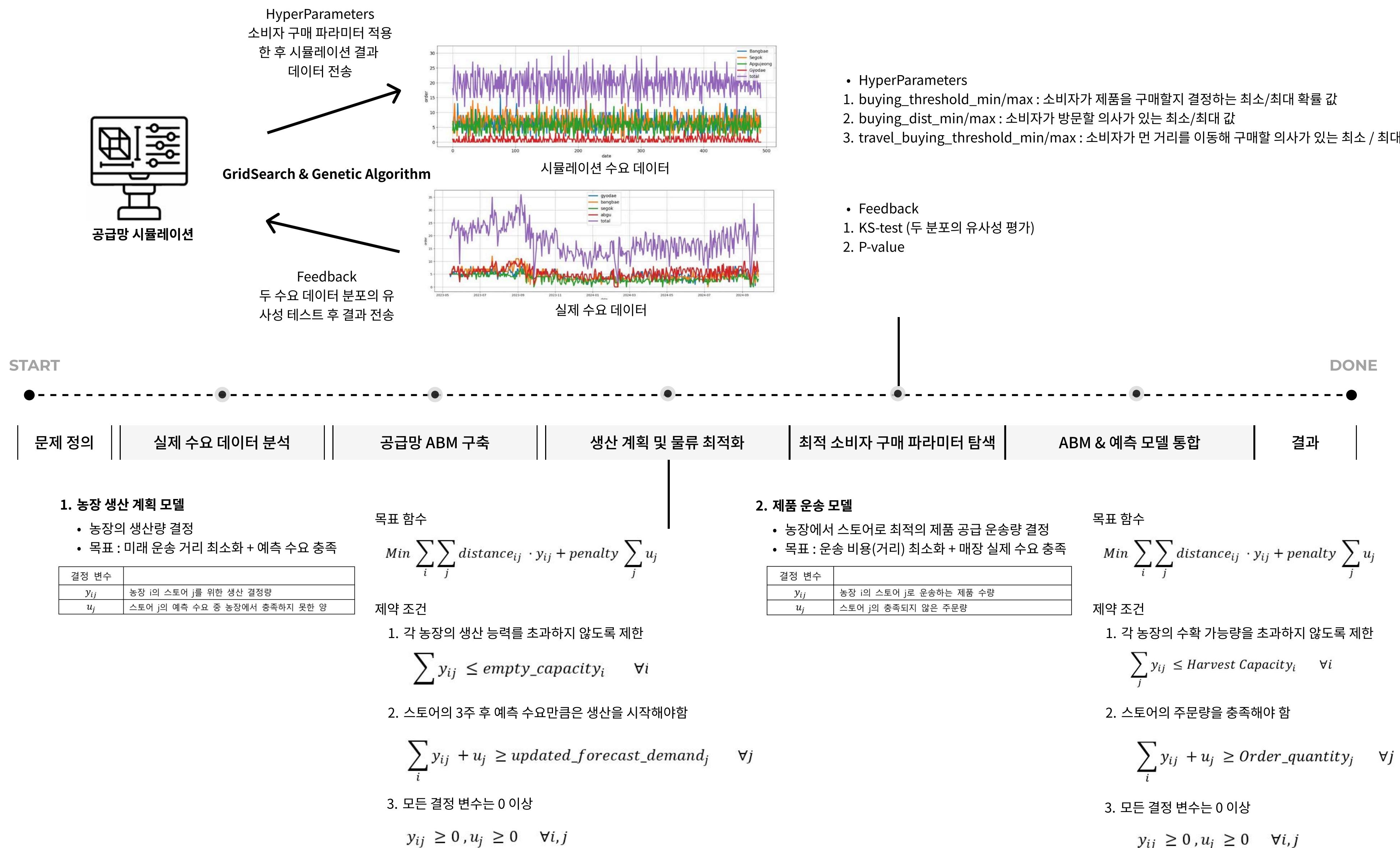
buying_threshold_min/max : 소비자가 제품을 구매할지 결정하는 최소/최대 값

buying_dist_min/max : 소비자가 방문할 의사가 있는 최소/최대 거리

travel_buying_threshold_min/max : 소비자가 먼 거리를 이동해 구매할 의사가 있는 최소/최대 확률값

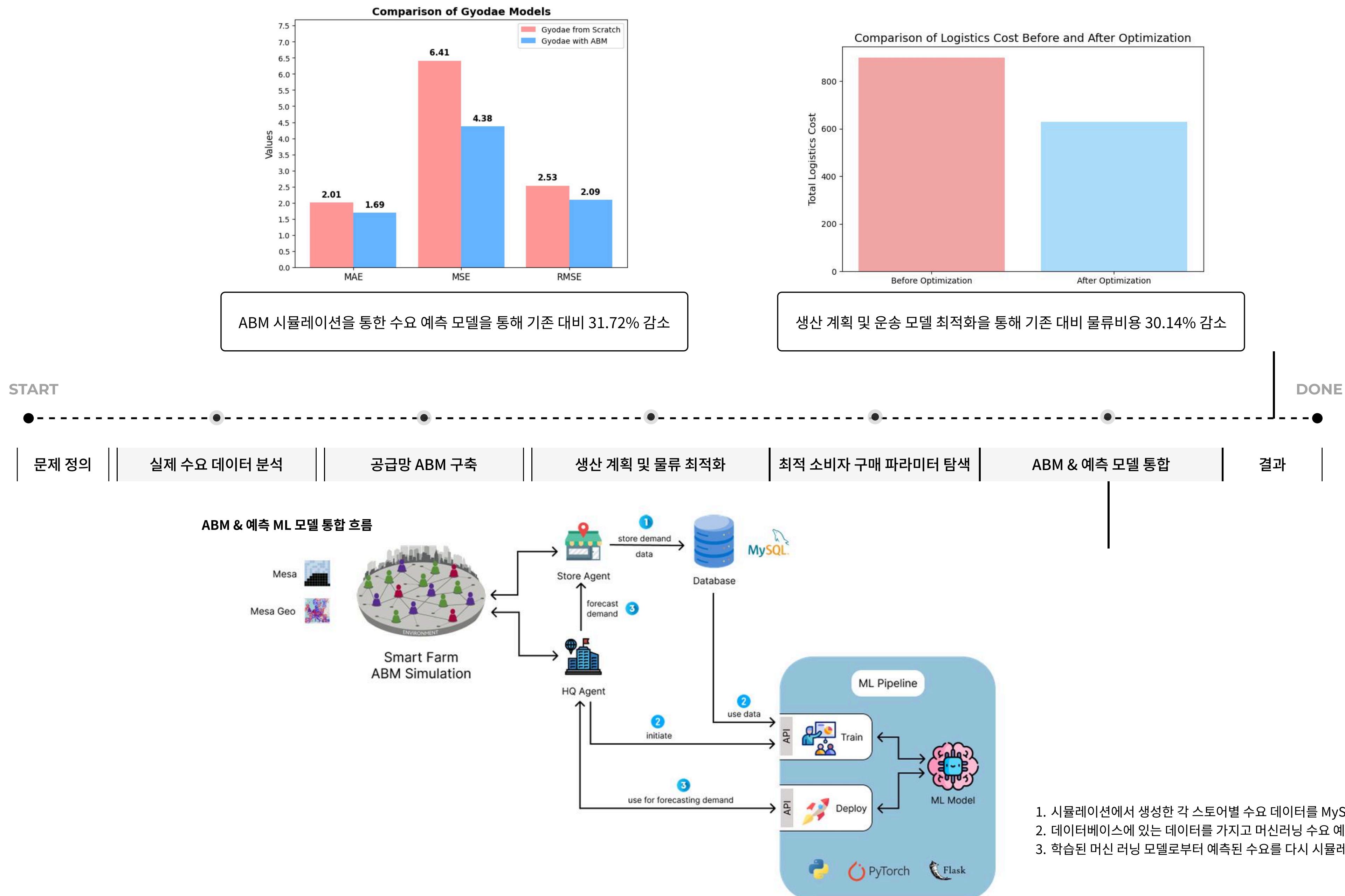
Research Process

이혜진



Research Process

이혜진



생산성과 안전성을 고려한 항만 야드 컨테이너 적재 최적화 연구

01 연구 배경

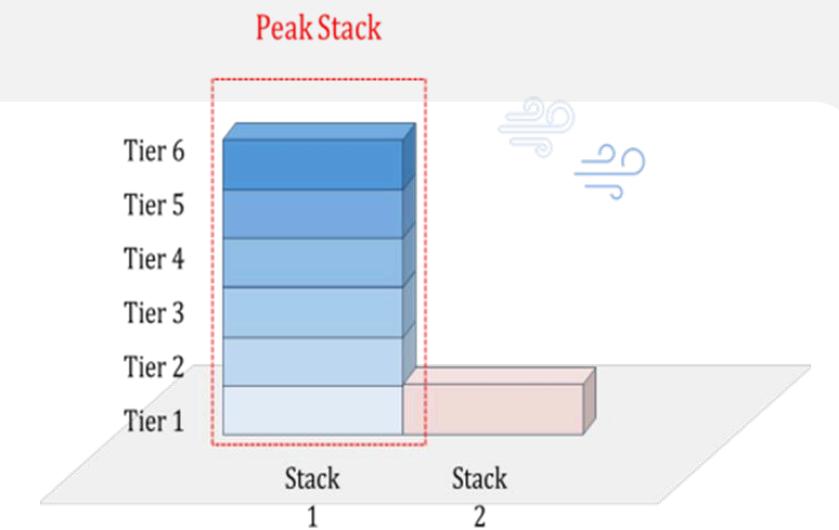
피크스택의 위험성 증가

- 기존 연구는 공간 효율성 및 생산성 측면에만 집중
- 현재 항만 컨테이너 야드장에서 피크스택(혼자 우뚝 서 있는 스택)으로 컨테이너 전도 위험 증가

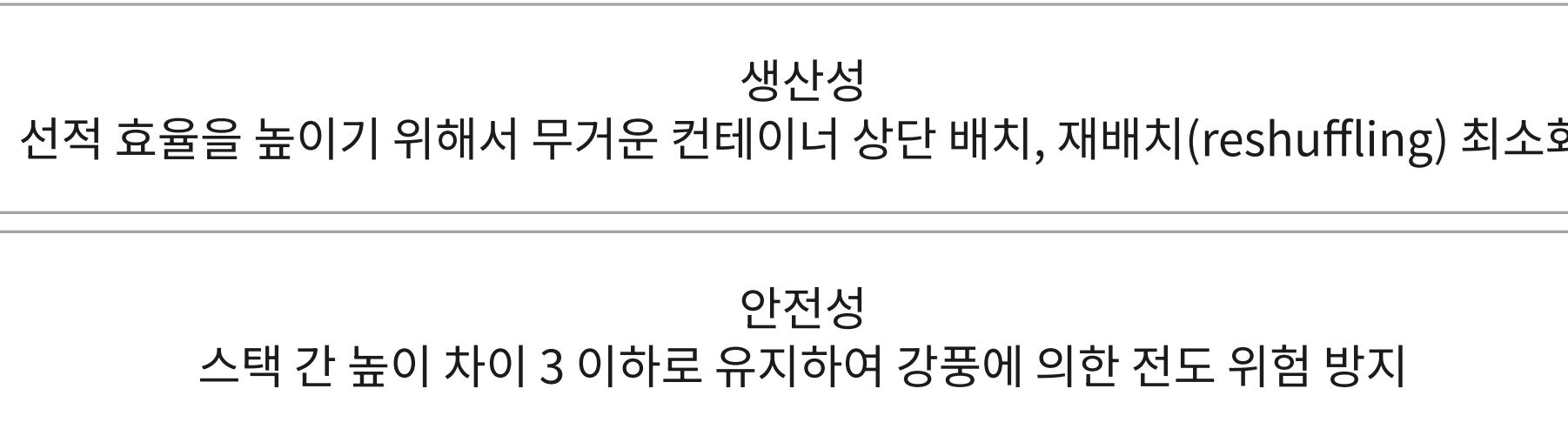
02 연구 목표

생산성과 안전성 동시에 고려

- 생산성과 안전성을 고려한 컨테이너 적재 최적화 형상 및 방법 개발
- 야드장 컨테이너 적재 시, 피크스택 발생 확률 감소

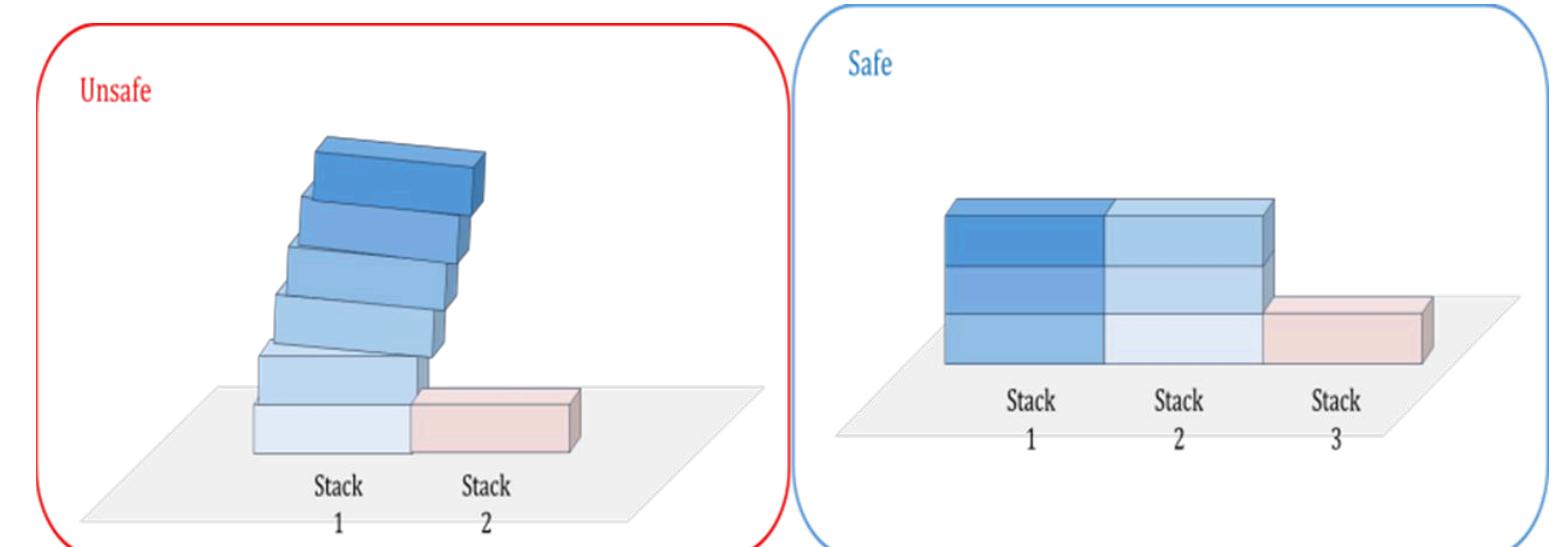
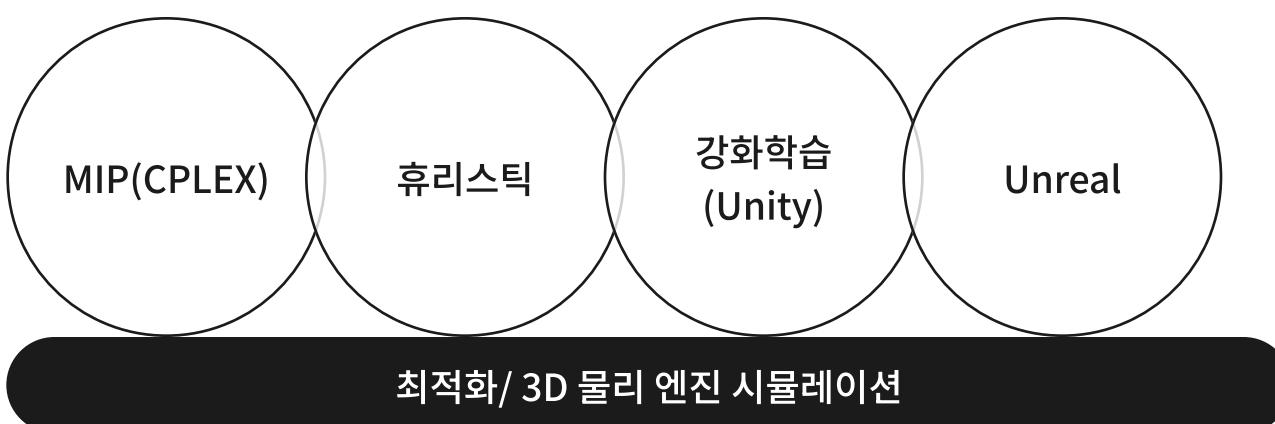


문제 정의



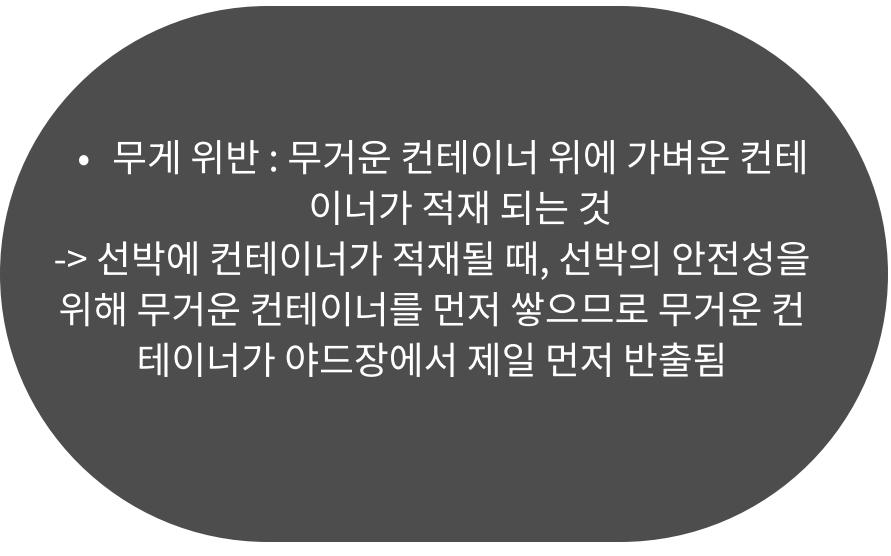
생산성과 안전성을 고려한 항만 야드 컨테이너 적재 계획

방법론



Research Process

이혜진



step 1 : 초기 상태 재정렬

- 목표 : 야드장에 이미 쌓여 있는 초기 컨테이너를 무게 위반과 무게 차이를 고려해서 한 쪽 스택으로 스택킹함으로서 새롭게 들어오는 컨테이너 스택킹 공간 확보

8				
6				
6				
3	4	5	7	8

<Before>

8				
6	7			
6	5			
3	4			

<After Initial Container relocation>

List of the container at the top of each stack :

[4, 5, 7, 8] -> [5, 7, 8, 8] -> [7, 8, 8]
-> [6, 8, 8] -> [6, 8, 8] -> [6, 8]

List of the container at the top of each stack에서

첫번째 컨테이너의 무게(4) 선택해서 무게 레벨 차이 계산

- Stack 1 : -4
 - Stack 2 : 4
 - Stack 3 : 4
 - Stack 4 : -1
 - Stack 5 : -3
 - Stack 6 : -4
- ⇒ Stack 2,3 have the smallest difference.

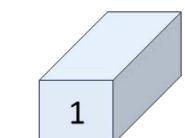
Stay on stack 2.

step 2 : 새로운 컨테이너 배치

- 목표 : 피크 스택 발생을 제한하면서 무게 차이와 무게 위반을 최소화

8				
6	7			
6	5			
3	4			

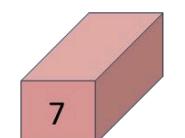
<Before weight 1 arrives>



New container arrive(weight 1)
[-5, None, 1, 1, 1, -5] -> [1, 1, 1, 1, -5]

8				
6	7			
6	5			
3	4	1		

<After weight 1 arrives>



New container arrive(weight 7)
[1, None, 6, 7, 7, 7] -> [1, 6, 7, 7, 7]

8				
6	7			
6	5			
3	4	1		

<After weight 7 arrives>

START

DONE

문제 정의

기하학적 중심을 이용한 MIP

휴리스틱 1

휴리스틱 2

Unity 시뮬레이션 ML-agent

Unreal 강풍 시뮬레이션

결과

Parameters

Parameter	Description
n	Number of Containers
m	Number of stacks
h	Capacity of tiers
g	Number of groups for priority
δ	Max tier for safety (peak stack height)
$i \in \{1, \dots, n\}$	Index of container
$j \in \{1, \dots, m\}$	Index of stack
$k \in \{1, \dots, h\}$	Index of tier
W	Set of weights
w_i	Weight of container i
w'_i	Weight of container i with priority
$s_i \in \{1, \dots, n\}$	Sequence of container i (when container is loaded)
$g_i \in \{100, \dots, g\}$	Priority of container i (when container is unloaded)
(\bar{x}_i, \bar{y}_i)	Geometric center of container i
M	Big number
$e_i \in \{0, 1\}$	Emergency status of container i
I	Info of initial containers
d_{min}	Minimum value of distance
d_{max}	Maximum value of distance

Variables

Variable	Description
x_{ijk}	$\begin{cases} 1 & \text{if container } i \text{ is on tier } k \text{ of stack } j, \forall i, j, k \\ 0 & \text{otherwise} \end{cases}$
r_{jk}	$\begin{cases} 1 & \text{if a container on tier } k \text{ of stack } j \text{ needs to relocate, } \forall j, k \\ 0 & \text{otherwise} \end{cases}$
d_i	Distance from (x_i, y_i)
x_i	Distance from \bar{x}_i
y_i	Distance from \bar{y}_i

Objective function

$$\text{Minimize } \alpha \times \sum_{j=1}^m \sum_{k=1}^h r_{jk} + \beta \times \sum_{i=1}^n \frac{d_i - d_{min}}{d_{max} - d_{min}}$$

Constraints

1) Container i must be loaded in one place.

$$\sum_{j=1}^m \sum_{k=1}^h x_{ijk} = 1, \forall i$$

2) Only one container can be loaded in a slot.

$$\sum_{i=1}^n x_{ijk} \leq 1, \forall j, k$$

3) The height of stack j must be less than or equal to h .

$$\sum_{k=1}^h \sum_{i=1}^n x_{ijk} \leq h, \forall j$$

4) You cannot stack slot k into slot $k+1$ with slot k empty.

$$\sum_{i=1}^n x_{ijk} \geq \sum_{i=1}^n x_{ij,k+1}, \forall j, 1 \leq k \leq h-1$$

5) Define d_i

$$d_i = x_i + y_i, \forall i$$

$$d_i = \sum_{j=1}^m \sum_{k=1}^h j \times x_{ijk} - \bar{x}_i, \forall i$$

$$x_i \geq \left(\sum_{j=1}^m \sum_{k=1}^h j \times x_{ijk} - \bar{x}_i \right), \forall i$$

$$y_i \geq \sum_{j=1}^m \sum_{k=1}^h k \times x_{ijk} - \bar{y}_i, \forall i$$

$$y_i \geq \left(\sum_{j=1}^m \sum_{k=1}^h k \times x_{ijk} - \bar{y}_i \right), \forall i$$

10) Set the origin location of the initial containers

$$x_{i,j,k} = 1, i, j, k \in I$$

목적 함수 : reshuffling 최소화 + 안전한 형상으로부터 가까도록 위치

$$\left(\sum_{i=1}^n w'_i \times x_{ijk} - \sum_{i=1}^n w'_i \times x_{ij,k+1} \right) / M \leq M \times \left(1 - \sum_{i=1}^n x_{ij,k+1} \right) + r_{jk}, \forall j, k < h$$

$$r_{jk} \leq M \times \left(1 - \sum_{i=1}^n x_{ij,k+1} \right) + r_{j,k+1}, \forall j, k < h$$

$$\sum_{i=1}^n x_{ijk} \geq r_{jk}, \forall j, k$$

9) Sequence

$$\sum_{i=1}^n s_i \times x_{ijk} \leq M \times \left(1 - \sum_{i=1}^n x_{ij,k+1} \right) + \sum_{i=1}^n s_i \times x_{ij,k+1}, \forall j, k < h$$

$$x_{i,j,k} = 1, i, j, k \in I$$

```
# MIP_P
# N = 1000
# decision Variables
x = model.binary_var_dict([(i,j,k) for i in range(n) for j in range(m) for k in range(h)], name = 'x')
r = model.binary_var_dict([(j,k) for j in range(m) for k in range(h)], name = 'r')
d_x = model.continuous_var_dict([(i) for i in range(n)], name = 'd_x')
d_y = model.continuous_var_dict([(i) for i in range(n)], name = 'd_y')

# # Constraint 0 : Define w_prime
# w_max = max(weight)
w_prime = []
for i in range(n):
    w_prime.append(w_max + p[i])
# constraint 1 : container i must be assigned to one stack and on tier
for i in range(n):
    model.add_constraint(sum(x[i,j,k] for j in range(m) for k in range(h)) == 1)

# constraint 2 : one slot can only have one container
for j in range(m):
    for k in range(h):
        model.add_constraint(sum(x[i,j,k] for i in range(n)) == 1)

# constraint 3 : the height of stack j must be less than or equal to h
for j in range(m):
    model.add_constraint(sum(x[i,j,k]*k for i in range(n)) <= h)

# constraint 4 : you can't stack a container on slot k if there is no container on slot k-1
for j in range(m-1):
    for k in range(h-1):
        model.add_constraint(sum(x[i,j,k] for i in range(n)) > sum(x[i,j,k+1] for i in range(n)))

# constraint 5 : prevent peak stacks
for j in range(m):
    model.add_constraint(sum(x[i,j,k] for k in range(h)) <= peak_init)

# constraint 6 : prevent slot limit
model.add_constraint(d_x[i] >= sum(x[i,j,k]*j for j in range(m) for k in range(h)) - center_d_x)
model.add_constraint(d_y[i] >= sum(x[i,j,k]*k for j in range(m) for k in range(h)) - center_y)
model.add_constraint(d_x[i] == x[i] + d_y[i])

# constraint 7 : prevent peak stacks
for j in range(m):
    for k in range(h-1):
        model.add_constraint((sum(w_prime[i]*x[i,j,k] for i in range(n)) + sum(x_prime[i]*x[i,j,k] for i in range(n))) <= h * (1 - sum(x[i,j,k] for i in range(n)) - peak_limit))

# constraint 8 : define r_jk
for j in range(m-1):
    for k in range(h-1):
        model.add_constraint(r[j,k] >= sum(x[i,j,k] for i in range(n)) + sum(x_prime[i]*x[i,j,k] for i in range(n)) - r[j,k])

# constraint 9 : sequence
for j in range(m-1):
    for k in range(h-1):
        model.add_constraint(sum(x[i,j,k] for i in range(n)) > r[j,k])

# constraint 10 : emergency
for j in range(m):
    for k in range(h-1):
        model.add_constraint(sum(e[i]*x[i,j,k] for i in range(n)) <= h * (1 - sum(x[i,j,k] for i in range(n)) + sum(x[i]*x[i,j,k] for i in range(n))))
```

구성한 MIP 모델을 CPLEX를 통해 모델링

Research Process

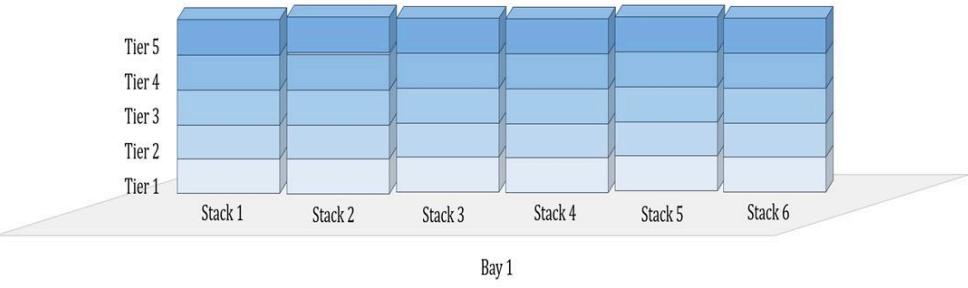
이혜진

ML-Agent with Unity (강화학습)

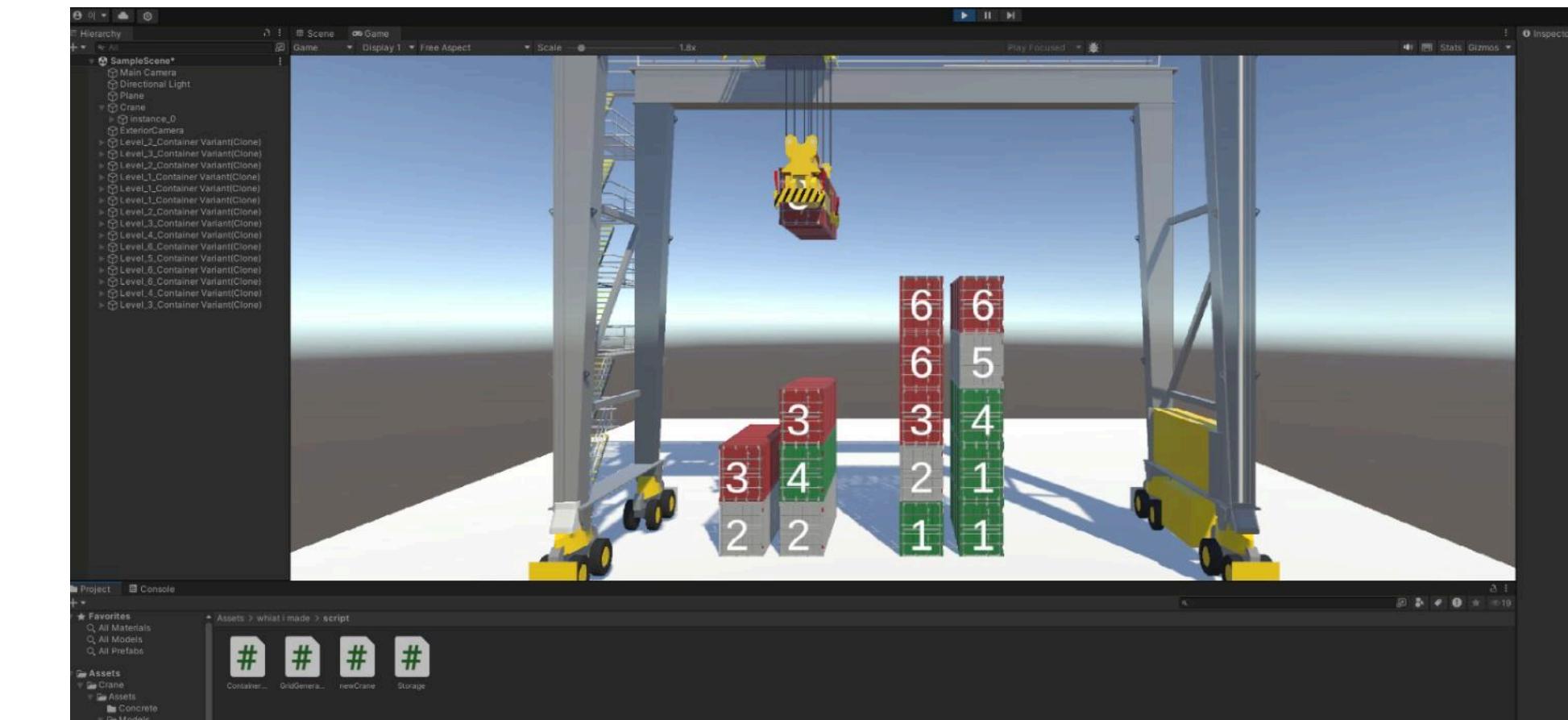
<Action>
야드 크레인의 그림이 stack 1에서 stack 6 사이로 이동과 동시에 잡고 있던 컨테이너를 drop

<Observation>
space size : 30
Container Grid[i,j] : 현재 bay에 컨테이너가 쌓여있는 grid

<Reward>
기본 보상 : 그립에 있는 컨테이너를 일정 시간 내에 bay에 배치하지 않으면 (-) 보상
피크스택 보상 : 컨테이너 쌓는 과정에서 피크 스택이 발생하면 (-) 보상
무게에 따른 보상 : 종료 조건이 만족된 후 스택별로 무게 위반이 있으면 (-) 보상
무게 위반이 없으면 무게 차이에 따라 차등적으로 (+) 보상



START



DONE

실제 구현 시뮬레이션

문제 정의 || **기하학적 중심을 이용한 MIP** || **휴리스틱 1** || **휴리스틱 2** || **Unity 시뮬레이션 ML-agent** || **Unreal 강풍 시뮬레이션** || **결과**

step 1 : 이상적인 형상 생성

- Version 1을 먼저 사용하다가, 무게 차이 규칙이 적용할 수 없을 만큼 자리가 부족하다면, Version 2로 전환

Version 1
한 스택 내에서 위아래로 무게 차이가 3 이상나면 다음 위치에 배치
무게레벨 : [1(3), 5(2), 6,7,8,9]


Version 2
단순히 무게 수준에 따라 지그재그 패턴으로 배치
계레벨 : [1(5), 4(3), 5(5), 6(4), 7(3), 8(5), 9]


step 2 : 초기 상태 재정렬

- 상태 플래그 관리 : 각 컨테이너가 무게 레벨에 맞는 이상적인 위치에 있는지를 True/False로 분류
- 초기 컨테이너 재배치 대상 : 각 스택 상단에 위치한 False 상태의 컨테이너들 설정

각 스택의 최상단에 위치한 False 상태의 컨테이너가 여러 개 있을 경우
1순위: 무게 레벨이 작은 컨테이너, 2순위: 같은 무게 레벨일 때는 더 높은 티어에 위치한 컨테이너

1/T	5/F	6/F	
3/F	1/T	5/F	7/F

■ : 재배치 대상 컨테이너
■ : 우선순위 1
■ : 우선순위 2

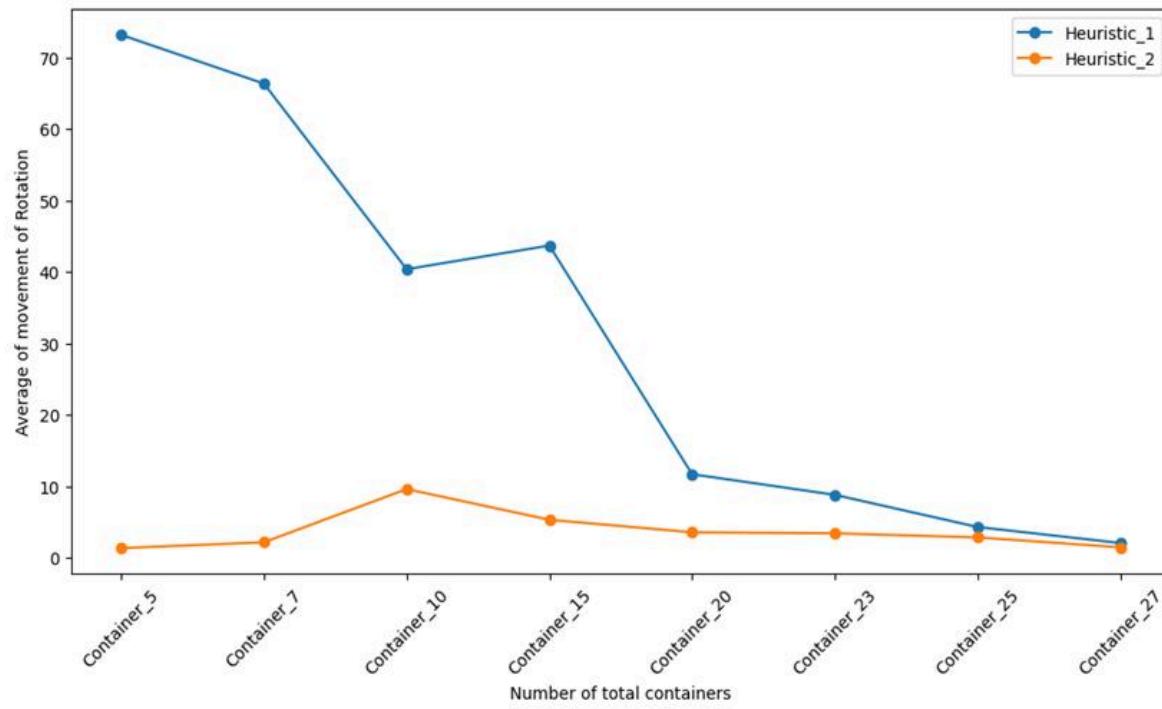
step 3 : 새로운 컨테이너 배치

- 컨테이너 무게 레벨에 맞는 이상적인 위치가 비어 있다면 해당 위치 배치 후 True 플래그 부여
- 배치할 컨테이너 무게 레벨이 각 스택의 최상단에 있는 컨테이너의 무게 레벨보다 크거나 같은 스택으로 이동
- 스택 높이가 가장 낮은 곳에 배치

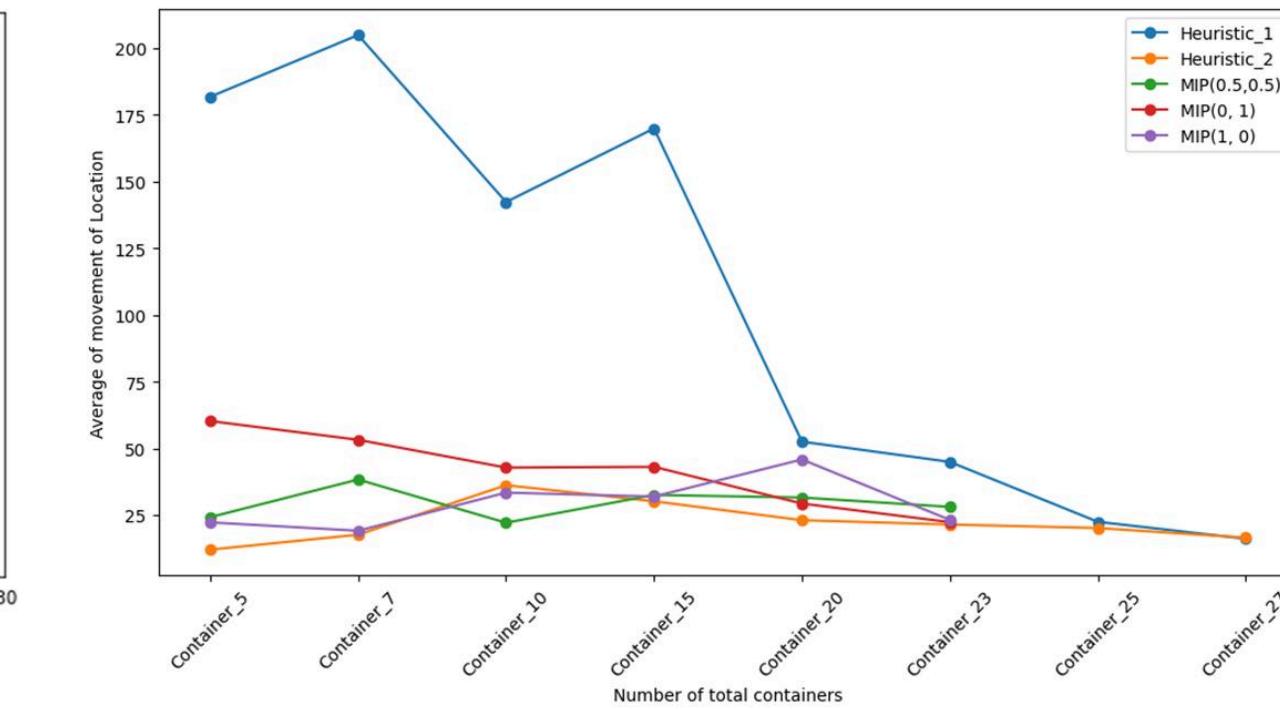
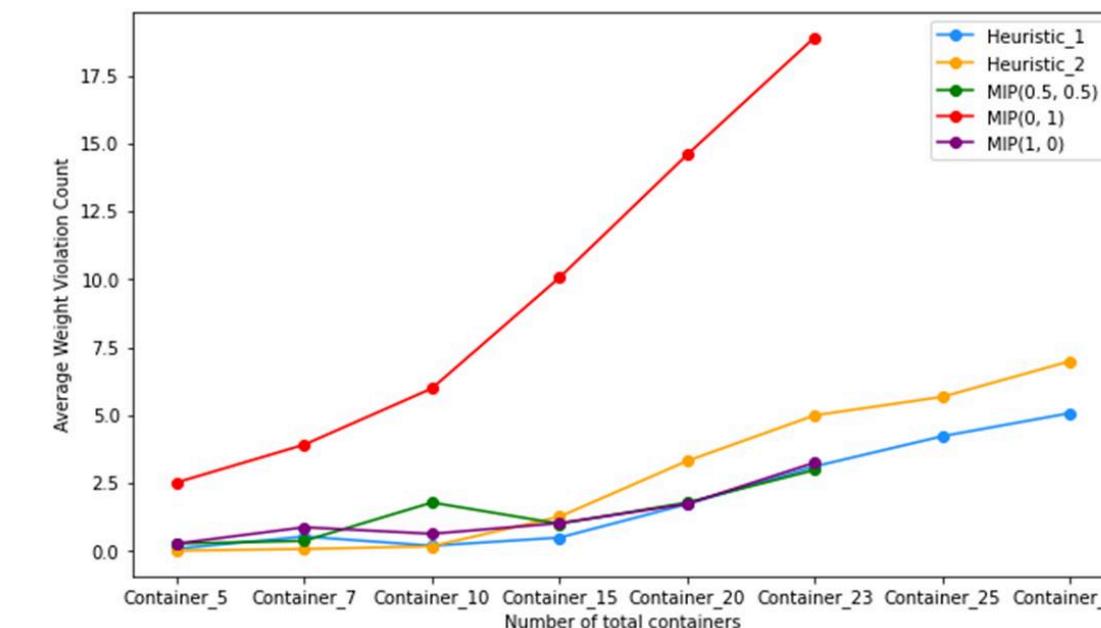
Research Process

이혜진

1. 피크스택 유무에 따른 안전성 평가



2. 각 방법론 별 생산성 및 안전성 비교

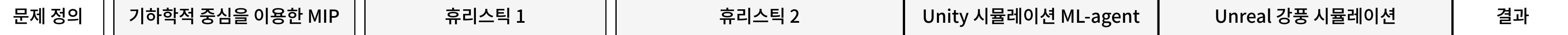


각 최적화 기법별 생산성 및 안전성 비교

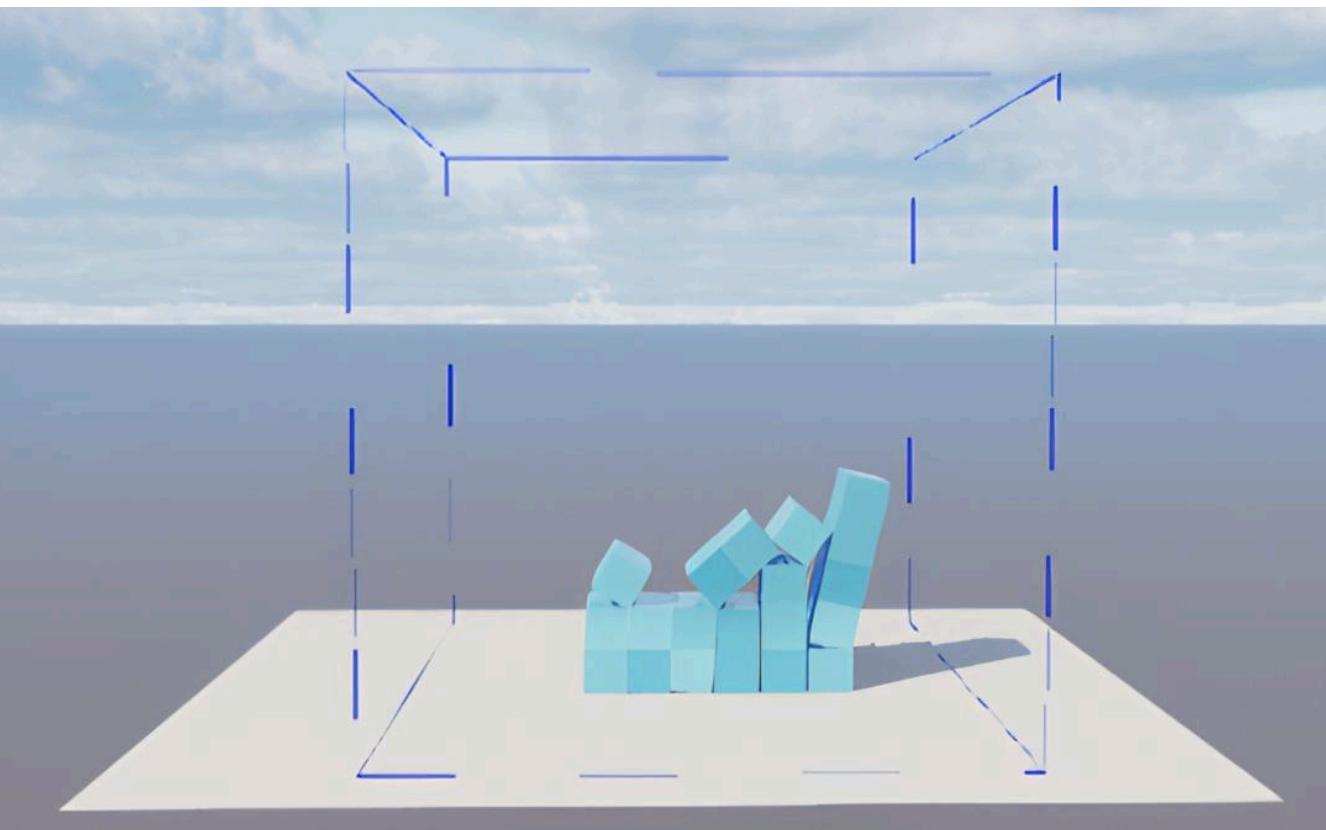
평균 피크스택 갯수가 더 많은 허리스틱 1이 안전성 현저히 낮은 걸 확인

START

DONE



강풍 시뮬레이션 구현



Unreal simulation

- 목표 : 본 연구에서 개발한 MIP 및 허리스틱 방법을 통해 만들어진 최종 배치 형상이 강풍에 안전한 적재 계획인지 검증하기 위함.
- 평가 기준 : 컨테이너가 바람에 의해 넘어지는 정도를 평가하여 안전성 측정 및 총 relocation 횟수 비교

<생산성>

- 총 relocation 횟수
- 무게 위반(무거운 컨테이너 위에 가벼운 컨테이너 스택킹) 횟수

<안전성>

- 강풍 시뮬레이션 적용 후, position 변동 위치
- 강풍 시뮬레이션 적용 후, rotation 변동 위치