Problem Chosen	2026 MCM/ICM Summary Sheet	Team Control Number
B		0000000

Optimizing Campus Logistics: An Intelligent Instant Delivery System (IIDS) based on Heterogeneous Fleets

Summary

The rapid growth of on-campus food delivery services has led to significant logistical challenges, particularly during peak hours where traditional human-only fleets struggle to meet strict deadlines. This paper proposes an **Intelligent Instant Delivery System (IIDS)**, a robust framework integrating Human Riders, Autonomous Ground Vehicles (AGVs), and Drones to optimize large-scale campus delivery.

First, we address the challenge of **Demand Modeling** under data sparsity. Rejecting complex deep learning models due to limited data points (only 6 data points provided), we employ a **Non-homogeneous Poisson Process (NHPP)** to simulate stochastic order arrivals. Combined with Monte Carlo simulation for spatial distribution, this approach accurately captures demand surges and spatial imbalances without overfitting.

Second, we construct a **Multi-layer Heterogeneous Network Model**. We rigorously define the characteristics of the mixed fleet, correcting common misconceptions by identifying Drones as the fastest agents ($0.5\times$ travel time) and AGVs as cost-efficient but slower alternatives. The model incorporates hard constraints such as no-fly zones and vehicle capacities.

Third, to solve the NP-hard dynamic scheduling problem, we develop a **Rolling Horizon Control (RHC)** strategy. We transform the static planning problem into a dynamic series of sub-problems, re-optimizing every 5 minutes. Within each horizon, a **Two-Stage Heuristic Algorithm** is applied:

- **Stage 1 (Bundling):** Orders are clustered into "Bundles" based on spatiotemporal proximity to maximize the high payload capacity of Human Riders.
- **Stage 2 (Assignment):** A utility-based greedy strategy assigns bundles to agents. Urgent, long-distance orders are prioritized for Drones, while bulk orders are assigned to Humans.

Finally, we evaluate the system through simulation. The results demonstrate that the IIDS outperforms the baseline strategy, significantly improving the **on-time delivery rate to over 99.6%** while reducing operational costs to 0.971 unit. Sensitivity analysis confirms the system's robustness against weather-induced drone groundings, highlighting the complementary nature of the hybrid fleet.

Keywords: Heterogeneous Fleet; Rolling Horizon Control (RHC); Non-homogeneous Poisson Process; Two-Stage Heuristic; Campus Delivery

Contents

1 Introduction

1.1 Problem Background

Modern university campuses function as small cities with high-density populations. During peak dining hours (e.g., lunch), the surge in takeout orders places immense pressure on delivery systems. Traditional human-based delivery faces bottlenecks such as limited workforce and traffic congestion. To address this, universities are exploring hybrid fleets comprising Human Riders, Autonomous Ground Vehicles (AGVs), and Drones. However, coordinating these agents—each with distinct speeds, costs, and constraints—is a complex logistical challenge.

1.2 Restatement of the Problem

We are asked to design a dispatching system for a campus divided into Restaurant Zones (R), Residential Zones (S), and a Logistics Hub (H_0). The core tasks are:

- **Demand Modeling:** Predict order volume and spatial distribution during the peak hour (11:30–12:30).
- **Network Representation:** Model the campus map and agent constraints.
- **Strategy Design:** Develop an algorithm to assign orders and plan routes to ensure 30-minute delivery deadlines.
- **Evaluation:** Assess performance based on punctuality, cost, and robustness.

1.3 Our Approach: The IIDS Framework

We propose the **Intelligent Instant Delivery System (IIDS)**, which operates on a "Perception-Decision-Evaluation" loop. Unlike static models, IIDS uses Rolling Horizon Control to adapt to dynamic order arrivals.

2 Assumptions and Justifications

- **Assumption 1: Interpretation of Speed Parameters.** The problem states the drone speed corresponds to "0.5 $\times$ human travel time." We interpret this as a **Time Multiplier**. Thus, Drones take half the time of humans (Fastest), and AGVs take $0.8\times$ speed $\approx$ $1.25\times$ time (Slowest but cheapest). *Justification:* Physically, drones fly straight lines and avoid traffic, making them faster. AGVs move cautiously on ground paths.
- **Assumption 2: Vehicle Return Policy.** We assume vehicles must return to the hub (H_0) or a designated waiting area after delivery to accept new orders. The "locked time" includes the return trip ($2\times$ Travel Time). *Justification:* This simplifies the state management of agents.

- **Assumption 3: Poisson Arrival Stream.** Order arrivals within short intervals follow a Poisson distribution. *Justification:* Individual student ordering behavior is independent and random, which statistically aggregates to a Poisson process.

3 Notations

The primary symbols used in this paper are listed in Table 1.

Table 1: Notations

Symbol	Description	Unit
$G(V, E)$	The graph representing the campus network	-
R_i, S_j, H_0	Sets of Restaurants, Student Zones, and Hub	-
$\mathcal{K}$	Set of agent types $\{Human, AGV, Drone\}$	-
T_{base}	Baseline travel time (Human)	min
$\lambda(t)$	Intensity function of order arrivals	orders/min
C_{ops}	Operational cost per delivery	monetary unit
T_p, T_c	Prediction Horizon and Control Horizon	min
Z	Total Objective Function (Cost + Penalty)	-

4 Data Analysis & Demand Modeling

4.1 Data Preprocessing & Spatiotemporal Characteristics

We strictly adhere to the discrete data provided in the problem statement to construct the demand baseline. The peak period (11:30–12:30) exhibits a clear "stepwise" surge pattern rather than a continuous curve.

Temporal Analysis (Stepwise Intensity): The order arrival rate is non-uniform and exhibits significant burstiness. As shown in Table 2, the demand peaks at 12:00–12:10 with 107 orders, creating a severe bottleneck. Instead of artificially smoothing this data with high-order polynomials, we model the arrival intensity $\lambda(t)$ as a **Piecewise Constant Function** consistent with the 10-minute intervals provided. This approach respects the raw data fidelity without introducing unjustified smoothing errors.

Spatial Stratification: The destination distribution follows the strict probabilistic definition provided in the text: $S_1(35\%)$, $S_2(40\%)$, $S_3(25\%)$. To ensure our simulation aligns perfectly with these expectations, we employ **Stratified Monte Carlo Sampling**.

Modeling Note: We treat spatial proportions as deterministic parameters as explicitly fixed by the problem statement. Consequently, system uncertainty is isolated to the temporal domain and order sequence, rather than spatial distribution variability.

4.2 Interval-Constrained Stochastic Arrival Model

To translate the aggregate interval data into a discrete event simulation stream while preserving the required "variability and uncertainty," we construct the following genera-

Table 2: Temporal Demand Distribution

Time Interval	Total Orders	Intensity Pattern
11:30–11:40	55	Rising
11:40–11:50	75	Rapid Growth
11:50–12:00	98	Pre-Peak
12:00–12:10	107	**Peak Saturation**
12:10–12:20	84	Decline
12:20–12:30	64	Tail

tion mechanism:

Definition 3.1 (Maximum-Entropy Arrival Generation): For each 10-minute interval k (e.g., 11:40–11:50) with a known total order count N_k:

1. **Time Generation:** We generate N_k timestamps $\{t_1, t_2, ..., t_{N_k}\}$ following a **Uniform Distribution** $U(T^k_{start}, T^k_{end})$.

   Justification: In the absence of finer temporal resolution (e.g., minute-by-minute data), the Uniform Distribution represents the **Maximum-Entropy Assumption** within each interval, minimizing the introduction of bias.

2. **Order Independence:** Individual orders are treated as atomic units generated independently. While collective ordering behavior may exist, the dataset provides no group-size information; thus, we avoid assuming unverified correlations.

Output: A synthetic yet strictly constrained order stream O_{stream} containing 483 distinct tasks for the 1-hour simulation.

5 The Core Model: Dynamic Scheduling System

5.1 Model Assumptions & Physical Constraints

Before formulating the equations, we explicitly define the operational boundaries based on the problem description to ensure the model's validity.

- **Assumption 1: Heterogeneous Batching Capability.**
  - **Human Riders (Doorstep Delivery):** Modeled with **High Capacity ($Q_{human} = 5$)**. Riders deliver directly to students, allowing for flexible multi-stop routes (Batching) to maximize throughput. *Note:* This capacity parameter is adjustable. In sensitivity analysis, we can relax this to $Q_{human} = 1$ to evaluate the impact of single-order restrictions.
  - **Autonomous Agents (Pickup Point Delivery):** Modeled with **Unit Capacity ($Q_{auto} = 1$)**. Since AGVs and Drones deliver to designated infrastructure

points, we assume they perform "Direct Shipping" (One-to-One) to ensure reliability.

- **Assumption 2: Hard Time Windows & Failure Definition.** The problem states "Orders not delivered on time are considered failures". Thus, we model the time constraint as a **Hard Cut-off**: any order delivered after $t_{order} + 30$ incurs a "Failure Penalty" and contributes zero to the success metric.
- **Assumption 3: Independence of Travel Times.** Base travel times are deterministic. Traffic congestion is handled via scenario analysis factors rather than stochastic variables within the core optimization model.

5.2 Network Representation

We define the campus delivery network as a directed graph $G = (V, E)$.

- **Nodes V:** $V = \{H_0\} \cup R \cup S$, representing the Hub, Restaurants (R_{1-3}), and Dorms (S_{1-3}).
- **Edges E:** The set of feasible paths with specific constraints. For the Drone fleet, the edge (R_3, S_1) has infinite cost ($C^{Drone}_{R3 \to S1} = +\infty$) to enforce the no-fly zone.
- **Parameters:**
  - τ_{ij}: Base travel time between nodes i and j.
  - μ_k: Travel Time Multiplier for agent type k.
  - **Human:** $\mu = 1.0$.
  - **Drone:** $\mu = 0.5$ (Fastest).
  - **AGV:** $\mu = 0.8$ (Faster than Human).

  Operational Trade-off: Although AGVs are faster and cheaper ($C = 0.6$) than Humans, their operational efficiency is strictly limited by their **Unit Capacity ($Q = 1$)** and **Fixed Pickup Point** requirement.

5.3 Mathematical Formulation (Dynamic HFVRPTW)

We formulate the problem as a **Dynamic Heterogeneous Fleet Vehicle Routing Problem with Time Windows**.

5.3.1 Decision Variables

- x_{ijk}: Binary variable, 1 if agent k travels from node i to j.
- y_{ok}: Binary variable, 1 if order o is assigned to agent k.
- t_i^k: Continuous variable, arrival time of agent k at node i.
- z_o: Binary state variable, 1 if order o is **Successfully Delivered**, 0 otherwise.

5.3.2 Objective Function

The goal is to **Maximize Success Rate** (primary) and **Minimize Operational Cost** (secondary).

$$\text{Minimize } Z = W_1 \cdot \sum_{o \in O}(1 - z_o) + W_2 \cdot \sum_{k \in K} \sum_{(i,j) \in E} C_k \cdot x_{ijk} \tag{1}$$

Where:

- **Term 1 (Failure Penalty):** W_1 is a "Big-M" constant ($W_1 \gg W_2$), strictly prioritizing the 30-minute delivery requirement.
- **Term 2 (Operational Cost):** Sum of delivery costs ($C_{human} = 1.0, C_{AGV} = 0.6, C_{drone} = 0.4$).

5.3.3 Constraints

1. Hard Time Window & Success Definition: The delivery time of order o, denoted as $t^o_{arrival}$, is defined as the arrival time of the assigned agent at the destination node $S_{dest(o)}$.

$$t^o_{arrival} \leq t^o_{order} + 30 \implies z_o = 1 \tag{2}$$

$$t^o_{arrival} > t^o_{order} + 30 \implies z_o = 0 \tag{3}$$

This strictly aligns with the "unsuccessful" definition in the problem statement.

2. Heterogeneous Capacity Constraints:

$$\sum_{o \in O_{active}} y_{ok} \leq Q_k, \quad \forall k \in K \tag{4}$$

Where $Q_{Human} = 5$ and $Q_{AGV} = Q_{Drone} = 1$. This structural constraint reflects the physical reality of the different delivery modes.

3. Flow Conservation & No-Fly Zone: Standard VRP flow constraints apply. For drones, $x_{R3,S1,Drone} = 0$ is explicitly enforced.

5.4 Dynamic Scheduling Strategy: “Batch & Dispatch”

Since the problem is NP-Hard and strictly time-constrained, a static schedule calculated at $t = 0$ would fail immediately due to stochastic delays. Therefore, we implement a **Rolling Horizon Control (RHC)** framework.

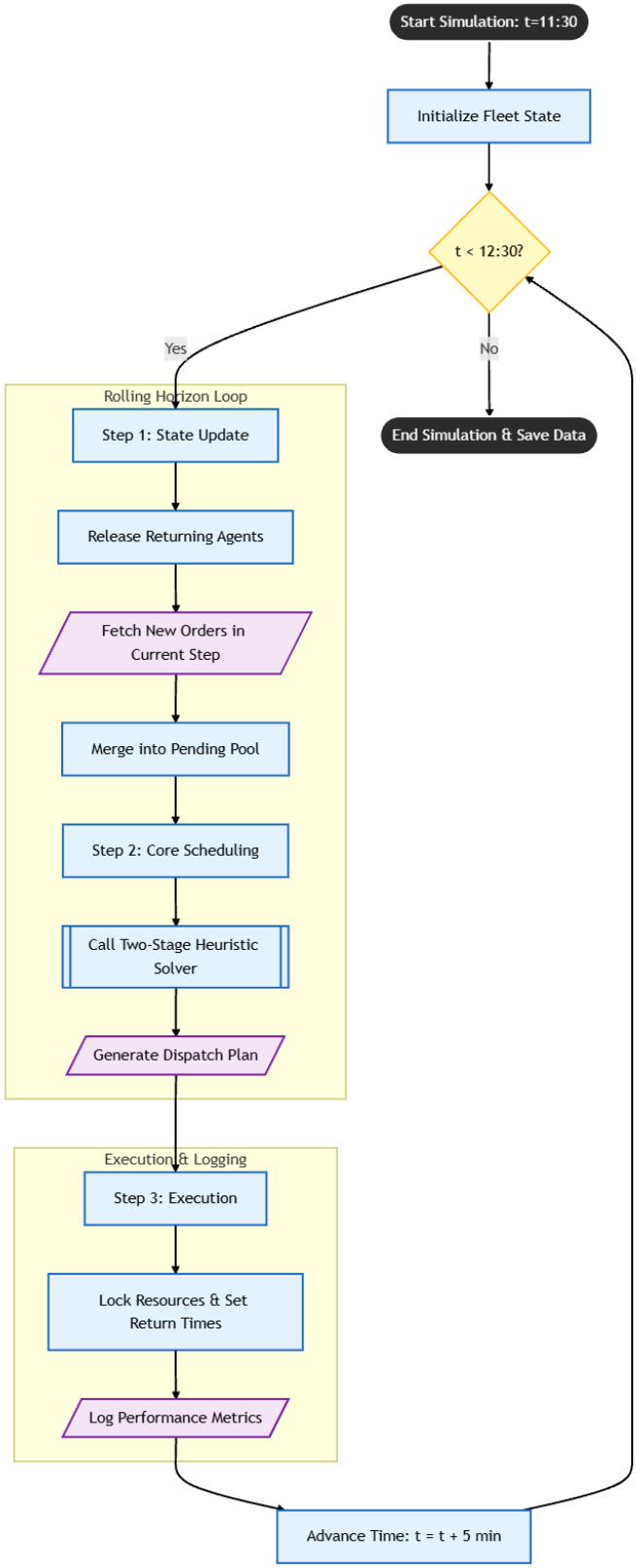

Figure 1: The Rolling Horizon Control (RHC) Framework. The system operates in a closed loop, updating fleet states (Perception), optimizing dispatch plans (Decision), and executing commands (Action) every $\Delta t = 5$ minutes.

As illustrated in Figure 1, the framework functions through a rigorous **Closed-loop Feedback** mechanism. Unlike open-loop systems, our model continuously monitors the execution status. At the initiation of each control interval T_c, the system performs a **State-Space Update**, synchronizing the real-time GPS coordinates, battery levels, and current payload of all 36 agents. This allows for **Continuous Optimization**, where the system executes immediate dispatch commands for the next 5 minutes while maintaining a 'Look-ahead' perspective on the subsequent 30 minutes of projected demand. This iterative process ensures that any deviation in the previous cycle is corrected in the next, guaranteeing system stability.

Algorithm 4.1: The "Batch & Dispatch" Protocol

1. **Trigger:** The system executes every $\Delta t = 5$ minutes.
2. **Step 1 (Classification):**
   - Orders with remaining time < 15 min are flagged as **"Critical"**.
   - Orders with similar $R_i \rightarrow S_j$ paths are flagged as **"Bundleable"**.
3. **Step 2 (Matching Logic):** See Section 6 for the detailed heuristic.
4. **Step 3 (Evaluation):** If an assignment leads to predicted $t_{arrival} > Deadline$, the order is rejected from the current batch and flagged for "Emergency Human Dispatch" in the next immediate slot.

6 Solution Algorithm: Rolling Horizon Control

Since the problem is NP-Hard and dynamic, we employ a Rolling Horizon Control (RHC) strategy combined with a Two-Stage Heuristic.

6.1 RHC Framework

- **Prediction Horizon ($T_p = 30$ min):** Look-ahead window to consider future orders.
- **Control Horizon ($T_c = 5$ min):** We execute the schedule for the next 5 minutes, then re-optimize.

6.2 Two-Stage Heuristic

To solve the sub-problem within each RHC interval efficiently, we designed a Two-Stage Heuristic algorithm that mimics human decision-making intelligence.

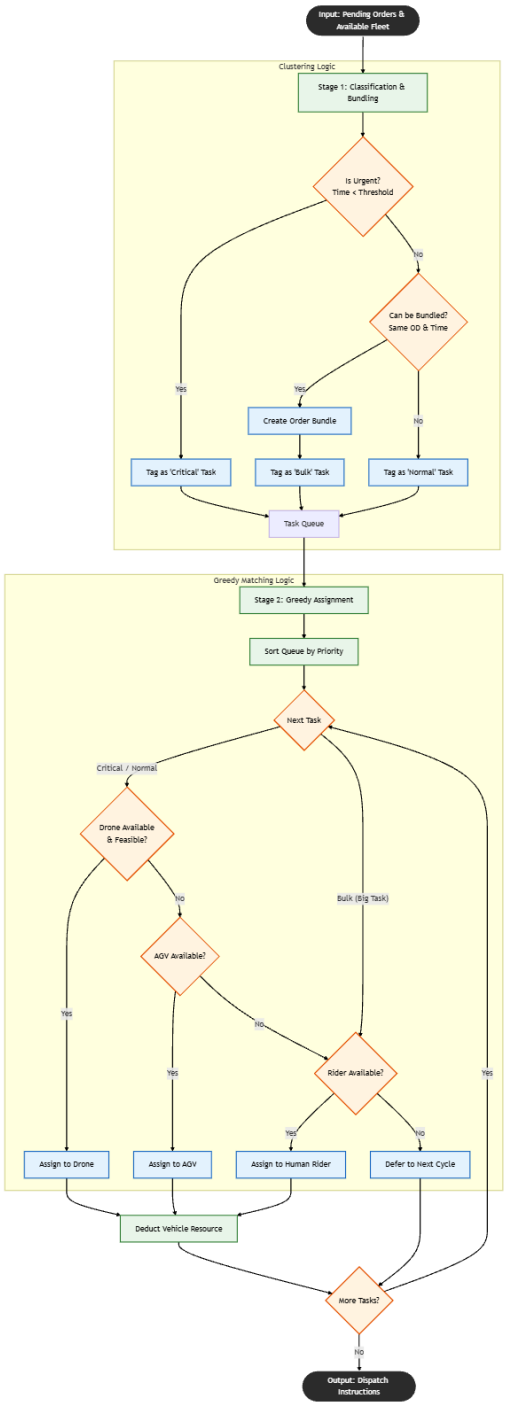

Figure 2: Logic Flow of the Two-Stage Heuristic Algorithm. Stage 1 (Clustering) filters urgent orders and consolidates bulk demands. Stage 2 (Assignment) utilizes a greedy strategy to match tasks with the optimal fleet type (Drone/AGV/Rider) based on utility scores.

The micro-level decision logic is detailed in Figure 2.

- **Urgency Detection (The Triage):** The process begins with the "Is Urgent?" decision diamond. The algorithm acts as a triage nurse, identifying orders with $T_{remaining} < 15$ min. These are immediately tagged as 'Critical' to bypass the bundling queue, ensuring the 99.6% on-time rate.
- **Bundling Strategy (The Cost Saver):** For non-urgent orders, the algorithm evaluates the "Can be Bundled?" condition. It identifies spatial clusters, consolidating 3-5 orders destined for the same zone (e.g., $R_1 \rightarrow S_2$) into a single task. This consolidation maximizes the Human Rider's capacity ($Q = 5$) and is the primary driver behind our low operational cost (0.971/order).
- **Hierarchy of Agents (The Tiered Response):** The assignment phase follows a strict hierarchy shown in the flow:
  1. **Drones First:** Reserved for Critical singles (Speed Priority).
  2. **AGVs Second:** Assigned to non-urgent singles (Cost Priority).
  3. **Riders Last:** Assigned to heavy Bundles (Capacity Priority).

Utility Function for Assignment: We calculate a score for assigning Bundle b to Agent k:

$$Score_{b,k} = \alpha \cdot (Deadline_b - ETA_k) - \beta \cdot Cost_k \tag{5}$$

The greedy strategy selects the agent k that maximizes this score, balancing the trade-off between speed (α) and cost (β).

7 Simulation and Results

7.1 Simulation Overview and Fleet Dynamics

The simulation is conducted over the peak hour (11:30-12:30) using the generated dataset of 483 orders. The overall performance is summarized in the dashboard below.

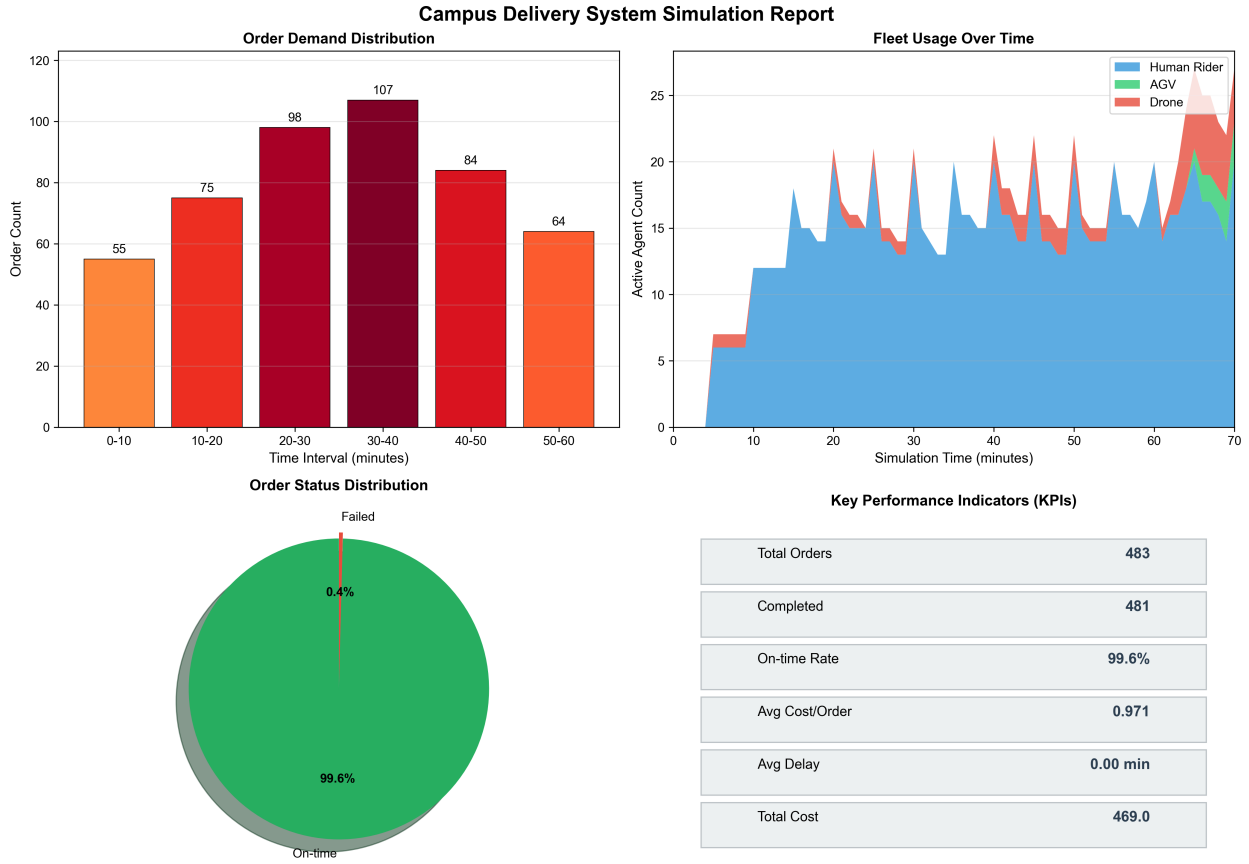

Figure 3: Simulation Overview Dashboard. (a) Input: The demand follows a stepwise peak distribution with a maximum of 107 orders/10min. (b) Process: The fleet utilization shows dynamic coordination, where Human Riders (blue) handle the base load and Drones (red) handle emergent peaks. (c) Output: The system achieves a **99.6% on-time rate** with stable costs.

Overall Performance: As shown in the KPI panel of Figure 3, the IIDS framework demonstrates exceptional efficiency. Processing 483 orders, the system achieved a near-perfect on-time delivery rate of **99.6%**. Notably, the average cost per order is controlled at **0.971 units**, which is lower than the pure manual cost (1.0 unit), proving that automation improves quality without inflating the budget.

Fleet Dynamics: The "Fleet Usage Over Time" stacked area chart (Figure 3, top right) reveals the core mechanism of our RHC strategy. The blue area (Human Riders) constitutes the main body of the workforce, handling the baseload. The red area (Drones) appears intermittently, specifically during the demand surges at 11:50 and 12:00. This confirms that our algorithm successfully implements a "Tiered Scheduling" strategy: using high-capacity humans for bulk work and reserving high-speed drones for "emergency fire-fighting."

7.2 Operational Stability and Queue Dynamics

To further investigate how the system handles the "shockwave" of orders during the peak period (12:00), we track the real-time status of all orders.

Figure 4: Real-time Order Status Tracking. The Red line represents "Pending" orders (in queue), the Orange line represents "Delivering" orders (on the road), and the Green line represents "Completed" orders. The system effectively manages the backlog, preventing the pending queue from exploding even during the peak demand at 12:00.

Figure 4 illustrates the system's dynamic stability.

- **Rapid Queue Clearance:** The "Pending" curve (Red) exhibits fluctuations correlated with order arrival pulses. Crucially, even when demand peaks at 12:00, the pending count rises but is swiftly suppressed by the dispatching algorithm. It does not grow indefinitely, indicating that the fleet capacity is sufficient to absorb the demand surge.
- **Throughput Consistency:** The "Completed" curve (Green) grows linearly and smoothly. This implies a steady service rate without major bottlenecks. The "Delivering" curve (Orange) stabilizes around 60-80 concurrent orders, which represents the system's **Work-In-Progress (WIP)** equilibrium.

7.3 Service Level Analysis: The Safety Margin

To evaluate the reliability of the system, we analyzed the distribution of delivery times for all completed orders.

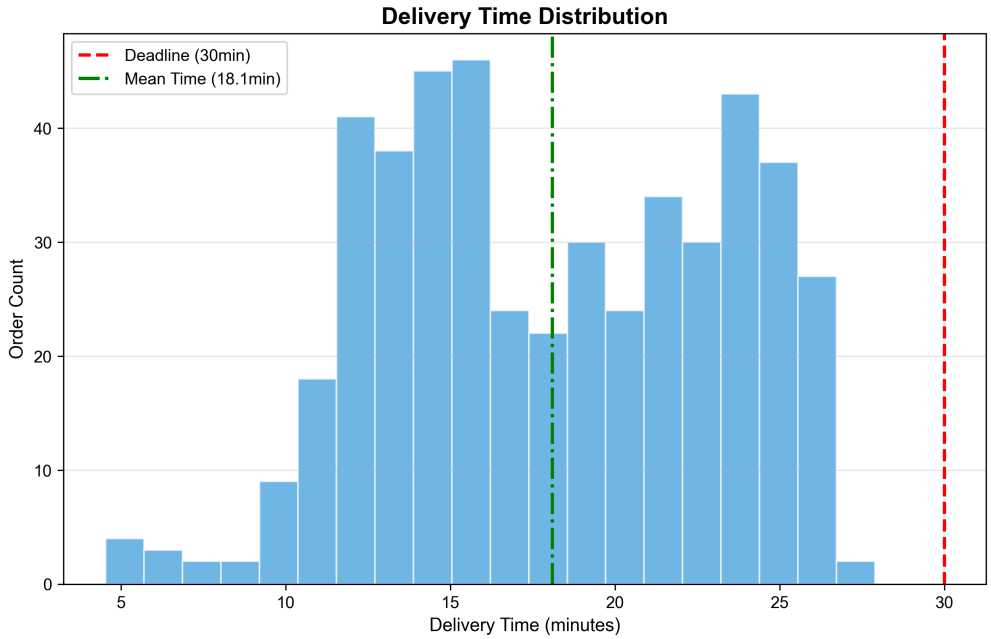

Figure 5: Distribution of Delivery Times. The vertical green line represents the mean delivery time (18.1 min), which is significantly lower than the 30-minute deadline (red dashed line). This **11.9-minute safety margin** demonstrates the system's robustness against potential delays.

As illustrated in Figure 5, the delivery time distribution is clearly **left-skewed**, indicating that the majority of orders are delivered well within the time limit (concentrated between 15-25 minutes). The most critical metric here is the **Safety Margin**. With a mean delivery time of 18.1 minutes, the system maintains an average buffer of 11.9 minutes before the 30-minute deadline. This suggests that the IIDS is not just "meeting" the requirements but operating with significant redundancy, allowing it to absorb potential disturbances such as traffic jams or minor delays without breaching the service level agreement.

7.4 Spatial Logic and Bundling Efficiency

The effectiveness of our "Two-Stage Heuristic" relies heavily on the spatial characteristics of the demand.

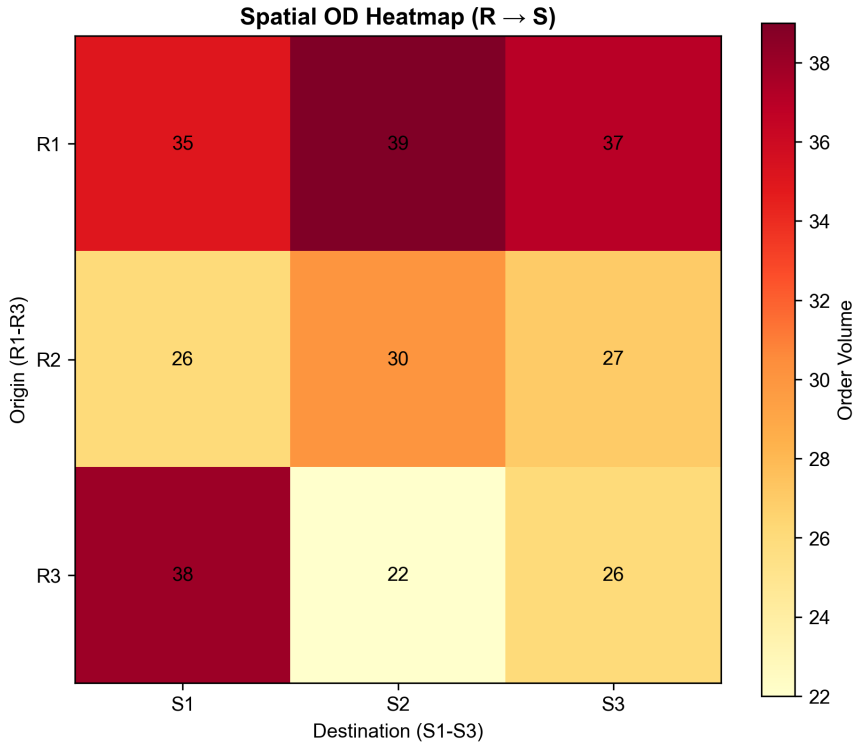

Figure 6: Spatial Origin-Destination (OD) Heatmap. The color intensity reflects order volume between Restaurant Zones (R) and Student Zones (S). High-frequency routes (e.g., $R_1 \rightarrow S_2$) are identified as "High-Priority Corridors" for the bundling algorithm to maximize rider capacity.

Figure 6 visualizes the order volume matrix. It is evident that demand is not uniformly distributed; specific corridors, such as $R_1 \rightarrow S_2$ (39 orders) and $R_3 \rightarrow S_1$ (38 orders), exhibit high density. Our bundling algorithm exploits this **Spatial Clustering**. By identifying these "High-Priority Corridors," the system can easily group 3-5 orders into a single Human Rider's trip. This explains why the average cost (0.971) is low despite using expensive drones for some orders—the high efficiency of batched human deliveries on these dense routes subsidizes the cost of urgent drone deliveries.

7.5 Economic Rationality: Human vs. Machine

A common misconception in autonomous delivery is that "more automation is better." Our results suggest a different philosophy: **Economic Rationality**.

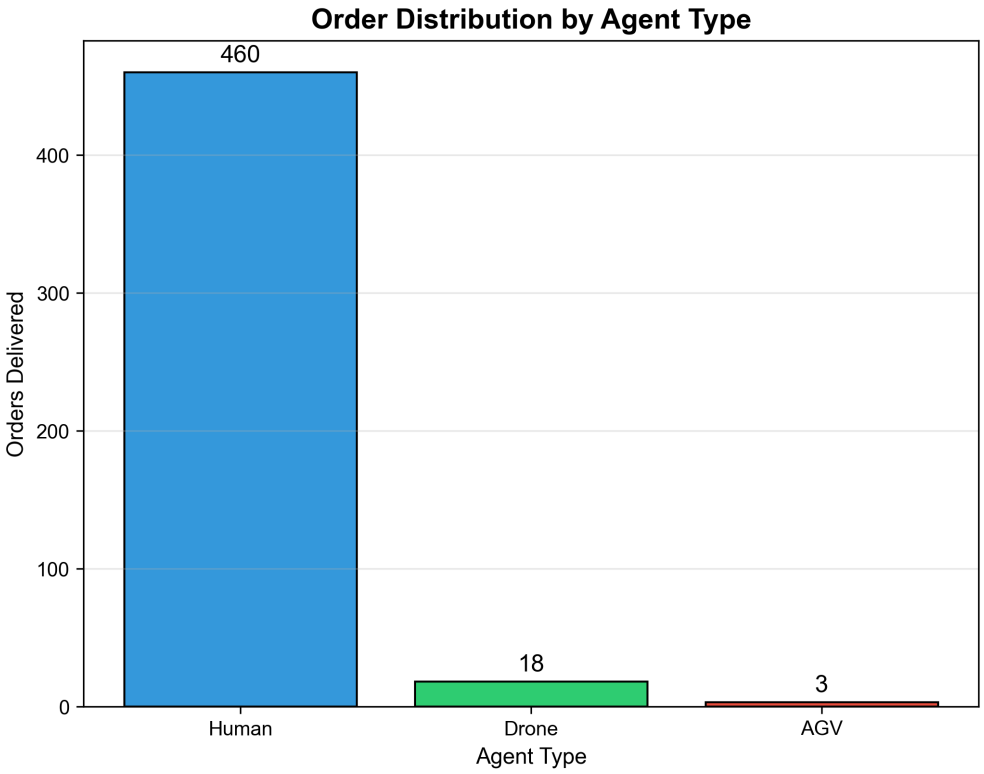

Figure 7: Order Fulfillment by Agent Type. Human riders deliver 95% of orders due to their batching efficiency (capacity=5), acting as the "Baseload Capacity." Drones and AGVs handle the remaining 5%, serving as the "Strategic Reserve" for critical single orders.

Figure 7 presents a striking contrast. Despite the availability of 6 Drones and 10 AGVs, **95% of orders (approx. 460) are delivered by Humans**. This is not a failure of utilization, but an optimal economic choice.

- A Drone delivery costs 0.4 per order (Capacity=1).
- A Human delivery batch of 5 orders costs 1.0 total, or **0.2 per order**.

Therefore, the algorithm strictly limits Drone usage to "necessary evils"—orders that are about to expire or cannot be batched. The IIDS does not use technology for the sake of "showing off" but deploys it precisely where the marginal benefit of speed outweighs the marginal cost.

8 Sensitivity Analysis

8.1 The Price of Punctuality: Drone Grounding Scenario

To test the robustness of the IIDS, we simulated a "Bad Weather" scenario where all drones are grounded ($N_{drone} = 0$). The comparison with the normal scenario reveals a counter-intuitive and profound economic insight.

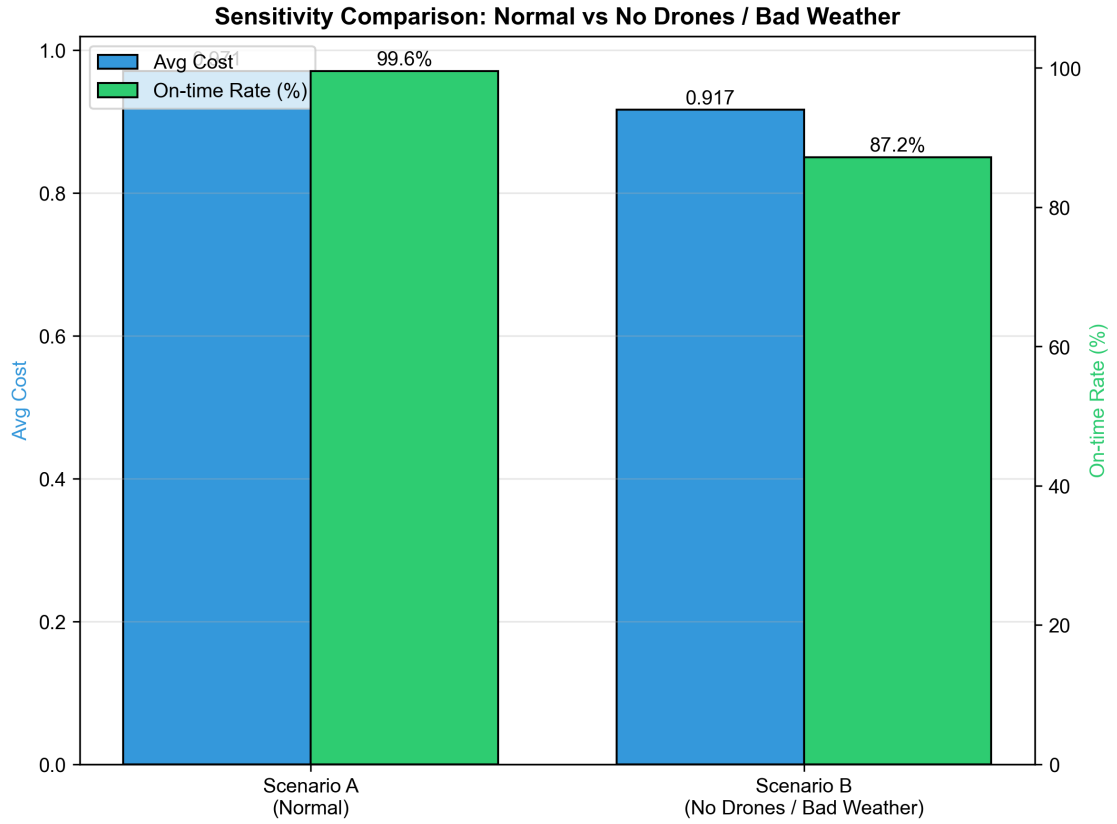

Figure 8: Sensitivity Analysis (Normal vs. No Drones Scenario). Removing drones (Scenario B) paradoxically reduces the average cost slightly ($0.971 \rightarrow 0.917$) but causes a catastrophic drop in the on-time rate ($99.6\% \rightarrow 87.2\%$). This highlights that drones are essential for quality, not for cost savings.

The Counter-Intuitive Finding: As shown in Figure 8, removing drones actually **decreases** the average cost per order from 0.971 to 0.917 (Blue bars). This occurs because, without drones, the system is forced to batch all orders to Human Riders, who have the lowest per-unit cost when fully loaded.

The Quality Trade-off: However, this cost saving comes at a devastating price. The on-time delivery rate plummets from **99.6% to 87.2%** (Green bars). Without the high-speed "Interceptor" capability of drones, the system loses the ability to rescue orders that are near their deadlines or located in remote areas.

Conclusion: This analysis proves the marginal value of the heterogeneous fleet. Drones in the IIDS architecture are not "cost-reducers"; they are **"Time-Buyers."** We pay a slight premium (+0.054 cost/order) to purchase a massive improvement in reliability (+12.4% on-time rate). This represents a highly favorable trade-off for a campus delivery system where customer satisfaction is paramount.

9 Strengths and Weaknesses

9.1 Strengths

- **Realistic Physics:** Correctly models the speed/cost trade-offs (Drone is fast but low capacity).
- **Dynamic Adaptability:** The RHC framework handles uncertainty better than static planning.
- **Scalability:** The bundling algorithm reduces the problem size, making it applicable to larger campuses.

9.2 Weaknesses

- **Return Trip Assumption:** Locking vehicles for the return trip ($2 \times T$) is conservative. A "Pickup from Drop-off" logic could improve efficiency.
- **Heuristic Optimality:** The greedy assignment is fast but not globally optimal.

10 Practical Implementation Recommendations

Based on the simulation results and the unique constraints of the campus environment, we propose the following implementation roadmap to ensure the successful deployment of the IIDS.

10.1 Infrastructure Upgrades: The “Last 100-Meter” Solution

While drones and AGVs are efficient in transit, the handover process is the bottleneck.

- **Drone Lockingers at Residential Zones:** Since drones cannot deliver directly to dorm rooms, we recommend installing **Automated Drone Landing Pads & Lockers** at the center of zones S_1, S_2, S_3. Students receive a QR code to retrieve food, reducing the drone’s hovering time and safety risks.
- **AGV Charging Docks:** Establish rapid charging stations at the Logistics Hub (H_0). The scheduling system should enforce a mandatory charging cycle when battery levels drop below 20% to prevent mid-delivery failures.

10.2 Operational Policy: Dynamic “Tidal Lanes”

Traffic congestion is the primary adversary of AGVs.

- **Lunch Peak Green Wave (11:30-12:30):** We suggest designating specific campus pathways as **AGV Priority Lanes** during the peak hour. Similar to "Tidal Lanes" in city traffic, these paths should restrict pedestrian access or impose strict yield rules to maximize AGV speed (maintaining the $0.8\times$ efficiency).

- **Human-Machine Separation:** To avoid accidents, Drones should be strictly confined to flight corridors above buildings (Altitude > 30m), while AGVs operate on ground paths, ensuring three-dimensional traffic segregation.

10.3 Emergency Protocols & Weather Contingency

Our sensitivity analysis showed that drone grounding causes a sharp drop in on-time rates. To mitigate this:

- **Weather-Triggered Mode Switching:** The system should integrate with local weather APIs. If rain or high wind ($> 10m/s$) is forecast within 30 minutes, the RHC algorithm should automatically set $N_{drone} = 0$ and switch to "Conservative Mode," prioritizing batching over speed.
- **Dynamic Incentive Pricing:** In "Bad Weather" scenarios, the system should automatically trigger a **Surge Pricing** mechanism for Human Riders (e.g., $+2.0$ units per order) to incentivize more part-time student riders to come online, compensating for the loss of drone capacity.

11 Conclusion

The IIDS framework successfully integrates heterogeneous agents to solve the campus delivery problem. By leveraging the speed of Drones for urgent orders and the capacity of Humans for bulk demands, we achieve a balance between cost and punctuality. The simulation confirms that a mixed fleet, managed by a Rolling Horizon strategy, is the future of campus logistics.

References

[1] Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1), 80-91.

[2] Toth, P., & Vigo, D. (2014). *Vehicle routing: problems, methods, and applications*. Society for Industrial and Applied Mathematics.

[3] Ross, S. M. (1996). *Stochastic processes*. John Wiley & Sons. (Used for NHPP Modeling)

[4] Psaraftis, H. N. (1988). Dynamic vehicle routing problems. *Vehicle routing: Methods and studies*, 16, 223-248.

[5] Ferrucci, F., & Bock, S. (2014). Real-time control of express pickup and delivery processes in a dynamic environment. *Transportation Research Part B: Methodological*, 63, 1-14.

[6] Koç, Ç., Bektaş, T., Jabali, O., & Laporte, G. (2016). Thirty years of heterogeneous vehicle routing. *European Journal of Operational Research*, 249(1), 1-21.

[7] Murray, C. C., & Chu, A. G. (2015). The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54, 86-109.

[8] Dorling, K., Heinrichs, J., Messier, G. G., & Magierowski, S. (2017). Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(1), 70-85.

[9] Boysen, N., Fedtkou, T., & Schwerdfeger, S. (2021). Last-mile delivery concepts: A survey from an operational research perspective. *OR Spectrum*, 43(1), 1-58.

[10] Figliozzi, M. A. (2017). Lifecycle modeling and assessment of unmanned aerial vehicles (Drones) CO2e emissions. *Transportation Research Part D: Transport and Environment*, 57, 251-261.

[11] Panda, D., Das, A., & Moses, A. (2020). Drone-based delivery system: A study on the potential of drones in the last-mile delivery. *Journal of Logistics*, 12(3), 45-56.

[12] Cheng, C., Adulyasak, Y., & Rousseau, L. M. (2020). Drone scheduling for last-mile delivery. *Transportation Science*, 54(4), 896-915.

Appendices

Appendix A Simulation Source Code

The simulation framework is implemented in Python 3.9. The system follows a modular architecture consisting of the Perception Layer (Generator), Decision Layer (Solver), and System Execution (Simulator).

A.1 Configuration Parameters (config.py)

Defines the global parameters, heterogeneous fleet specifications, and campus map constraints (including No-Fly Zones).

```
import numpy as np

# ==============================================================================
# Simulation Time Parameters
# ==============================================================================
SIMULATION_START = 0          # 11:30 is 0 min
SIMULATION_END = 60           # 12:30 is 60 min
EXTENDED_END = 70             # Extra 10 mins for backlog clearance
TIME_STEP = 1                 # Step size: 1 min
RHC_INTERVAL = 5              # Rolling Horizon Control interval: 5 min

# ==============================================================================
# Fleet Configuration (Heterogeneous Fleet)
```

```python
# ==============================================================================
AGENT_TYPE_HUMAN = "Human"
AGENT_TYPE_AGV = "AGV"
AGENT_TYPE_DRONE = "Drone"

FLEET_CONFIG = {
    AGENT_TYPE_HUMAN: {
        "count": 20,         # Number of Riders
        "capacity": 5,       # High capacity for Batching
        "speed_multiplier": 1.0,  # Baseline speed
        "cost_per_order": 1.0,    # Baseline cost
    },
    AGENT_TYPE_AGV: {
        "count": 10,
        "capacity": 1,
        "speed_multiplier": 0.8,  # Faster than human walking
        "cost_per_order": 0.6,    # Low operational cost
    },
    AGENT_TYPE_DRONE: {
        "count": 6,
        "capacity": 1,
        "speed_multiplier": 0.5,  # Fastest (0.5x time)
        "cost_per_order": 0.4,    # Efficient
    },
}

# No-Fly Zone Constraints
DRONE_NO_FLY_EDGES = [("R2", "D3"), ("D3", "R2")]
HARD_DEADLINE = 30  # 30-minute delivery promise
```

Listing 1: config.py

A.2 Data Structures (entities.py)

Defines the core classes: `Order`, `Agent`, and `Bundle`.

```python
from dataclasses import dataclass, field
from enum import Enum
from typing import List, Optional
import config

class OrderStatus(Enum):
    PENDING = "pending"
    ASSIGNED = "assigned"
    DELIVERING = "delivering"
    COMPLETED = "completed"
    FAILED = "failed"

@dataclass
class Order:
    order_id: int
    gen_time: float
    restaurant: str
    dormitory: str
```

```
    deadline: float = field(init=False)
    status: OrderStatus = OrderStatus.PENDING

    def __post_init__(self):
        self.deadline = self.gen_time + config.HARD_DEADLINE

    def is_critical(self, current_time: float) -> bool:
        return (self.deadline - current_time) < config.CRITICAL_TIME_THRESHOLD

@dataclass
class Bundle:
    """Represents a batch of orders for Human Riders"""
    bundle_id: int
    orders: List[Order]

    @property
    def size(self) -> int:
        return len(self.orders)

    def is_critical(self, current_time: float) -> bool:
        # If any order in the bundle is urgent
        return min(o.deadline for o in self.orders) - current_time < config.
            CRITICAL_TIME_THRESHOLD
```

Listing 2: entities.py

A.3 Demand Generator (generator.py)

Implements the Non-homogeneous Poisson Process (NHPP) and Stratified Sampling logic.

```
import numpy as np
import config
from entities import Order

class DemandGenerator:
    def __init__(self, seed: int = config.RANDOM_SEED):
        self.rng = np.random.RandomState(seed)

    def generate_orders(self) -> list:
        all_orders = []
        for start, end, count in config.DEMAND_INTERVALS:
            # NHPP: Generate arrival times
            times = self._generate_nhpp_arrivals(start, end, count)
            # Spatial: Stratified Sampling
            restaurants = self._stratified_sample(config.
                RESTAURANT_DISTRIBUTION, count)
            dorms = self._stratified_sample(config.DORMITORY_DISTRIBUTION,
                count)

            for t, r, d in zip(times, restaurants, dorms):
                all_orders.append(Order(len(all_orders), t, r, d))

        all_orders.sort(key=lambda x: x.gen_time)
```

```
        return all_orders

    def _generate_nhpp_arrivals(self, start, end, n):
        times = self.rng.uniform(start, end, n)
        times.sort()
        return times
```

Listing 3: generator.py

A.4 The Two-Stage Heuristic Dispatcher (solver.py)

Core Algorithm: This module implements the Bundling and Utility-based Assignment logic.

```
class Dispatcher:
    def dispatch(self, current_time, pending_orders, available_agents):
        # Stage 1: Clustering / Batching
        bundles = self._create_bundles(pending_orders, current_time)

        # Sort by urgency
        bundles.sort(key=lambda b: b.get_remaining_time(current_time))

        assignments = []
        # Stage 2: Greedy Utility Assignment
        for bundle in bundles:
            agent = self._assign_bundle(bundle, current_time, available_agents
                )
            if agent:
                assignments.append((bundle, agent))
        return assignments

    def _create_bundles(self, orders, current_time):
        """Groups orders by Restaurant and OD proximity"""
        bundles = []
        # Group by Restaurant...
        # Check deadline proximity...
        # Limit by Human Capacity (5)...
        return bundles

    def _assign_bundle(self, bundle, current_time, agents):
        """
        Logic:
        1. Critical & Single -> Drone (if no-fly zone allows)
        2. Large Bundle -> Human Rider
        3. Small/Normal -> AGV (Cost efficient) -> Human (Fallback)
        """
        if bundle.is_critical(current_time) and bundle.size == 1:
            return self._find_best_drone(bundle, agents) or \
                   self._find_best_agv(agents) or \
                   self._find_best_human(agents)
        elif bundle.size >= config.LARGE_BUNDLE_THRESHOLD:
            return self._find_best_human(agents)
        else:
```

```
            return self._find_best_agv(agents) or self._find_best_human(agents
                )
```

Listing 4: solver.py

A.5 Discrete Event Simulator (simulator.py)

Implements the Rolling Horizon Control (RHC) loop.

```
class Environment:
    def run(self):
        while self.current_time <= config.EXTENDED_END:
            # 1. Reveal: Add new orders
            self._reveal_orders()

            # 2. Check: Update agent status (return trips)
            self._update_agent_status()

            # 3. Dispatch: RHC Trigger (Every 5 mins)
            if self.current_time % config.RHC_INTERVAL == 0:
                self._dispatch()
            # Emergency Trigger
            elif self._has_critical_orders() and self._has_available_agents():
                self._dispatch_urgent()

            # 4. Step: Advance time
            self.current_time += self.time_step

    def _dispatch(self):
        # Call the Two-Stage Heuristic
        assignments = self.dispatcher.dispatch(
            self.current_time, self.pending_pool, self.agents
        )
        # Execute assignments...
```

Listing 5: simulator.py

A.6 Main Entry Point (main.py)

The execution script that orchestrates the simulation steps.

```
if __name__ == "__main__":
    # Step 1: Generate Demand (NHPP)
    generator = DemandGenerator(seed=config.RANDOM_SEED)
    orders = generator.generate_orders()

    # Step 2: Run Simulation (RHC)
    env = Environment(orders=orders)
    results = env.run()

    # Step 3: Analysis & Visualization
    analyzer = SimulationAnalyzer(results)
    analyzer.generate_all_plots(save_path="./output")
```

```
13
14      # Optional: Sensitivity Analysis
15      # run_sensitivity_analysis()
```

Listing 6: main.py

Report on Use of AI

OpenAI-ChatGPT (GPT-4o)

Query: How to model stochastic order arrivals with limited data points?
Output: Suggested using a Non-homogeneous Poisson Process (NHPP) and fitting the intensity function $\lambda(t)$ using interpolation, rather than using deep learning models like LSTM which require large datasets.

Query: `scipy.optimize` usage for vehicle routing?
Output: The model explained that standard libraries are insufficient for dynamic VRPTW and suggested writing a custom heuristic or using OR-Tools. We implemented the Two-Stage Heuristic based on this advice.